AWS Well-Architected Framework

Machine Learning Lens

Machine Learning Lens: AWS Well-Architected Framework

Table of Contents

Machine Learning Lens - AWS Well-Architected Framework

Publication date: **November 19, 2025** (*Document revisions*)

Machine learning (ML) has evolved from research and development to the mainstream, driven by the exponential growth of data sources, generative AI and scalable cloud-based compute resources. AWS customers use AI/ML for a wide variety of applications, ranging from foundation model development and fine-tuning to sophisticated computer vision implementations. Common use cases include call center operations, personalized recommendations, fraud detection, social media content moderation, audio and video content analysis, product design services, and identity verification. These applications use both custom-built models and pre-trained solutions to address specific business needs. AI/ML adoption has become common across nearly every industry, including healthcare and life sciences, automotive, industrial and manufacturing, financial services, media and entertainment, and telecommunications.

Machine learning harnesses algorithms to discover patterns in data, delivering considerable value to customers while requiring responsible implementation. AWS is committed to developing fair and accurate AI and ML services and providing you with the tools and guidance needed to build AI and ML applications responsibly.

This whitepaper provides you with a set of best practices. You can apply this guidance and architectural principles when designing your ML workloads and after your workloads have entered production as part of continuous improvement. Although the guidance is cloud- and technology-agnostic, the paper also includes guidance and resources to assist you to implement these best practices on AWS.

Introduction

The AWS Well-Architected Framework assists you to understand the benefits and risks of decisions made while building workloads on AWS. Through the Framework, you learn operational and architectural best practices for designing and operating cloud workloads. It provides a consistent method to measure your operations and architectures against best practices and identify improvement opportunities.

Your ML models depend on high-quality input data to generate accurate results. As data evolves over time, continuous monitoring is essential to detect, correct, and mitigate accuracy and performance issues, often requiring model retraining with refined datasets.

While application workloads rely on deterministic, step-by-step instructions to solve problems, ML workloads enable algorithms to learn from data through iterative and continuous cycles. The ML Lens complements and builds upon the Well-Architected Framework to address the fundamental differences between traditional application workloads and machine learning workloads.

This paper is intended for those in a technology role, such as chief technology officers (CTOs), architects, developers, data scientists, and ML engineers. After reading this paper, you will understand the best practices and strategies to use when you design and operate ML workloads on AWS.

Distinction from the Generative AI Lens

The Machine Learning Lens addresses the broad spectrum of ML workloads, including traditional supervised and unsupervised learning, predictive analytics, classification, regression, and clustering tasks. Common ML use cases covered by this lens include computer vision for object detection and image classification, fraud detection and risk scoring, recommendation engines, predictive maintenance, demand forecasting, customer churn prediction, anomaly detection, and medical diagnosis systems. In contrast, the Generative AI Lens focuses on foundation models and generative AI applications that create content, such as text generation, image synthesis, and conversational AI systems.

While both lenses share common ML principles, the Generative AI Lens emphasizes prompt engineering, foundation model selection, retrieval-augmented generation (RAG) architectures, and the specific governance challenges of generative AI systems. The Machine Learning Lens provides comprehensive guidance for the full ML lifecycle across ML paradigms, making it the foundational lens for ML workloads.

Lens availability

Custom lenses extend the best practice guidance provided by AWS Well-Architected Tool. AWS WA Tool allows you to create your own custom lenses, or to use lenses created by others that have been shared with you.

To begin reviewing your machine learning workload, download and import the Machine Learning Lens into AWS Well-Architected Tool from the public AWS Well-Architected custom lens GitHub repository.

Well-Architected machine learning design principles

Well-Architected ML design principles are a set of considerations used as the basis for a well-architected ML workload.

Following the Well-Architected Framework guidelines, use these general design principles to facilitate good design in the cloud for ML workloads:

- **Assign ownership:** Apply the right skills and the right number of resources along with accountability and empowerment to increase productivity.
- **Provide protection:** Apply security controls to systems and services hosting model data, algorithms, computation, and endpoints. This facilitates secure and uninterrupted operations.
- **Enable resiliency:** Verify fault tolerance and the recoverability of ML models through version control, traceability, and explainability.
- **Enable reusability:** Use independent modular components that can be shared and reused. This assists to enable reliability, improve productivity, and optimize cost.
- **Enable reproducibility:** Use version control across components, such as infrastructure, data, models, and code. Track changes back to a point-in-time release. This approach enables model governance and audit standards.
- **Optimize resources:** Perform trade-off analysis across available resources and configurations to achieve optimal outcome.
- **Reduce cost:** Identify the potentials for reducing cost through automation or optimization, analyzing processes, resources, and operations.
- **Enable automation:** Use technologies, such as pipelining, scripting, and continuous integration (CI), continuous delivery (CD), and continuous training (CT), to increase agility, improve performance, sustain resiliency, and reduce cost.
- **Enable continuous improvement:** Evolve and improve the workload through continuous monitoring, analysis, and learning.
- **Minimize environmental impact:** Establish sustainability goals and understand the impact of ML models. Use managed services and adopt efficient hardware and software and maximize their utilization.

Well-Architected machine learning lifecycle

The ML lifecycle is the cyclic iterative process with instructions and best practices to use across defined phases while developing an ML workload. The ML lifecycle adds clarity and structure for making a machine learning project successful. The end-to-end machine learning lifecycle process illustrated in Figure 1 includes the following phases:

- Business goal identification
- ML problem framing
- Data processing (data collection, data preprocessing, feature engineering)
- Model development (training, tuning, evaluation)
- Model deployment (inference, prediction)
- Model monitoring

The phases of the ML lifecycle are not necessarily sequential in nature and can have feedback loops, a few of which are illustrated in Figure 1, to interrupt the cycle across the lifecycle phases.

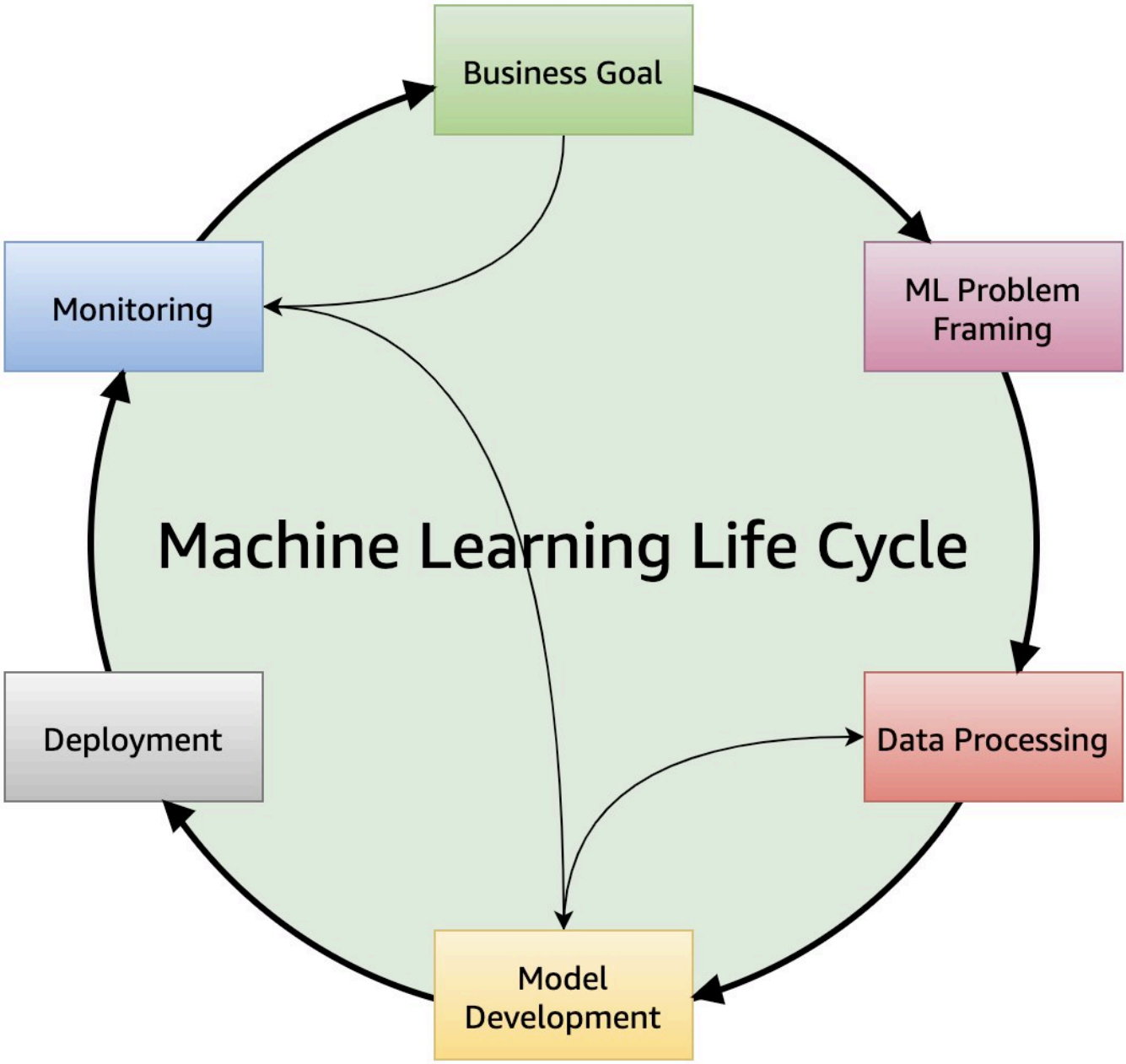

Figure 1: Machine learning lifecycle

The following is a quick introduction to each phase, which will be expanded upon later in this paper.

Business goal

An organization considering ML should have a clear idea of the problem, and the business value to be gained by solving that problem. You must be able to measure business value against specific business objectives and success criteria.

ML problem framing

In this phase, the business problem is framed as a machine learning problem: what is observed and what should be predicted (known as a label or target variable). Determining what to predict and how performance and error metrics must be optimized is a key step in this phase.

Data processing

Training an accurate ML model requires data processing to convert data into a usable format. Data processing steps include collecting data, preparing data, and feature engineering that is the process of creating, transforming, extracting, and selecting variables from data.

Model development

Model development consists of model building, training, tuning, and evaluation. Model building includes creating a CI/CD pipeline that automates the build, train and release to staging and production environments.

Deployment

After a model is trained, tuned, evaluated and validated, you can deploy the model into production. You can then make predictions and inferences against the model.

Monitoring

Model monitoring system verifies your model is maintaining a desired level of performance through early detection and mitigation.

Well-Architected machine learning

The six phases for the ML lifecycle referenced in this lens are illustrated in Figure 2 in a sequence.

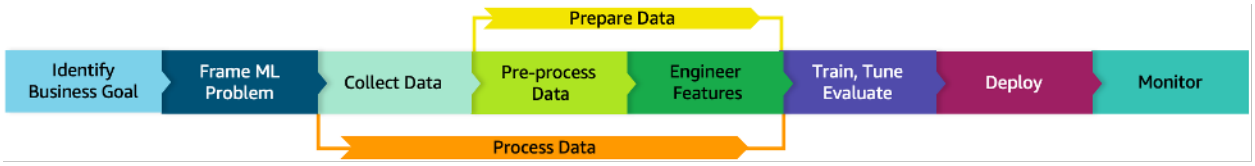

Figure 2: Machine learning lifecycle

The following sections describe Well-Architected machine learning for each of the lifecycle phases.

Lifecycle phases

- Business goal identification
- ML problem framing
- ML lifecycle architecture diagram
- Data processing
- Model development
- Deployment
- Monitoring

Business goal identification

Business goal identification is the most important phase of the ML lifecycle. An organization considering ML should have a clear idea of the problem to be solved, and the business value to be gained. You must be able to measure business value against specific business objectives and success criteria. While this holds true for technical solutions, this step is particularly challenging when considering ML solutions because ML is a constantly evolving technology.

After you determine your criteria for success, evaluate your organization's ability to move toward that target. The target should be achievable and provide a clear path to production. Involve relevant stakeholders from the beginning to align them to this target and new business processes that result from this initiative.

Start the review by determining if ML is the appropriate approach for delivering your business goal. Evaluate the options that you have available for achieving the goal. Determine how accurate the resulting outcomes would be, while considering the cost and scalability of each approach.

For an ML-based approach to be successful, verify that enough relevant, high-quality training data is available to the algorithm. Carefully evaluate the data to make sure that the correct data sources are available and accessible.

Steps in this phase:

The following work steps should be followed to establish your business goals.

- Business considerations:
  - Understand business requirements.
  - Align affected stakeholders with this initiative.
  - Form a business question.
  - Identify critical, must-have features.
  - Consider new business processes that might come out of this implementation.
  - Consider how business value can be measured using business metrics that the ML model can assist to improve.
- Frame the ML problem:
  - Define the machine learning task based on the business question.
  - Review proven or published works in similar domains, if available.
  - Design small, focused POCs to validate those aspects of the approach where inadequate confidence exists.
- Determine the optimization objective:
  - Determine key business performance metrics for the ML use case, such as uplift in new business acquisition, fraud detection rate, and anomaly detection. Increase CSAT according to the business needs.
- Review data requirements:
  - Review the project's ML feasibility and data requirements.
- Cost and performance optimization:
  - Evaluate the cost of data acquisition, training, inference, and wrong predictions.
  - Evaluate whether bringing in external data sources might improve model performance.
- Production considerations:
  - Review how to handle ML-generated errors.
  - Establish pathways to production.

ML problem framing

In this phase, the business problem is framed as a machine learning problem: what is observed and what should be predicted (known as a label or target variable). Determining what to predict and how performance must be optimized is a key step in ML. For example, consider a scenario where a manufacturing company wants to maximize profits. There are several possible approaches including forecasting sales demand for existing product lines to optimize output, forecasting the required input materials and components required to reduce capital locked up in stock, and predicting sales for new products to prioritize new product development.

It's necessary to work through framing the ML problems in line with the business challenge.

Steps in this phase:

- Define criteria for a successful outcome of the project.
- Establish an observable and quantifiable performance metric for the project, such as accuracy.
- Define the relationship between the technical metric (for example, accuracy) and the business outcome (for example, sales).
- Assist to verify business stakeholders understand and agree with the defined performance metrics.
- Formulate the ML question in terms of inputs, desired outputs, and the performance metric to be optimized.
- Evaluate whether ML is the right approach. Some business problems don't need ML as simple business rules can do a much better job. For other business problems, there might not be sufficient data to apply ML as a solution.
- Create a strategy to achieve the data sourcing and data annotation objective.
- Start with a model that is simple to interpret and makes debugging more manageable.
- Map the technical outcome to a business outcome.
- Iterate on the model by gathering more data, optimizing the parameters, or increasing the complexity as needed to achieve the business outcome.

ML lifecycle architecture diagram

Figure 3 shows the ML lifecycle phases with the *data processing phase* (for example, Process Data) expanded into a *data collection sub-phase* (Collect Data) and a *data preparation sub-phase* (Pre-process Data and Engineer Features). These sub-phases are discussed in more detail in this section.

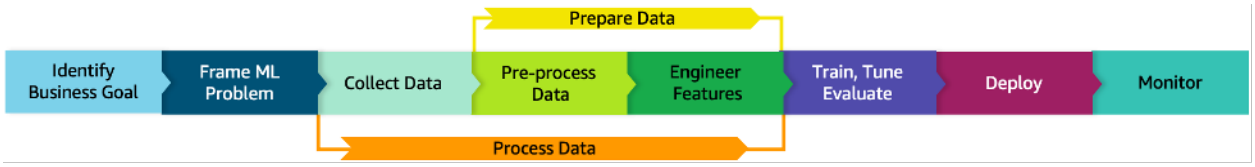

Figure 3: Lifecycle data with data processing sub-phases included

Figure 4 illustrates the details of the ML lifecycle phases that occur following the problem framing phase and shows how the data-processing sub-phases interact with the subsequent phases, that is, the *model development phase*, the *model deployment phase*, and the *model monitoring phase*.

The model development phase includes training, tuning, and evaluation. The model deployment phase includes the staging environment for model validation for security and robustness. Monitoring is key in timely detection and mitigation of drifts. Feedback loops across the ML lifecycle phases are key enablers for monitoring. Feature stores (both online and offline) provide consistent and reusable features across model development and deployment phases. The model registry enables the version control and lineage tracking for model and data components. This figure also emphasizes the lineage tracking and its components that are discussed in this section in more detail.

The cloud agnostic architecture diagrams in this paper provide high-level best practices with the following assumptions:

- All concepts presented here are cloud and technology agnostic.
- Solid black lines are indicative of process flow.
- Dashed color lines are indicative of input and output flow.
- Architecture diagram components are color-coded for ease of communication across this document.

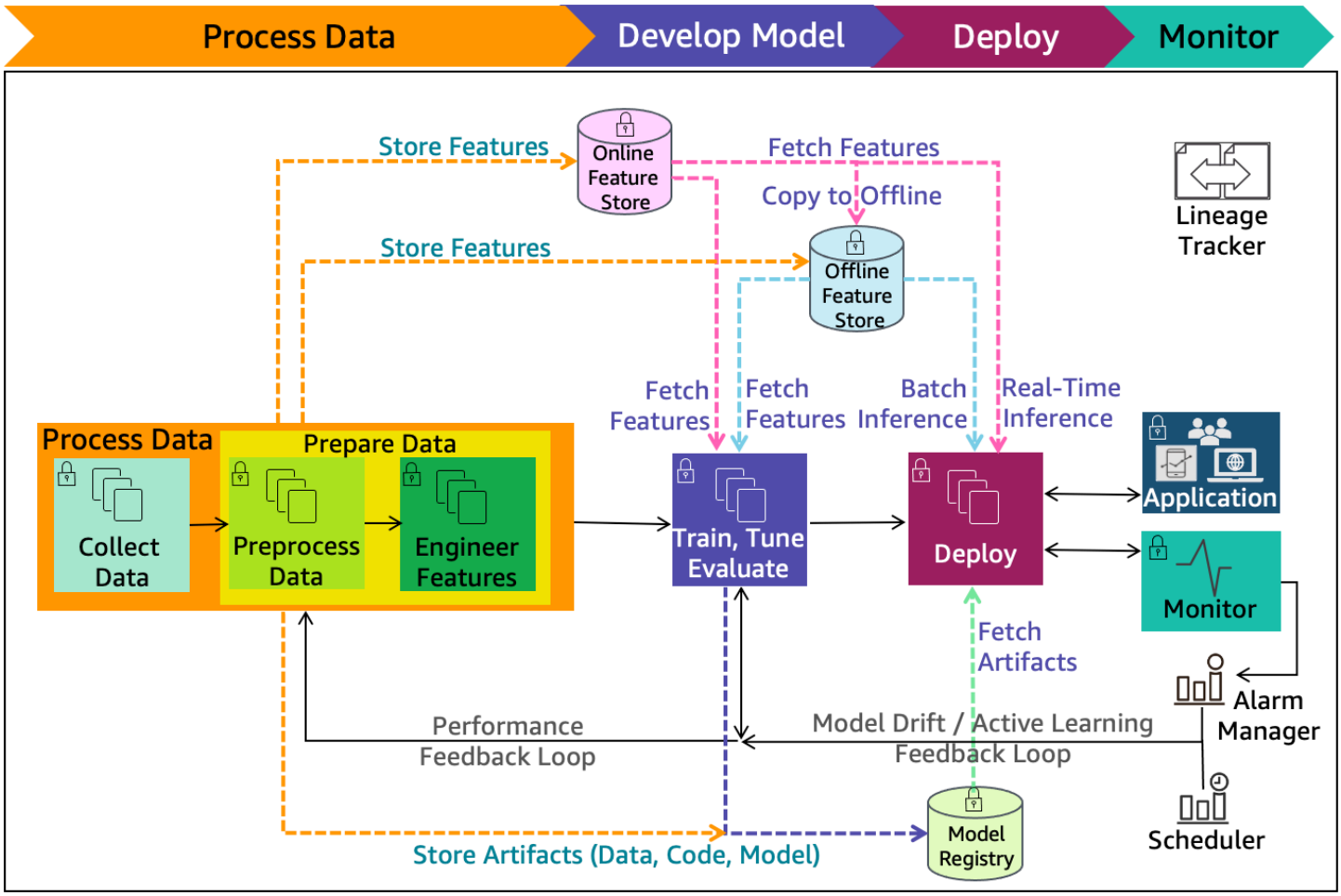

Figure 4: ML lifecycle with detailed phases and extended components

The components of the sub-phases of the ML lifecycle shown in Figure 4 are as follows:

- **Online/offline feature store:** Reduces duplication and the need to rerun feature engineering code across teams and projects. An online store with low-latency retrieval capabilities is ideal for real-time inference. On the other hand, an offline store is designed for maintaining a history of feature values and is suited for training and batch scoring.
- **Model registry:** A repository for storing ML model artifacts including trained model and related metadata (such as data, code, and model). It enables the tracking of the lineage of ML models as it can act as a version control system.
- **Performance feedback loop:** Informs the iterative data preparation phase based on the evaluation of the model during the model development phase.
- **Model drift feedback loop:** Informs the iterative data preparation phase based on the evaluation of the model during the production deployment phase.

- **Alarm manager:** Receives alerts from the model monitoring system. It then publishes notifications to the services that can deliver alerts to target applications. The model update re-training pipeline is one such target application.
- **Scheduler:** Initiates a model re-training at business-defined intervals.
- **Lineage tracker:** Enables reproducible machine learning experiences. It enables the re-creation of the ML environment at a specific point in time, reflecting the versions of resources and environments at that time.

The ML lineage tracker collects references to traceable data, model, and infrastructure resource changes. It consists of the following components:

- System architecture (infrastructure as code to address environment drift)
- Data (metadata, values, and features)
- Model (algorithm, features, parameters, and hyperparameters)
- Code (implementation, modeling, and pipeline)

The lineage tracker collects changed references through alternative iterations of ML lifecycle phases. Alternative algorithms and feature lists are evaluated as experiments for final production deployment.

Figure 5 includes machine learning components and their information that the lineage tracker collects across different releases. The collected information enables going back to a specific point-in-time release and re-creating it.

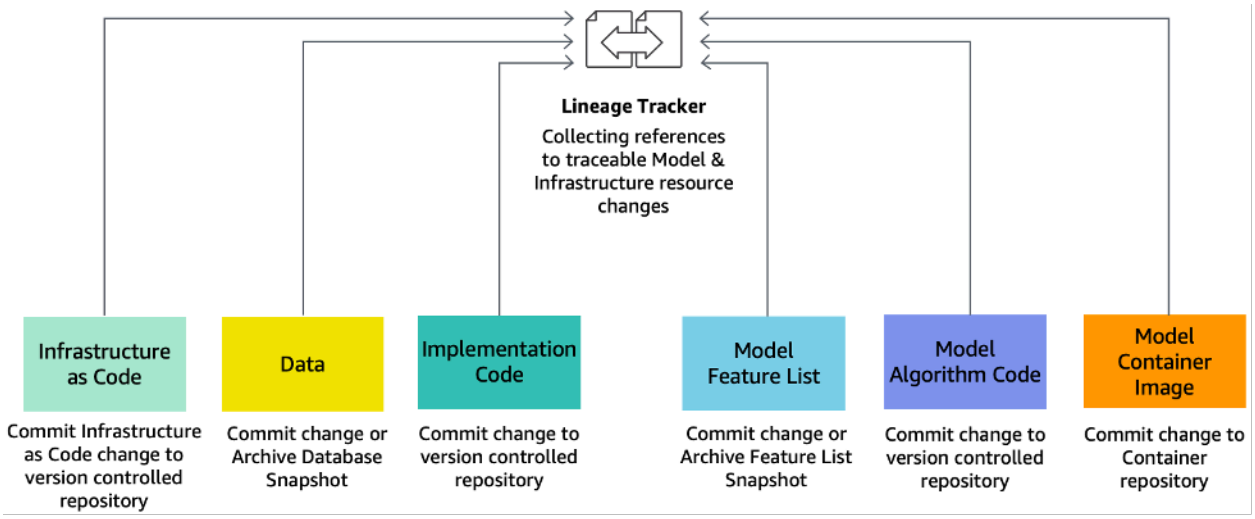

Figure 5: Lineage tracker

Lineage tracker components include:

- **Infrastructure as code (IaC):** Modeling, provisioning, and managing cloud computing resources (compute, storage, network, and application services) can be automated using infrastructure as code. Cloud computing takes advantage of virtualization to enable the on-demand provisioning of resources. IaC avoids configuration drift through automation, while increasing the speed and agility of infrastructure deployments. IaC code changes are committed to version-controlled repository.
- **Data:** Store data schemes and metadata in version control systems. Store the data in a storage media, such as a data lake. The location or link to the data can be in a configuration file and stored in code version control media.
- **Implementation code:** Changes to implementation code can be stored as point-in-time using version control media.
- **Model feature list:** A *feature store*, discussed earlier in this section (Figure 4), maintains the details of the features as well as their previous versions for any point-in-time changes.
- **Model algorithm code:** Changes to model algorithm code at specific points-in-time can be stored in version control media.
- **Model container image:** Versions of model container images for specific point-in-time changes can be stored in container repositories managed by container registry.

Data processing

In ML workloads, the data (inputs and corresponding desired output) serves important functions including:

- Defining the goal of the system: the output representation and the relationship of each output to each input, by means of the input and output pairs.
- Training the algorithm that associates inputs to outputs.
- Measuring the performance of the model against changes in data distribution or data drift.
- Building a baseline dataset to capture data drift.

As shown in Figure 6, data processing consists of data collection and data preparation. Data preparation includes data preprocessing and feature engineering. It mainly uses data wrangling for interactive data analysis and data visualization for exploratory data analysis (EDA). EDA focuses on understanding data, sanity checks, and validation of data quality.

It is important to note that the same sequence of data processing steps that is applied to the training data needs to also be applied to the inference requests.

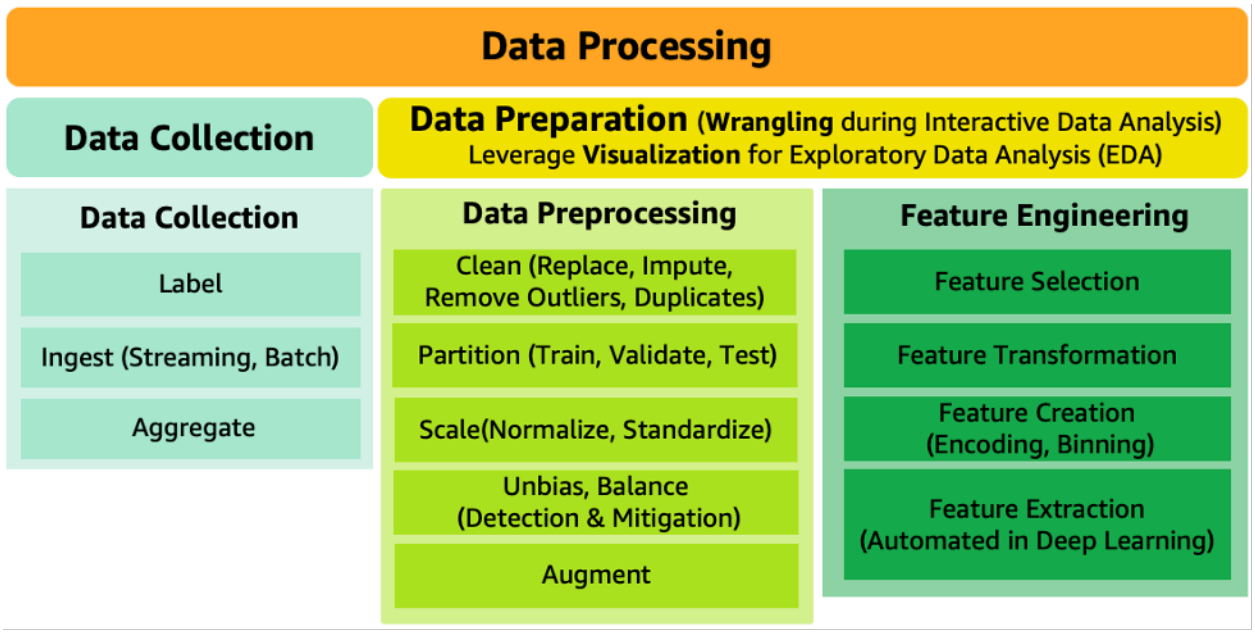

Figure 6: Data processing components

Sub-phases

- Data collection
- Data preparation

Data collection

Important steps in the ML lifecycle are to identify the data needed, followed by the evaluation of the various means available for collecting that data to train your model.

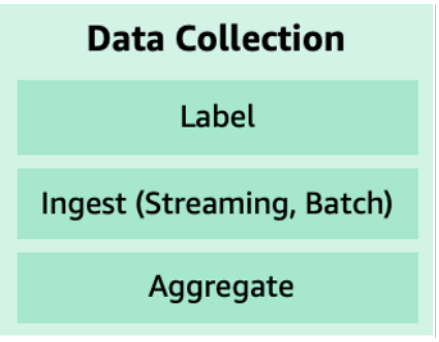

Figure 7: Main components of data collection

- **Label:** *Labeled data* is a group of samples that have been tagged with one or more labels. If labels are missing, then some effort is required to label it (either manual or automated).
- **Ingest and aggregate:** Data collection includes ingesting and aggregating data from multiple data sources.

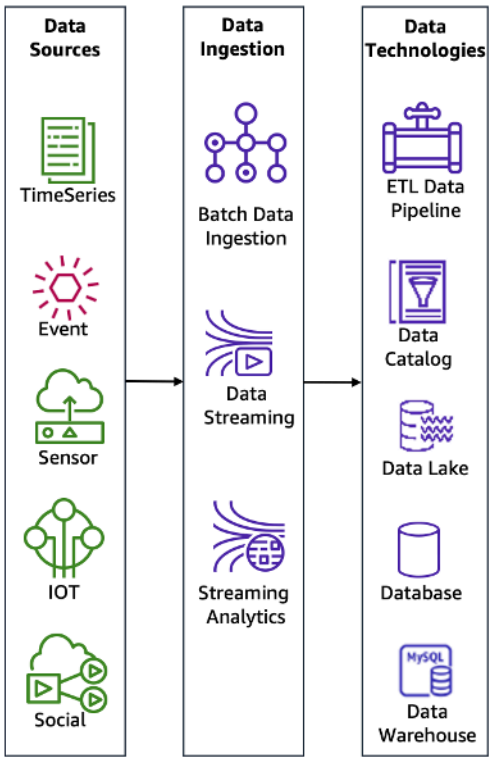

Figure 8: Data sources, data ingestion, and data technologies

The sub-components of the *ingest and aggregate* component (shown in Figure 8) are as follows:

- **Data sources:** Data sources include time-series, events, sensors, IoT devices, and social networks, depending on the nature of the use case. You can enrich your data sources by using the geospatial capability of Amazon SageMaker AI to access a range of geospatial data sources from AWS (for example, Amazon Location Service), open-source datasets (for example, Open Data on AWS), or your own proprietary data including from third-party providers (such as Planet Labs). To learn more about the geospatial capability in Amazon SageMaker AI, visit Geospatial ML with Amazon SageMaker AI.
- **Data ingestion:** Data ingestion processes and technologies capture and store data on storage media. Data ingestion can occur in real-time using streaming technologies or historical mode using batch technologies.

- **Data technologies:** Data storage technologies vary from transactional (SQL) databases, to data lakes and data warehouses to form a lake house with marketplace governance across teams and partners. Extract, transform, and load (ETL) pipeline technology automates and orchestrates the data movement and transformations across cloud services and resources. A lake house enables storing and analyzing structured and unstructured data.

Data preparation

ML models are only as good as the data that is used to train them. Verify that suitable training data is available and is optimized for learning and generalization. Data preparation includes data preprocessing and feature engineering.

A key aspect to understanding data is to identify patterns. These patterns are often not evident with data in tables. Exploratory data analysis (EDA) with visualization tools can assist in quickly gaining a deeper understanding of data. Prepare data using data wrangler tools for interactive data analysis and model building. Employ no-code/low-code, automation, and visual capabilities to improve the productivity and reduce the cost for interactive analysis. Use generative AI code tools.

Sub-phases

- Data preprocessing
- Feature engineering

Data preprocessing

Data preprocessing puts data into the right shape and quality for training. There are many data preprocessing strategies including: data cleaning, balancing, replacing, imputing, partitioning, scaling, augmenting, and unbiasing.

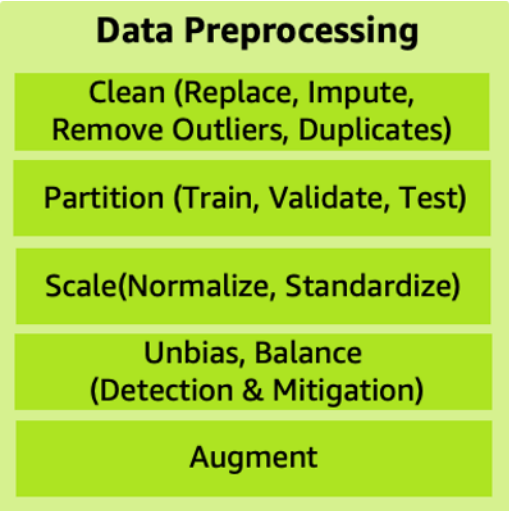

Figure 9: Data preprocessing main components

The data preprocessing strategies listed in Figure 9 can be expanded as the following:

- **Clean (replace, impute, remove outliers and duplicates):** Remove outliers and duplicates, replace inaccurate or irrelevant data, and correct missing data using imputation techniques that will minimize bias as part of data cleaning.
- **Partition:** To block ML models from overfitting and to evaluate a trained model accurately, randomly split data into train, validate, and test sets. Data leakage can happen when information from hold-out test dataset leaks into the training data. One way to avoid data leakage is to remove duplicates before splitting the data.
- **Scale (normalize, standardize):** Normalization is a scaling technique in machine learning that is applied during data preparation to change the values of numeric columns in the dataset to use a common scale. This technique assists to verify that each feature of the machine learning model has equal feature importance when they have different ranges. Normalized numeric features will have values in the range of [0,1]. Standardized numeric features will have a mean of 0 and standard deviation of 1. Standardization assists in handling outliers.
- **Unbias, balance (detection and mitigation):** Detecting and mitigating bias assists to avoid inaccurate model results. Biases are imbalances in the accuracy of predictions across different groups, such as age or income bracket. Biases can come from the data or algorithm used to train your model.
- **Augment:** Data augmentation increases the amount of data artificially by synthesizing new data from existing data. Data augmentation can assist to regularize and reduce overfitting.

Feature engineering

Every unique attribute of the data is considered a *feature* (also known as an *attribute*). For example, when designing a solution for predicting customer churn, the data used typically includes features such as customer location, age, income level, and recent purchases.

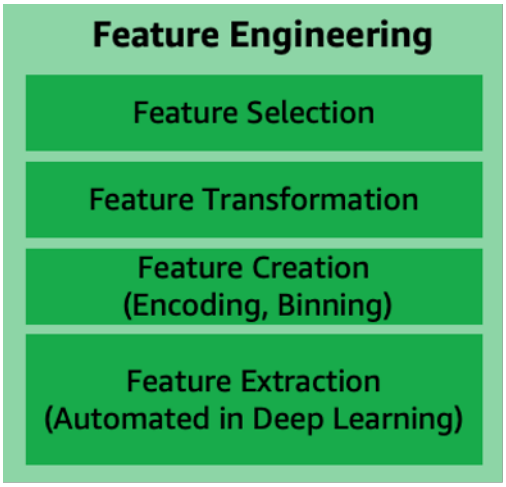

Figure 10: Feature engineering main components

Feature engineering is a process to select and transform variables when creating a predictive model using machine learning or statistical modeling. Feature engineering typically includes feature creation, feature transformation, feature extraction, and feature selection as listed in Figure 10. With deep learning, feature engineering is automated as part of the algorithm learning.

- *Feature creation* refers to the creation of new features from existing data to assist with better predictions. Examples of feature creation include one-hot-encoding, binning, splitting, and calculated features.
- *Feature transformation and imputation* include steps for replacing missing features or features that are not valid. Some techniques include forming Cartesian products of features, non-linear transformations (such as binning numeric variables into categories), and creating domain-specific features.
- *Feature extraction* involves reducing the amount of data to be processed using dimensionality reduction techniques. These techniques include Principal Components Analysis (PCA), Independent Component Analysis (ICA), and Linear Discriminant Analysis (LDA). This reduces the amount of memory and computing power required, while still accurately maintaining original data characteristics.
- *Feature selection* is the process of selecting a subset of extracted features. This is the subset that is relevant and contributes to minimizing the error rate of a trained model. Feature importance

score and correlation matrix can be factors in selecting the most relevant features for model training.

Model development

Model development consists of model building, training, tuning, and evaluation.

Sub-phases

- Model training and tuning
- Model evaluation

Model training and tuning

In this phase, you select a machine learning algorithm that is appropriate for your problem and then train the ML model. You provide the algorithm with the training data, set an objective metric for the ML model to optimize on, and set the hyperparameters to optimize the training process.

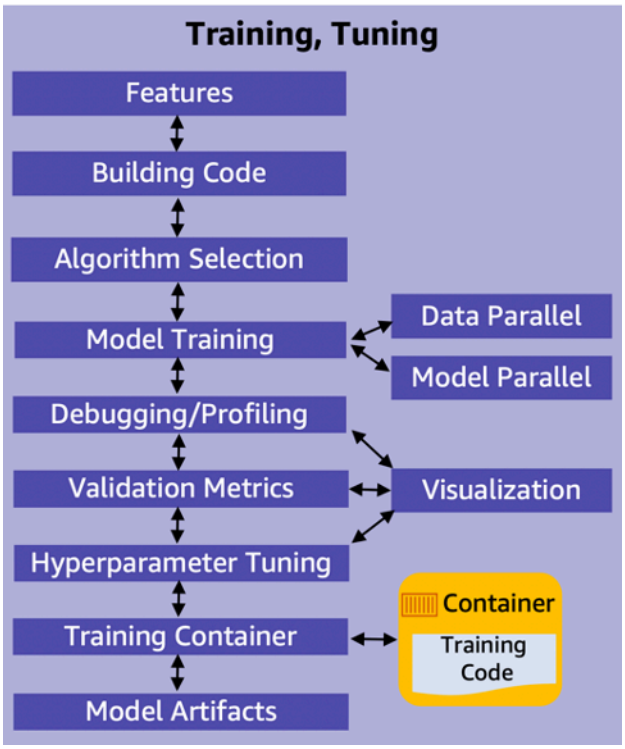

Figure 11: ML model training and tuning main components

Model training, tuning, and evaluation require prepared data and engineered features. The following are the main activities in this stage, as listed in Figure 11:

- **Features:** Features are selected as part of the data processing after data quality is assured
- **Building code:** Model development includes building the algorithm and its supporting code. The code-building process should support version control and continuous build, test, and integration through a pipeline.
- **Algorithm selection:** Selecting the right algorithm involves running many experiments with parameter tuning across available options. Factors to consider when evaluating each option can include success metrics, model explainability, and compute requirements (training/prediction time and memory requirements).
- **Model training (data parallel, model parallel):** The process of training a ML model involves providing the algorithm with training data to learn from. Distributed training enables splitting large models and training datasets across compute instances to reduce training time significantly. Model parallelism and data parallelism are techniques to achieve distributed training.
- **Model parallelism** is the process of splitting a model up between multiple instances or nodes.
- **Data parallelism** is the process of splitting the training set in mini-batches evenly distributed across nodes. Thus, each node only trains the model on a fraction of the total dataset.
- **Debugging or profiling:** A machine learning training job can have problems including: system bottlenecks, overfitting, saturated activation functions, and vanishing gradients. These problems can compromise model performance. A debugger provides visibility into the ML training process through monitoring, recording, and analyzing data. It captures the state of a training job at periodic intervals.
- **Validation metrics:** Typically, a training algorithm computes several metrics such as loss and prediction accuracy. These metrics determine if the model is learning and generalizing well for making predictions on unseen data. Metrics reported by the algorithm depend on the business problem and the ML technique used. For example, a *confusion matrix* is one of the metrics used for classification models, and Root Mean Squared Error (RMSE) is one of the metrics for regression models.
- **Hyperparameter tuning:** Settings that can be tuned to control the behavior of the ML algorithm are referred to as *hyperparameters*. The number and type of hyperparameters in ML algorithms are specific to each model. Examples of commonly used hyperparameters include: learning rate, number of epochs, hidden layers, hidden units and activation functions. Hyperparameter tuning, or optimization, is the process of choosing the optimal hyperparameters for an algorithm.
- **Training code container:** Create container images with your training code and its entire dependency stack. This will enable the training and deployment of models quickly and reliably at scale.

- **Model artifacts:** Model artifacts are the outputs that results from training a model. They typically consist of trained parameters, a model definition that describes how to compute inferences, and other metadata.
- **Visualization:** Enables exploring and understanding data during metrics validation, debugging, profiling, and hyperparameter tuning.

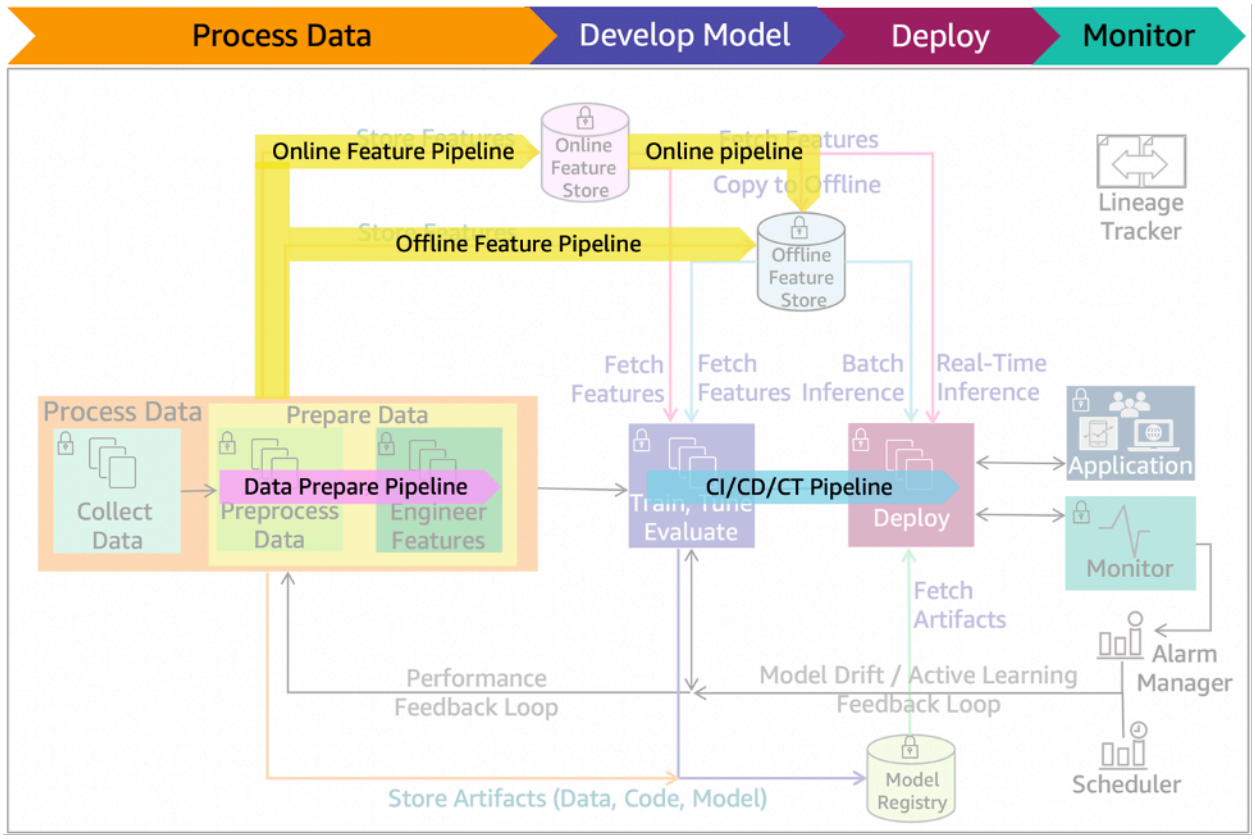

Figure 12: ML lifecycle with pre-production pipelines

Figure 12 shows the pre-production pipelines. The *data prepare* pipeline automates data preparation tasks. The feature pipeline automates the storing, fetching, and copying of the features into and from online/offline store. The CI/CD/CT pipeline automates the build, train, and release to staging and production environments.

Model evaluation

After the model has been trained, evaluate it for its performance and success metrics. You might want to generate multiple models using different methods and evaluate the effectiveness of each model. You also might evaluate whether your model must be more sensitive than specific, or more specific than sensitive. For multiclass models, determine error rates for each class separately.

You can evaluate your model using historical data (offline evaluation) or live data (online evaluation). In offline evaluation, the trained model is evaluated with a portion of the dataset that has been set aside as a *holdout set*. This holdout data is never used for model training or validation, but rather to evaluate errors in the final model. The holdout data annotations must have high assigned label correctness for the evaluation to make sense. Allocate additional resources to verify the correctness of the holdout data.

Based on the evaluation results, you might fine-tune the data, the algorithm, or both. When you fine-tune the data, you apply the concepts of data cleansing, preparation and feature engineering.

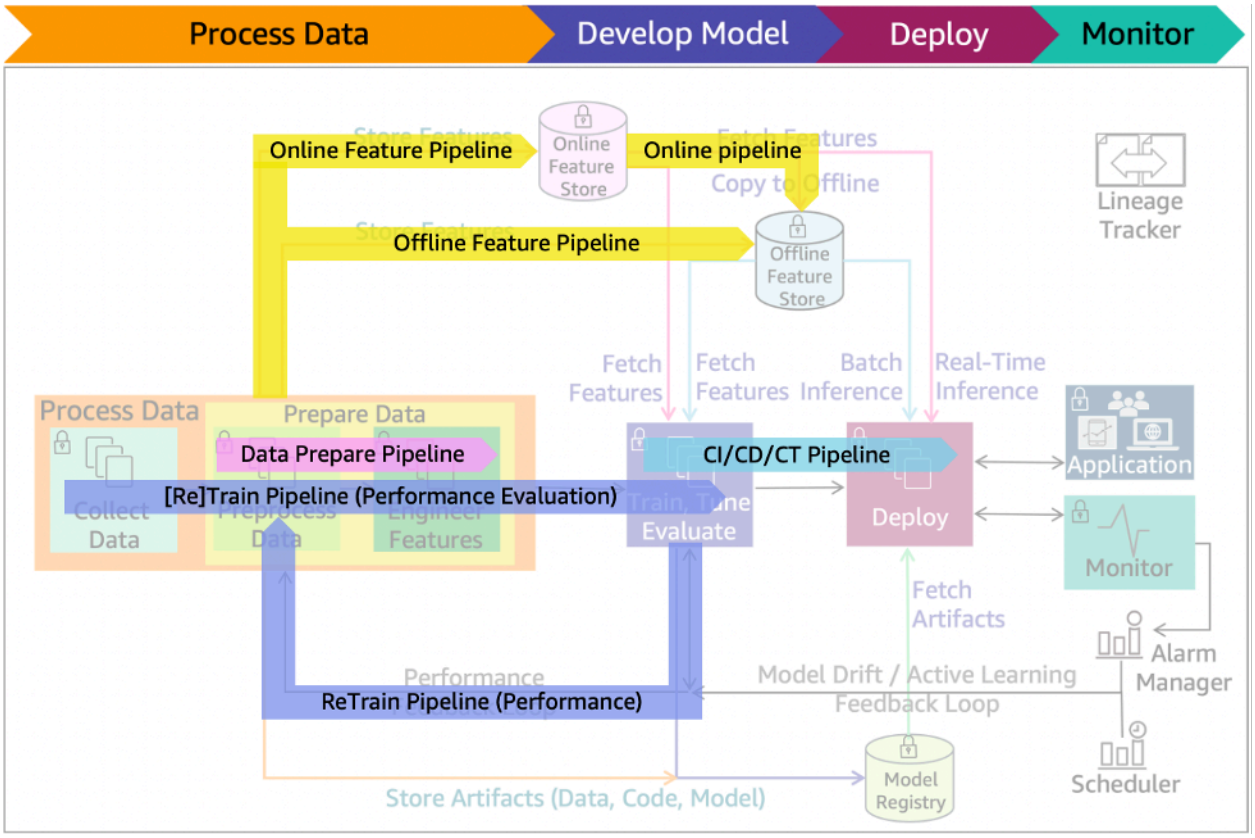

Figure 13: ML lifecycle with performance evaluation pipeline added

Figure 13 includes the model performance evaluation, the *data prepare* and CI/CD/CT pipelines that fine-tune data and algorithms, re-training, and evaluation of model results.

Deployment

After you have trained, tuned, and evaluated your model, you can deploy it into production and make predictions against this deployed model. Amazon SageMaker AI Studio can convert notebook code to production-ready jobs without the need to manage the underlying infrastructure. Be sure

to use a governance process. Controlling deployments through automation combined with manual or automated quality gates facilitates that changes can be effectively validated with dependent systems prior to deployment to production.

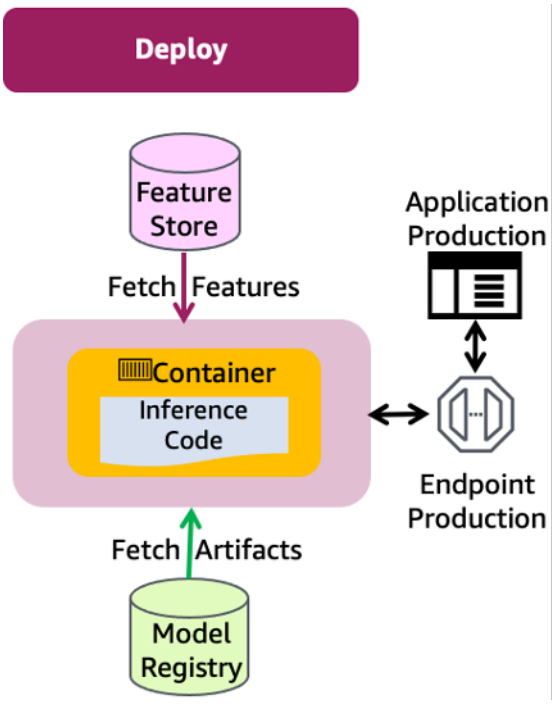

Figure 14: Deployment architecture diagram

Figure 14 illustrates the deployment phase of the ML lifecycle in production. An application sends request payloads to a production endpoint to make inference against the model. Model artifacts are fetched from the model registry, features are retrieved from the feature store, and the inference code container is obtained from the container repository.

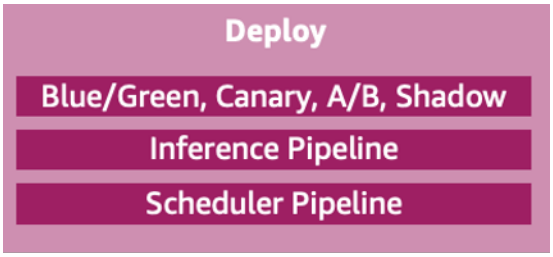

Figure 15: Deployment main components

Figure 15 lists key components of production deployment including:

- **Blue/green, canary, A/B, shadow deployment/testing:** Deployment and testing strategies that reduce downtime and risks when releasing a new or updated version.

- The *blue/green* deployment technique provides two identical production environments (initially *blue* is the existing infrastructure and *green* is an identical infrastructure for testing). Once testing is done on the *green* environment, live application traffic is directed to it from the *blue* environment. Then the roles of the blue/green environments are switched.
- With a *canary* deployment, a new release is deployed to a small group of users while other users continue to use the previous version. Once you're satisfied with the new release, you can gradually roll it out to the users.
- *A/B* testing strategy enables deploying changes to a model. Direct a defined portion of traffic to the new model. Direct the remaining traffic to the old model. A/B testing is similar to canary testing, but has larger user groups and a longer time scale, typically days or even weeks.
- With a *shadow* deployment strategy, the new version is available alongside the old version. The input data is run through both versions. The older version is used for servicing the production application and the new one is used for testing and analysis.
- **Inference pipeline:** Figure 16 shows the inference pipeline that automates capturing of the prepared data, performing predictions and post-processing for real-time or batch inferences.
- **Scheduler pipeline:** Deployed model is representative of the latest data patterns. When configured as shown in Figure 16, re-training at intervals can minimize the risk of data and concept drifts. A scheduler can initiate a re-training at business defined intervals. Data preparation, CI/CD/CT, and feature pipelines will also be active during this process.

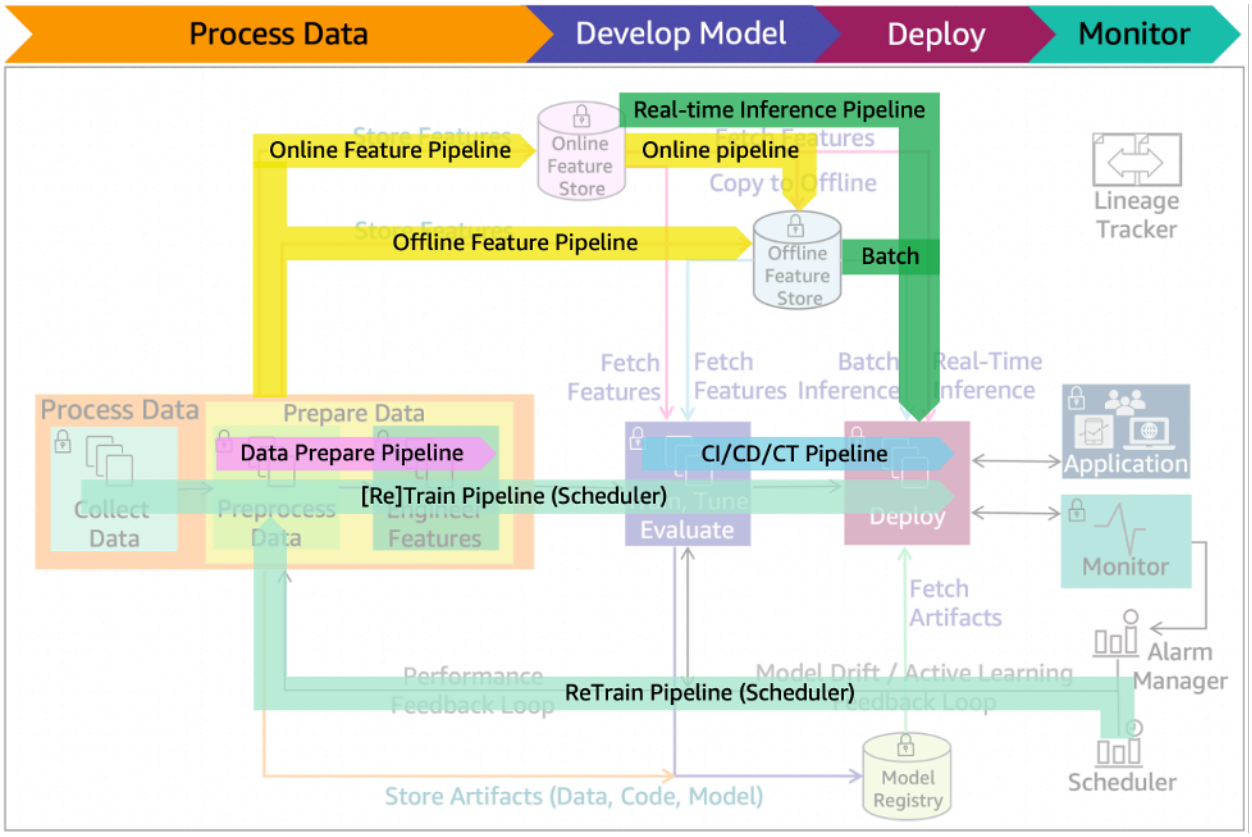

Figure 16: ML lifecycle with scheduler retrain and batch or real-time inference pipelines

Monitoring

The model monitoring system must capture data, compare that data to the training set, define rules to detect issues, and send alerts. This process repeats on a defined schedule, when initiated by an event, or when initiated by human intervention. The issues detected in the monitoring phase include: data quality, model quality, bias drift, and feature attribution drift.

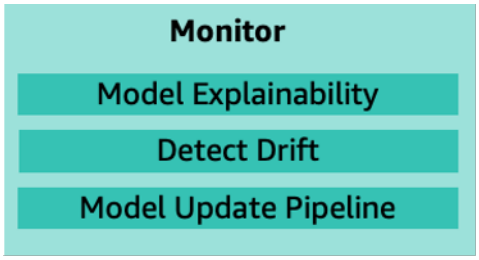

Figure 17: Post-deployment monitoring main components

Figure 17 lists key components of monitoring, including:

- **Model explainability:** Monitoring system uses *explainability* to evaluate the soundness of the model and if the predictions can be trusted.
- **Detect drift:** Monitoring system detects data and concept drifts, initiates an alert, and sends it to the alarm manager system. Data drift is significant changes to the data distribution compared to the data used for training. Concept drift is when the properties of the target variables change. Data drift can result in model performance degradation.
- **Model update pipeline:** If the alarm manager identifies violations, it launches the model update pipeline for a re-train. This can be seen in Figure 18. The *Data prepare*, *CI/CD/CT*, and *Feature* pipelines will also be active during this process.

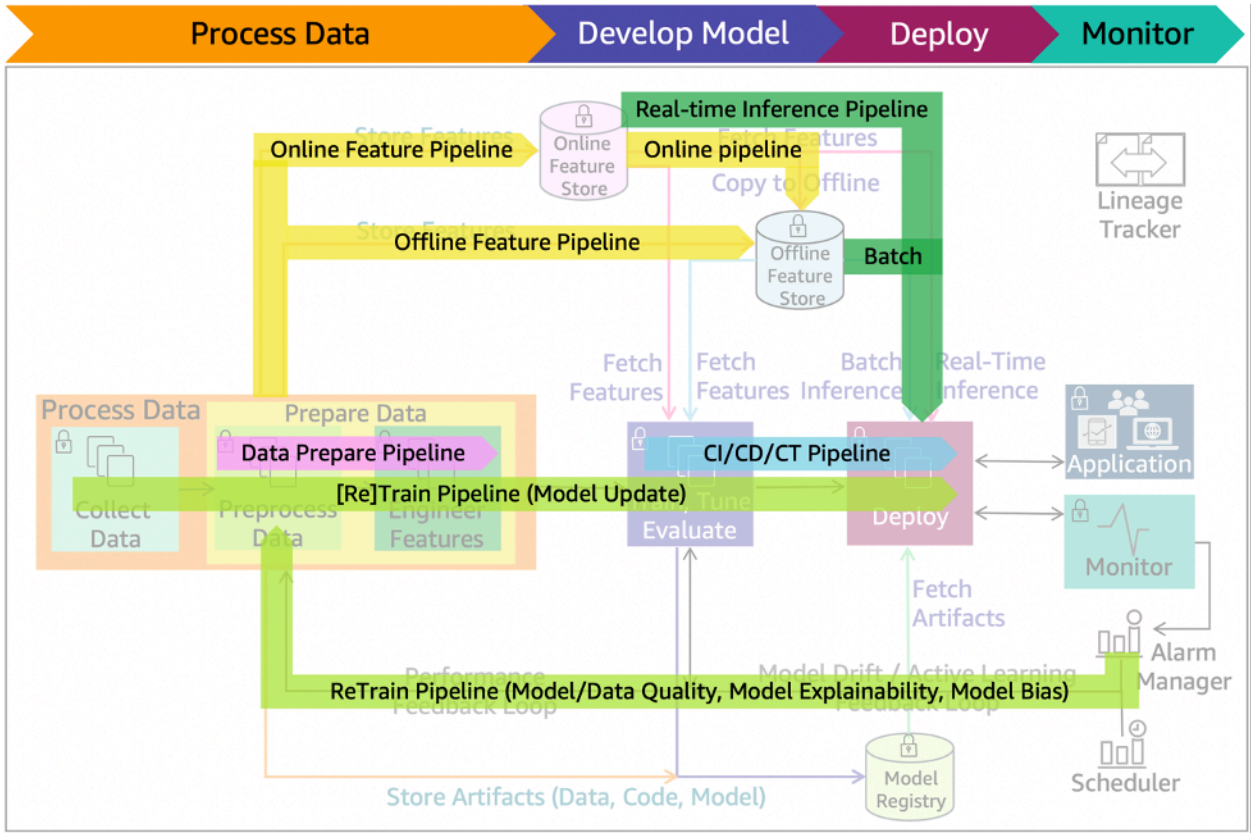

Figure 18: ML lifecycle with model update, retrain, and batch or real-time inference pipelines

Operational excellence

The operational excellence pillar includes the ability to run and monitor systems to deliver business value and to continually improve supporting processes and procedures. This section includes best practices to consider while identifying the business goal.

Each best practice in this section is presented based on its place in the ML lifecycle as detailed in *Machine learning lifecycle*.

Lifecycle phases

- Business goal identification
- ML problem framing
- Data processing
- Model development
- Deployment
- Monitoring

Business goal identification

Best practices

- MLOPS01-BP01 Develop the right skills with accountability and empowerment
- MLOPS01-BP02 Discuss and agree on the level of model explainability
- MLOPS01-BP03 Monitor model adherence to business requirements

MLOPS01-BP01 Develop the right skills with accountability and empowerment

Artificial intelligence (AI) has many different and growing branches, such as machine learning, deep learning, and computer vision. Given the complexity and fast-growing nature of ML technologies, plan to hire specialists with the understanding that additional training will be needed as ML evolves. Keep teams learning new skills, engaged, and motivated while encouraging accountability and empowerment. Building ML models is a complex and iterative process that can infuse bias or unfair predictions against a certain entity. It's important to promote and enforce the ethical use of AI across enterprises. AWS provides clear guidance to customers for responsible AI practices.

Desired outcome: You establish a skilled, ethically responsible ML workforce that continuously evolves with technology advancements. You create an environment where your teams are empowered to innovate with AI/ML while maintaining accountability for fair and unbiased AI solutions. Your organization develops a strong foundation in ML concepts, end-to-end lifecycle processes, and efficient use of AWS ML infrastructure and tools, enabling you to maximize business value through ethical and responsible AI implementation.

Common anti-patterns:

- Hiring ML specialists without a continuous learning plan.
- Focusing only on technical skills while ignoring ethical considerations.
- Creating siloed ML teams without cross-functional collaboration.
- Assuming ML models are inherently unbiased and fair.

Benefits of establishing this best practice:

- Increased innovation through skilled and empowered ML teams.
- Reduced risk of biased or unfair AI predictions.
- Improved ability to adapt to rapidly evolving ML technologies.
- Greater business value through responsible and ethical AI implementation.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Building a successful ML workforce requires a comprehensive approach that includes both technical and ethical skill development. Organizations must invest in continuous learning across various ML domains while also establishing clear accountability frameworks for responsible AI. By developing these capabilities together, you can maximize the benefits of ML while minimizing potential risks.

Creating an environment that fosters innovation while maintaining ethical standards is essential for sustainable ML success. This includes establishing clear guidelines for model development, testing for bias, and providing explainability of AI decisions. Cross-functional teams that combine technical expertise with domain knowledge and ethical considerations will deliver the most robust ML solutions.

As ML technologies evolve rapidly, your organization must remain adaptable and committed to ongoing skill development. This includes staying current with AWS's latest ML services, tools, and best practices while also keeping pace with advances in responsible AI methodologies.

Implementation steps

1. **Develop comprehensive ML skills training programs**. Create structured learning paths for different roles within your ML teams. Provide training on fundamental ML concepts and algorithms, end-to-end ML lifecycle processes, and efficient use of ML infrastructure with Amazon SageMaker AI. Include training on AI-assisted development tools like Amazon Q Developer for code generation and productivity enhancement. Incorporate specialized training in areas aligned with your business needs, such as computer vision, natural language processing (NLP), and reinforcement learning. Utilize resources like AWS Skill Builder, and Amazon Machine Learning University
2. **Establish cross-functional ML teams**. Form diverse teams with specialists from multiple disciplines including data science, engineering, ethics, legal, and domain experts. This multidisciplinary approach provides a holistic perspective on ML implementation and identifies potential issues early in the development process. Encourage collaboration across teams to share knowledge, best practices, and lessons learned from ML initiatives.
3. **Implement bias detection and fairness protocols**. Use Amazon SageMaker AI Clarify to detect and mitigate bias in your datasets and model predictions. Establish guidelines for evaluating models for fairness across different demographic groups and protected attributes. Create checkpoints throughout the ML lifecycle to assess and address potential bias before models are deployed into production environments.
4. **Create an ML ethics framework**. Develop clear guidelines and principles for ethical AI use within your organization. Establish an AI Ethics Board comprising representatives from legal, ethics, IT, data science, and key business units. Their responsibility should include creating policies for responsible AI implementation, providing guidance on complex ethical questions, and improving adherence to relevant regulations and industry standards.
5. **Foster accountability through documentation and governance**. Implement comprehensive documentation processes for each aspect of the ML lifecycle, including data sources, model development decisions, testing procedures, and deployment criteria. Create clear ownership and accountability structures for ML projects, with designated responsibilities for model performance, fairness, and explainability. Use Amazon SageMaker AI Model Cards to document model details, intended uses, limitations, and ethical considerations.
6. **Empower continuous learning and experimentation**. Create opportunities for teams to expand their ML knowledge through immersion days, hackathons, certification programs, and participation in the broader ML community. Establish sandboxed environments using Amazon SageMaker AI Unified Studio for integrated data and AI workflows where teams can experiment with new ML techniques and tools without risk to production systems.

7. **Measure and improve ML operations**. Implement metrics to evaluate the effectiveness of your ML teams and processes, including model performance, development efficiency, and ethics. Regularly review these metrics to identify areas for improvement and adjust your approach accordingly. Establish feedback loops that incorporate insights from model performance in production to continuously refine both technical implementations and team capabilities.
8. **Implement responsible generative AI practices**. As you explore generative AI capabilities, establish clear guardrails for using large language models and other generative technologies. Use Amazon Bedrock to access foundation models through a single API with enterprise-grade security and compliance features. Implement content filtering, safety measures, and human review processes for generative AI outputs to align with your organization's values and ethical standards.

Resources

Related documents:

- AWS ML Certification Paths
- AWS Ramp-Up Guides for Machine Learning
- Create a model card
- Configure a SageMaker AI Clarify Processing Job
- Responsible use of Artificial Intelligence and Machine Learning
- What is Amazon SageMaker AI?
- Building ML excellence: A practical training guide for Amazon SageMaker AI

MLOPS01-BP02 Discuss and agree on the level of model explainability

Establish clear expectations with business stakeholders about the level of model explainability needed for your machine learning use case. Explainability provides an understanding of how and why a model makes predictions, which builds trust, enables auditing, and supports adherence to regulatory requirements.

Desired outcome: You establish explainability requirements early in your machine learning project lifecycle. You implement appropriate methods to provide the agreed level of model explainability, building stakeholder trust and improving your adherence to regulations. You use explainability metrics in your evaluations and tradeoff analyses, verifying that the model meets business needs while remaining interpretable.

Common anti-patterns:

- Treating explainability as an afterthought rather than a core requirement.
- Choosing the most complex model without considering explainability requirements.
- Failing to establish explainability metrics before model development.
- Neglecting to communicate model decisions in terms understandable to business stakeholders.

Benefits of establishing this best practice:

- Increased stakeholder trust in model predictions.
- Better ability to troubleshoot and improve models.
- Enhanced ability to detect and address model biases.
- Improved model adoption by users who understand how decisions are made.

Level of risk exposed if this best practice is not established: High

Implementation guidance

When implementing machine learning models, you need to balance prediction accuracy with explainability. Highly complex models like deep neural networks might provide superior predictive performance but often function as an opaque system, making their decision-making processes difficult to interpret. In contrast, simpler models like decision trees offer greater transparency but may sacrifice some accuracy.

The appropriate level of explainability depends on your specific use case. In regulated industries like healthcare, finance, or insurance, high explainability might be mandatory for compliance-aligned reasons. For applications where human safety or significant financial decisions are involved, understanding why a model made a specific prediction becomes critical.

Work with your business stakeholders to understand their explainability requirements before selecting modeling approaches. Consider both technical and non-technical aspects of explainability - technical stakeholders may need detailed feature importance measures, while business users might need simple, intuitive explanations of model decisions.

Implementation steps

1. **Understand business requirements for explainability**. Meet with stakeholders to determine how much transparency is needed based on use case, industry regulations, and business

objectives. In regulated industries like healthcare and finance, regulations often mandate that automated decisions be explainable to affected individuals.

2. **Evaluate model types based on explainability needs**. Consider inherently interpretable models (linear regression, decision trees) for high explainability requirements, or more complex models with following explanation techniques when higher accuracy is the priority.
3. **Set up SageMaker AI Clarify**. Implement Amazon SageMaker AI Clarify to create explainability reports and detect potential biases in your datasets or models. Clarify includes enhanced bias detection and new fairness metrics for more comprehensive explainability analysis. SageMaker AI Clarify integrates with SageMaker AI's model building, training, and deployment capabilities.
4. **Choose appropriate SHAP baselines**. Shapley Additive exPlanations (SHAP) values determine how each feature contributes to predictions. Configure appropriate baselines in SageMaker AI Clarify based on your data characteristics. You can choose baselines with "low information content" (for example, average values from the training dataset) or high information content (representing a specific class of interest).
5. **Generate and interpret feature attribution reports**. Use SageMaker AI Clarify to generate feature attribution reports showing which features most influenced model predictions and how they did so. Review these reports with stakeholders to verify that they provide the required level of understanding.
6. **Create user-friendly explanation interfaces**. Develop appropriate visualization tools or explanation interfaces that present model insights in ways that are meaningful to various stakeholders, from data scientists to business users.
7. **Implement continuous explainability monitoring**. Set up ongoing monitoring of model explanations to detect drift in feature importance or unexpected behavior patterns over time.
8. **Apply responsible AI principles to generative models**. For generative AI applications, implement additional explainability measures such as prompt transparency, citation of sources, and confidence scores to assist users in understanding how outputs were generated.

Resources

Related documents:

- Model Explainability
- Configure a SageMaker AI Clarify Processing Job
- SHAP Baselines for Explainability
- Explain text classification model predictions using Amazon SageMaker AI Clarify

MLOPS01-BP03 Monitor model adherence to business requirements

Machine learning models degrade over time due to changes in the real world, such as *data drift* and *concept drift*. If not monitored, these changes could lead to models becoming inaccurate or even obsolete over time. It's important to have a periodic monitoring process in place to make sure that your ML models continue to comply to your business requirements and that deviations are captured and acted upon promptly.

Desired outcome: You implement a robust model monitoring framework that continuously evaluates model performance against your business requirements. This enables early detection of model drift, keeping your ML models accurate and effective over time. You establish clear metrics tied to business outcomes and have automated processes to respond to detected drifts, minimizing potential negative impacts.

Common anti-patterns:

- Implementing monitoring without clear metrics tied to business requirements.
- Focusing only on technical metrics while ignoring business impact metrics.
- Lacking a clear action plan for when drift is detected.
- Monitoring models infrequently or irregularly.
- Not establishing thresholds for acceptable levels of drift.

Benefits of establishing this best practice:

- Early detection of model performance degradation.
- Maintained alignment between model outputs and business goals.
- Reduced risk of financial or operational impact from degraded models.
- Increased stakeholder confidence in deployed ML systems.
- Improved model lifecycle management.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Model monitoring is a critical aspect of maintaining ML systems that continue to deliver business value over time. As real-world conditions evolve, your models can experience both data drift (changes in the distribution of input data) and concept drift (changes in the relationship between inputs and outputs). These drifts can significantly impact model accuracy and reliability.

To effectively monitor model adherence to business requirements, establish a comprehensive monitoring strategy that bridges technical metrics with business outcomes. This involves defining clear thresholds for acceptable performance, implementing automated monitoring solutions, and creating response plans for different drift scenarios.

Amazon SageMaker AI Model Monitor provides capabilities to automatically monitor models in production and detect deviations from the baseline. By using these capabilities, you can proactively address potential issues before they impact your business operations.

Implementation steps

1. **Define relevant metrics aligned with business objectives.** Begin by clearly establishing the metrics that are most relevant to your business outcomes. Include both technical metrics (like accuracy, precision, and recall) and business metrics (like revenue impact and customer satisfaction). Make sure these metrics directly relate to the business requirements the model is expected to fulfill.
2. **Establish baseline performance and thresholds.** Create a performance baseline using your validation dataset. Using Amazon SageMaker AI Model Monitor, you can automatically generate statistics and constraints from your baseline data that define normal behavior. Set appropriate thresholds that will alert when model performance deviates beyond acceptable limits.
3. **Implement automated data quality monitoring.** Configure SageMaker AI Model Monitor to regularly check the quality of input data against the baseline. This can detect data drift that could affect model performance. Monitor features for statistical changes in distributions, missing values, or other quality issues.
4. **Configure model quality monitoring.** Set up monitoring for model outputs and quality metrics. SageMaker AI Model Monitor can track prediction distributions and performance metrics over time, alerting you when they deviate from expected patterns.
5. **Set up bias drift detection.** Use SageMaker AI Clarify integration with Model Monitor to detect changes in bias metrics over time. This keeps your model fair and unbiased as production data evolves.
6. **Create visualization dashboards.** Implement dashboards using Amazon CloudWatch, Amazon Managed Grafana, or use Quick with GenBI capabilities to automatically generate monitoring dashboards. These dashboards should present both technical and business metrics in an understandable format for stakeholders.
7. **Develop response protocols for drift detection.** Create clear action plans for different types and severities of detected drift. These might include automated retraining pipelines, manual reviews, or temporary fallback strategies depending on the scenario.

8. **Implement alert mechanisms.** Configure alerts using Amazon CloudWatch to notify appropriate team members when metrics exceed thresholds. Check that your alerts provide actionable information about the nature and potential impact of the drift.
9. **Establish regular review processes.** Schedule periodic reviews of monitoring results even when no alerts go off. This can identify gradual drifts that might not immediately alert but could impact performance over time.
10. **Document monitoring systems and processes.** Maintain comprehensive documentation of your monitoring setup, including metrics definitions, thresholds, alert configurations, and response protocols to preserve organizational knowledge.
11. **Leverage generative AI for root cause analysis.** Use generative AI capabilities to analyze complex patterns in your monitoring data and provide human-readable explanations of potential drift causes. Tools like Amazon Bedrock Knowledge Bases can interpret changes in model behavior and suggest remediation approaches.

Resources

Related documents:

- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- Schema for Violations (constraint_violations.json file)
- Amazon SageMaker AI metrics in Amazon CloudWatch
- What is Amazon CloudWatch?
- Amazon SageMaker AI Clarify
- Amazon SageMaker AI ML Governance

ML problem framing

Best practices

- MLOPS02-BP01 Establish ML roles and responsibilities
- MLOPS02-BP02 Prepare an ML profile template
- MLOPS02-BP03 Establish model improvement strategies
- MLOPS02-BP04 Establish a lineage tracker system
- MLOPS02-BP05 Establish feedback loops across ML lifecycle phases
- MLOPS02-BP06 Review fairness and explainability

MLOPS02-BP01 Establish ML roles and responsibilities

Clearly defining roles, responsibilities, and team interactions in machine learning projects creates an efficient operational framework that maximizes overall effectiveness. By understanding who does what and how teams collaborate, organizations can streamline their ML initiatives and deliver better business outcomes.

Desired outcome: You establish well-defined roles and responsibilities across your ML teams, enabling proper collaboration and accountability. You have mechanisms to efficiently manage access controls for various ML functions, providing your team members access to the tools and resources they need while maintaining appropriate security boundaries. This creates a foundation for successful ML operations that supports both innovation and governance.

Common anti-patterns:

- Undefined or overlapping responsibilities causing confusion among team members.
- Relying on a single person to perform ML-related tasks rather than building specialized expertise.
- Over-privileged access controls that compromise security.
- Manual, one-time processes for managing user permissions that don't scale.
- Siloed teams with poor communication channels between technical and business stakeholders.

Benefits of establishing this best practice:

- Clear accountability and ownership throughout the ML lifecycle.
- Improved collaboration between technical and business teams.
- Streamlined decision-making processes and faster project initiation.
- Better governance and risk management through proper access controls.
- Enhanced ability to scale ML operations across the organization.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Establishing clear ML roles and responsibilities requires thoughtful planning about how teams will work together throughout the entire ML lifecycle. Machine learning projects span multiple domains including business expertise, data engineering, model development, and operations, each

requiring specialized skills. Without clearly defined roles, projects often face delays, quality issues, or governance challenges.

Begin by identifying which functions are critical for your organization's ML initiatives. Consider your business objectives, technical requirements, and regulatory constraints. Develop a team structure that balances specialization with collaboration, allowing for efficient workflows while maintaining appropriate separation of duties for governance purposes.

For enterprise-grade ML solutions, establish cross-functional teams with clear responsibilities for each role. Consider how these roles interact throughout the ML lifecycle and create communication channels to facilitate collaboration. Pay special attention to governance responsibilities, as these are often overlooked in early ML initiatives but become critical as projects move into production.

Implementation steps

1. **Map ML functions to organizational needs**. Begin by identifying the ML capabilities required to support your business objectives. Consider the entire ML lifecycle from problem definition through to production monitoring. Review your current organizational structure and identify gaps in skills or functions that need to be addressed. Create a matrix showing the relationship between ML functions and existing teams or roles.
2. **Establish cross-functional teams with defined roles**. Create a formal structure for your ML organization with clear roles and responsibilities for each team member. Gather representation from both technical and business domains to maintain alignment with business outcomes. The following roles should be considered:
   - **Domain expert:** Provides functional knowledge about the business problem and validates ML approaches against real-world requirements.
   - **Data engineer:** Transforms raw data into formats suitable for analysis and model training.
   - **Data scientist:** Applies statistical modeling and machine learning techniques to derive insights from data.
   - **ML engineer:** Converts data science prototypes into production-ready software systems.
   - **MLOps engineer:** Builds automation pipelines for model training, testing, and deployment.
   - **IT auditor:** Analyzes system access, identifies anomalies, and recommends remediations.
   - **Model risk manager:** Checks that models meet internal and external control requirements.
   - **Cloud security engineer:** Configures and manages cloud resources with appropriate security controls.
   - **Prompt engineer:** Designs effective interactions with foundation models for generative AI applications.

3. **Implement role-based access control**. Design a permissions framework that follows the principle of least privilege while enabling teams to be productive. Avoid one-time methods for managing access policies that don't scale. Instead, use Amazon SageMaker AI Role Manager to efficiently control access based on pre-defined templates aligned with your organizational roles. This allows administrators to quickly create appropriate access policies within minutes, reducing the time and effort required to onboard users and manage permissions over time.
4. **Establish governance processes**. Create clear processes for model lifecycle management, including approval workflows, validation requirements, and regulatory checks. Document who is responsible for key decisions at each stage of development. Implement model monitoring mechanisms to track performance and alert when intervention is needed. Use Amazon SageMaker AI Model Dashboard to maintain visibility across your model inventory and track performance metrics.
5. **Develop collaboration frameworks**. Establish standard communication channels and collaboration tools to facilitate interaction between different roles. Create documentation templates that promote knowledge sharing and make handoffs between teams more efficient. Schedule regular cross-functional reviews to gain alignment throughout the ML lifecycle. Consider using Amazon SageMaker AI Unified Studio as a collaborative environment that unifies data and AI workflows where data scientists and engineers can work together.
6. **Train teams on responsibilities and interfaces**. Provide training to verify that your team members understand not only their own responsibilities but also how their work impacts others in the ML lifecycle. Create reference materials that clarify handoff points and dependencies between different roles. Consider establishing a center of excellence or community of practice to share knowledge and best practices across teams.
7. **Adapt roles for generative AI initiatives**. When implementing generative AI projects, consider how traditional ML roles need to adapt. Prompt engineers may be needed to design effective interactions with foundation models. Ethical AI specialists can address concerns around bias, transparency, and responsible use. Integration engineers may be required to connect foundation models from services like Amazon Bedrock with enterprise applications and data sources.

Resources

Related documents:

- Amazon SageMaker AI Role Manager
- Personas for an ML platform
- ML Learning Paths

- Amazon SageMaker AI Model Dashboard
- Why should you use MLOps?
- Amazon SageMaker AI ML Governance
- Define customized permissions in minutes with Amazon SageMaker AI Role Manager
- New ML Governance Tools for Amazon SageMaker AI – Simplify Access Control and Enhance Transparency Over Your ML Projects

MLOPS02-BP02 Prepare an ML profile template

Creating an ML profile template allows you to systematically capture machine learning workload characteristics across different lifecycle phases. Use this template to evaluate your ML workload's current maturity status and strategically plan for improvements that align with your business requirements.

Desired outcome: You gain a comprehensive understanding of your ML workload's deployment characteristics by creating templated profiles that capture critical metrics and thresholds. By maintaining current and target profiles, you can effectively track your ML workload maturity journey and make data-driven decisions about architecture, resources, and deployment options that best align with your business needs.

Common anti-patterns:

- Creating ML profiles without clear thresholds or maturity rankings.
- Failing to document rationale for architectural and deployment choices.
- Focusing on technical metrics without connecting to business requirements.
- Not considering future state or alternative deployment options.
- Ignoring cost implications of different deployment scenarios.

Benefits of establishing this best practice:

- Enables objective assessment of ML workload maturity status.
- Provides a structured approach to planning ML workload improvements.
- Creates alignment between technical implementation and business requirements.
- Facilitates better resource planning and cost optimization.
- Supports strategic decision-making with documented rationale.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Machine learning workloads have unique characteristics that impact their deployment architecture, infrastructure requirements, and operational management. Create a standardized ML profile template to document these characteristics systematically across the stages of the ML lifecycle. By capturing key metrics and establishing thresholds, you can objectively assess the maturity level of your ML workloads and identify areas for improvement.

For each ML workload, you should maintain at least two profiles: one representing the current state and another representing the target or future state. This approach creates a clear path for improvement while documenting the rationale behind architectural and deployment decisions.

When developing your ML profile template, focus on the most impactful characteristics that influence infrastructure choices, deployment options, and operational considerations. Include metrics that span model characteristics, architectural decisions, and operational requirements. For each characteristic, establish a spectrum from lower to higher ranges to position your workload on a maturity scale.

Implementation steps

1. **Capture ML workload deployment characteristics**. Identify and document the most impactful deployment characteristics of your ML workload. These characteristics can determine optimal deployment architecture, computing requirements, and instance sizing. Use Amazon SageMaker AI Inference Recommender, which includes more sophisticated instance selection algorithms and support for multi-model endpoints, to optimize instance selection based on your model's performance and cost requirements.
2. **Document model deployment characteristics**. Record key metrics about your ML models, including:
   - Model size (model.tar.gz) in bytes
   - Number of models deployed per endpoint
   - Instance size (for example, r5dn.4x.large) as suggested by the inference recommender
   - Retraining and model endpoint update frequency (hourly, daily, weekly, monthly, or per-event)
   - Model deployment location (on premises, Amazon EC2, container, serverless, or edge)
3. **Map architectural deployment characteristics**. Capture information about the internal architecture of your ML solution:
   - Inference pipeline architecture (single endpoint or chained endpoints)
   - Neural architecture (single framework like Scikit-learn or multi-framework like PyTorch, Scikit-learn, TensorFlow)

- Containers (SageMaker AI prebuilt container, bring your own container)
- Location of containers and models (on premises, cloud, or hybrid)
- Serverless inferencing (pay as you go) options like Amazon SageMaker AI Serverless Inference
- Inference Components for modular inference pipeline architecture (mix and match pre-processing, model serving, and post-processing components)

4. **Define traffic pattern characteristics**. Document how your ML model will be used:
   - Traffic pattern (steady or spiky)
   - Input size (number of bytes)
   - Latency requirements (low, medium, high, or batch)
   - Concurrency needs (single thread or multi-thread)
5. **Determine cold start tolerance**. Document the acceptable latency for cold starts in milliseconds, as this impacts the choice between always-on and serverless deployment options.
6. **Evaluate network deployment characteristics**. Assess and document network-related requirements, including AWS KMS encryption needs, multi-variant endpoints, network isolation requirements, and use of third-party Docker repositories.
7. **Analyze cost considerations**. Document cost considerations for different deployment options, including the potential use of Amazon EC2 Spot Instances for non-critical workloads, Amazon SageMaker AI Serverless Inference for pay-per-use models, or multi-model endpoints for cost sharing.
8. **Create a provisioning matrix**. Develop a matrix of expected capacity requirements across different environments (development, staging, production) and regions. This should include the number and types of instances needed for training, batch inference, real-time inference, and development notebooks.
9. **Map workload characteristics across maturity spectrum**. For each characteristic in your profile, establish a spectrum from lower to higher maturity. This spectrum positions your current implementation and defines targets for improvement.
10. **Document rationale for target profile**. Provide clear justification for the values selected in your target profile, linking them to specific business requirements and expected outcomes.
11. **Evaluate and update profiles regularly**. Revisit your ML profiles periodically to check that they remain aligned with evolving business requirements and to incorporate learnings from production experience.
12. **Foundation model deployment considerations**. For generative AI workloads, include additional profile characteristics such as model quantization level, context window requirements, token processing speed, and memory footprint using optimized foundation model containers to properly size infrastructure for these resource-intensive models.

Resources

Related documents:

- SageMaker AI Inference Recommender
- AWS Pricing Calculator - SageMaker AI
- Deploy models with Amazon SageMaker AI Serverless Inference
- Multi-model endpoints
- Amazon EC2 Instance Types
- What is Amazon SageMaker AI?
- Improved ML model deployment using Amazon SageMaker AI Inference Recommender
- Unlock cost savings with the new scale down to zero feature in SageMaker AI Inference
- Beyond the basics: A comprehensive foundation model selection framework for generative AI

MLOPS02-BP03 Establish model improvement strategies

Planning improvement drivers for optimizing machine learning model performance is essential before development begins. By establishing a clear strategy for model enhancement, you can systematically improve your ML models through techniques like data collection, cross-validation, feature engineering, hyperparameter tuning, and ensemble methods.

Desired outcome: By implementing this best practice, you establish a systematic approach for improving your machine learning models. You create a structured methodology for experimentation that allows you to progressively enhance model performance by testing different improvement strategies. You gain visibility into which approaches yield the best results for your specific use case, enabling you to make data-driven decisions about model development and optimization.

Common anti-patterns:

- Starting with overly complex models without establishing a baseline performance.
- Ignoring data quality issues and focusing solely on model complexity.
- Making arbitrary hyperparameter changes without systematic experimentation.
- Neglecting to document experimental results and configurations.
- Implementing advanced techniques without understanding fundamental model performance issues.

Benefits of establishing this best practice:

- Enables structured experimentation and measurable model improvements.
- Provides clarity on which improvement strategies deliver the best results.
- Reduces time spent on ineffective optimization approaches.
- Creates a systematic pathway from simple to more advanced modeling techniques.
- Identifies the most important features and data for your specific business problem.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Effective model improvement requires a structured approach that follows a progression from simple to complex. Begin by establishing clear performance metrics tied to your business objectives, as these will guide your improvement efforts. Develop a baseline model using straightforward algorithms and minimal feature engineering to set initial performance benchmarks.

Organizing your experiments systematically is crucial. Document your approach, model configurations, and results to track improvements and understand the impact of different strategies. This documentation serves as institutional knowledge that can guide future model development and improvement efforts.

Work closely with domain experts who understand the business context. Their insights can identify key features, validate model outputs, and align your improvements with business objectives. Remember that model improvement is an iterative process requiring patience and methodical experimentation.

Implementation steps

1. **Create a baseline model with minimal complexity**. Start with minimal data cleaning and the most obvious data features. Train simple classical models using algorithms like linear regression or logistic regression for classification tasks. This establishes a performance benchmark against which you can measure improvements. Use Amazon SageMaker AI to quickly develop these baseline models.
2. **Organize and track experiments using MLFlow on Amazon SageMaker AI**. Use Amazon SageMaker AI MLFlow to organize and track multiple tests comparing different configurations and algorithms. This allows you to maintain a structured approach to experimentation and

compare results across different model versions so you can identify which changes lead to meaningful improvements.

3. **Implement effective feature selection**. Collaborate with subject matter experts to identify the most significant features related to target values. Iteratively add more complex features and remove less important ones to improve model accuracy and robustness. Test different feature engineering techniques such as one-hot encoding, normalization, and dimensionality reduction to understand their impact on model performance.
4. **Consider deep learning for complex patterns**. When you have large volumes of training data, consider deep learning models to discover previously unknown features and improve model accuracy. Amazon SageMaker AI provides built-in algorithms for deep learning and supports popular frameworks like TensorFlow and PyTorch, making it simpler to experiment with neural network architectures.
5. **Explore ensemble methods**. Ensemble methods can provide further accuracy improvements by combining the best characteristics of various algorithms. Consider techniques like random forests, gradient boosting, or stacking different models. Be aware of the tradeoffs with computational performance and maintenance complexity, evaluating whether these approaches align with your specific business use case.
6. **Apply automatic machine learning (AutoML)**. Use Amazon SageMaker AI Canvas for no-code/low-code ML model development with natural language support for data exploration and preparation. Canvas includes Amazon Q integration for conversational data analysis and can directly deploy models to production.
7. **Optimize hyperparameters systematically**. Fine-tune hyperparameters for each algorithm to obtain optimal performance. Use Amazon SageMaker AI Hyperparameter Optimization to automate this process through techniques like Bayesian optimization, grid search, or random search to find the most effective parameter combinations.
8. **Integrate experiments into automated workflows**. Automate your experimental trials using SageMaker AI Pipelines by using the integration with experiments. This fosters reproducibility and creates a systematic approach to model improvement that can be incorporated into your ML operations workflow.
9. **Use generative AI for synthetic data creation**. For scenarios with limited training data, use generative AI techniques like generative adversarial networks (GANs) to create synthetic data that can expand your training dataset. Amazon SageMaker AI JumpStart provides an expanded library of pre-built generative AI models and more industry-specific solutions, while Amazon Bedrock offers foundation models that can augment your training data, especially for scenarios with limited or imbalanced datasets.

10**For generative AI workloads, apply foundation models with RAG for knowledge-intensive tasks**. For tasks requiring domain knowledge, implement retrieval-augmented generation (RAG) using Amazon Bedrock to enhance foundation models with your organization's specific information, combining the power of large language models with your proprietary data.

Resources

Related documents:

- Amazon SageMaker AI Experiments in Studio Classic
- Accelerate generative AI development using managed MLFlow on SageMaker AI
- Amazon SageMaker AI Canvas
- Automatic model tuning with SageMaker AI
- SageMaker AI JumpStart pretrained models
- Amazon SageMaker AI Experiments Integration
- Create, store, and share features with Feature Store
- Model Registration Deployment with Model Registry
- LLM experimentation at scale using Amazon SageMaker AI Pipelines and MLFlow

Related examples:

- Amazon SageMaker AI Experiments Examples
- Hyperparameter Tuning Examples
- Ensemble Predictions from Multiple Models
- Amazon SageMaker AI Autopilot Examples

MLOPS02-BP04 Establish a lineage tracker system

Maintain a system that tracks changes for each release to enable reproducibility, speed up problem diagnosis, and improve regulatory adherence. Tracking lineage for model development includes changes in documentation, environment, model, data, code, and infrastructure.

Desired outcome: You have a comprehensive lineage tracking system that records the history of artifacts involved in your ML model development and deployment lifecycle. This system enables you to reproduce previous model versions, diagnose issues quickly, roll back to stable versions when needed, and improve your regulatory adherence. Your organization can trace model results back to their originating data sources, code versions, and infrastructure configurations.

Common anti-patterns:

- Manually tracking changes in spreadsheets or documents.
- Not capturing all dependent artifacts necessary for model reproduction.
- Inconsistent tracking practices across teams.
- Lacking integration between different components of the ML pipeline.
- Failing to track infrastructure and environment configurations.

Benefits of establishing this best practice:

- Enables reproducibility of model versions for debugging.
- Accelerates problem diagnosis and resolution when issues arise.
- Supports regulatory adherence and audit requirements.
- Facilitates rollback to previous stable versions when needed.
- Improves collaboration among team members with transparent tracking.
- Enhances model governance and responsible AI practices.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Implementing a comprehensive lineage tracker system is essential for maintaining the traceability and reproducibility of your machine learning models. Without proper lineage tracking, your organization may struggle to debug issues, comply with regulations, or reproduce previous model versions when needed. The lineage tracker should capture information about components that influence your model's behavior, including the data used for training, preprocessing steps, hyperparameters, model architecture, evaluation metrics, and deployment environment.

A robust lineage tracking system starts with identifying the key artifacts that need to be tracked throughout the ML lifecycle. Once identified, you can use AWS services like SageMaker AI ML Lineage Tracking to automatically record and store the relationships between these artifacts. This information becomes invaluable when you need to audit your models, reproduce results, or diagnose issues in production.

For example, if a model suddenly begins producing unexpected results in production, your lineage tracking system should allow you to trace back to the exact version of the model, the data it was trained on, the code used to train it, and the infrastructure configuration at the time of

deployment. This comprehensive tracking enables faster problem resolution and maintains trust in your ML systems.

Implementation steps

1. **Identify artifacts needed for tracking**. Begin by identifying artifacts that contribute to your model's development and deployment. This includes raw data, processed data, feature sets, model parameters, code versions, training environments, and deployment configurations. Understanding what needs to be tracked is essential for meeting regulatory requirements and enabling reproducibility. Refer to Data and artifacts lineage tracking for guidance on the artifacts to include.
2. **Implement SageMaker AI ML Lineage Tracking**. Use Amazon SageMaker AI ML Lineage Tracking to automatically record information about the steps in your ML workflow. SageMaker AI tracks relationships between datasets, algorithms, training jobs, and model artifacts, enabling you to reproduce workflows, track model and dataset lineage, and establish governance standards. The service creates entities for your ML workflow components and stores their relationships, making it more straightforward to audit and verify model provenance.
3. **Set up SageMaker AI Unified Studio**. Use Amazon SageMaker AI Unified Studio as your integrated development environment that unifies data and AI workflows. SageMaker AI Unified Studio provides enhanced collaborative features, team sharing capabilities, and visual tools to track the lineage of your ML pipelines, making it more straightforward to understand the relationships between different components across data and AI personas.
4. **Configure SageMaker AI Feature Store**. Implement Amazon SageMaker AI Feature Store to create a centralized repository for storing, sharing, and managing features with enhanced feature management capabilities. This purpose-built repository enables you to organize features in a consistent way, making them easily accessible across teams. SageMaker AI Feature Store fosters feature consistency between training and inference phases without requiring additional code, and maintains a record of feature versions over time.
5. **Use SageMaker AI Model Registry**. Implement Amazon SageMaker AI Model Registry to catalog models for production, manage model versions, and associate metadata with models. The Model Registry enables lineage tracking by maintaining a history of model versions, approval status, and deployment details. This creates a centralized repository for managing model lifecycles, which facilitates governance and improves adherence to regulations.
6. **Build ML pipelines with SageMaker AI Pipelines**. Create reproducible ML workflows using Amazon SageMaker AI Pipelines to automate and standardize the steps in your ML process. SageMaker AI Pipelines allows you to track data history within the pipeline and integrates with

SageMaker AI ML Lineage Tracking to analyze input data, its sources, and generated outputs. This integration creates comprehensive lineage tracking across your entire ML workflow.

7. **Implement version control practices**. Use version control systems like Git for code, model configurations, and pipeline definitions. Integrate these systems with your lineage tracking to properly link code changes to model versions and training runs. This practice maintains a complete history of how your models have evolved over time.
8. **Establish model attributes for training runs**. Use model attributes in SageMaker AI to track specific details about your training runs. This allows you to compare different experiments, understand which parameters led to better model performance, and maintain records of training decisions. For more detail, see Using model attributes to track your training runs on Amazon SageMaker AI.
9. **Implement access controls and auditing**. Set up appropriate access controls for your lineage data and implement auditing capabilities to track who accesses or modifies lineage information. Use AWS Lake Formation with SageMaker AI Studio to control and audit data exploration activities, as demonstrated in this example.
10. **Develop regular verification processes**. Establish procedures to regularly verify that your lineage tracking system is capturing necessary information for compliance-aligned purposes. Create automated reports that demonstrate the completeness of your lineage tracking to adhere to regulatory requirements.
11. **Foundation model lineage tracking**. Consider implementing Amazon Bedrock for tracking lineage in generative AI applications. With foundation models becoming increasingly important, tracking prompt engineering changes, model parameters, and fine-tuning datasets is critical for reproducibility and governance of generative AI systems. Use Amazon Bedrock's governance features to maintain comprehensive lineage tracking when working with foundation models.

Resources

Related documents:

- Lineage Tracking Entities
- Track the lineage of a pipeline
- MLOps Automation With SageMaker AI Projects
- Data and artifacts lineage tracking
- Pipelines
- Model Registration Deployment with Model Registry
- Create, store, and share features with Feature Store

Related examples:

- End-to-End MLOps with Amazon SageMaker AI
- Controlling and auditing data exploration activities with SageMaker AI Studio and AWS Lake Formation
- SageMaker AI Feature Store Examples

MLOPS02-BP05 Establish feedback loops across ML lifecycle phases

Establishing feedback loops across machine learning (ML) lifecycle phases is essential for continuous improvement of ML workloads. By implementing robust mechanisms to share successful experiments, analyze failures, and document operational activities, you can enhance model performance and adapt to changing data patterns over time. Feedback loops can identify model drifts and enable practitioners to refine monitoring and retraining strategies, allowing for experimentation with data augmentation and different algorithms until optimal outcomes are achieved.

Desired outcome: You have established comprehensive feedback mechanisms across your ML lifecycle that facilitate continuous learning and improvement. Your organization can detect model drift early, automatically initiate retraining when needed, and incorporate human review when appropriate. This creates a culture of experimentation where successes and failures contribute equally to knowledge advancement, improving both model quality and operational efficiency over time.

Common anti-patterns:

- Treating model deployment as the final step without ongoing evaluation.
- Failing to document experiments and operational learnings.
- Ignoring model drift until it significantly impacts performance.
- Working in silos without sharing insights across ML teams.
- Lacking automated processes to respond to detected model issues.

Benefits of establishing this best practice:

- Early detection of model performance degradation.
- Reduced manual effort through automated monitoring and retraining.
- Improved model quality through systematic experimentation.
- Knowledge retention and sharing across teams.

- Enhanced ability to adapt to changing data patterns.
- Accelerated innovation through documented learnings.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Feedback loops are crucial to maintaining the effectiveness of ML models over time. As real-world data evolves, the performance of deployed models can deteriorate due to concept drift or model drift. By establishing systematic feedback mechanisms, you can monitor these changes, learn from them, and adapt your models accordingly.

A comprehensive feedback loop strategy begins with monitoring model performance metrics and comparing them against baseline expectations. When deviations occur, automated alerts can notify teams or run retraining pipelines. The results of these actions should be documented to inform future development decisions. This creates a continuous cycle of learning where each iteration builds upon previous insights.

For example, a financial services company deployed a fraud detection model that began showing decreased accuracy after six months. Their established feedback system detected this drift, automatically started a retraining pipeline using recent data, and documented the patterns that caused the drift. Data scientists can use this information to improve feature engineering in subsequent model versions, resulting in more resilient performance.

Human review is also an essential component of feedback loops, especially for sensitive applications. Including human validation through tools like Amazon A2I provides ground truth data that can be used to further refine models and build trust in automated decisions.

Implementation steps

1. **Establish comprehensive model monitoring with SageMaker AI Model Monitor**. Amazon SageMaker AI Model Monitor provides capabilities to continuously monitor the quality of machine learning models in production. Configure it to detect data and concept drift by comparing production data statistics against the baseline. SageMaker AI Model Monitor supports monitoring data quality, model quality, bias drift, and feature attribution drift, providing a complete view of your model's performance over time.
2. **Configure CloudWatch alerts and notifications**. Set up Amazon CloudWatch to monitor metrics generated by SageMaker AI Model Monitor and create custom dashboards to visualize model performance. Configure CloudWatch Alarms to send notifications when thresholds are exceeded,

indicating potential model drift. These notifications can be delivered using SNS topics to email, SMS, or other channels for prompt attention.

3. **Implement the SageMaker AI Model Dashboard**. Use the SageMaker AI Model Dashboard as the central interface for tracking models and their performance. The dashboard provides a comprehensive view of deployed models, their endpoints, monitoring schedules, and historical behavior. This allows teams to quickly identify issues and understand performance trends across multiple models.
4. **Automate retraining pipelines**. Create automated retraining workflows using AWS Step Functions and SageMaker AI Pipelines. Configure EventBridge Events rules to run these pipelines when SageMaker AI Model Monitor detects drift or anomalies. This verifies that models are retrained with fresh data when performance begins to degrade, maintaining high accuracy with minimal manual intervention.
5. **Incorporate human review with Amazon A2I**. Implement Amazon Augmented AI (A2I) to route predictions with low confidence scores to human reviewers. This creates a human-in-the-loop feedback mechanism where reviewers can validate model outputs and provide corrections. The reviewed data becomes valuable ground truth that can be used to improve model performance in future iterations.
6. **Document and share learnings**. Create a knowledge repository using services like Quick with GenBI capabilities to automatically generate visualizations and dashboards, and Amazon S3 for storing experiment results and operational reports. This documentation should include successful approaches, failed experiments, and operational insights to facilitate knowledge sharing across teams.
7. **Establish regular feedback review sessions**. Schedule recurring meetings with stakeholders to review monitoring results, discuss model performance, and prioritize improvements. These sessions should include data scientists, ML engineers, and business stakeholders to align between technical improvements and business outcomes.

Resources

Related documents:

- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- Amazon SageMaker AI Model Dashboard
- Amazon Augmented AI (A2I)
- Train a machine learning model using Amazon SageMaker AI
- Pipelines

- Creating rules that react to events in Amazon EventBridge
- Monitoring Amazon ML with Amazon CloudWatch Metrics

MLOPS02-BP06 Review fairness and explainability

Evaluating model fairness and explainability verifies that your machine learning solutions are ethical, transparent, and equitable across user groups. By systematically addressing these concerns throughout the ML lifecycle, you build trust with stakeholders and reduce the risk of harmful bias in your models.

Desired outcome: You implement a comprehensive approach to fairness and explainability across your machine learning lifecycle. You can identify potential biases in data and models, explain model predictions to stakeholders, and know that your AI systems make equitable decisions across user segments. Your ML systems are continuously monitored for fairness, comply with relevant regulations, and maintain transparency that builds trust with users and stakeholders.

Common anti-patterns:

- Treating fairness as an afterthought rather than integrating it throughout the ML lifecycle.
- Focusing solely on model accuracy without considering ethical implications.
- Using non-representative training data that leads to biased outcomes.
- Deploying complex, opaque models when explainability is required.
- Failing to monitor models in production for drift in fairness metrics.

Benefits of establishing this best practice:

- Builds trust with customers and stakeholders through transparent AI systems.
- Reduces the risk of regulatory issues related to algorithmic fairness.
- Identifies and mitigates harmful biases before models enter production.
- Enables better understanding of model decisions through explainability techniques.
- Creates more inclusive AI systems that work equitably for user groups.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Fairness and explainability should be considered fundamental components of your machine learning development process rather than optional add-ons. Approaching these concerns

proactively can verify that your models work equitably for your users while providing transparency into how decisions are made.

Start by assessing the ethical implications of your ML solution during problem framing. Consider whether an algorithm is the appropriate approach and what impacts it might have on different user groups. For data management, analyze whether your training data adequately represents the diversity of your user population, and check for existing biases in labels or features that could be perpetuated by your model.

During training and evaluation, consider incorporating fairness constraints directly into your optimization functions. Evaluate models using appropriate fairness metrics alongside traditional performance measures. When deploying models, carefully assess whether the deployment context matches the training conditions, especially regarding population characteristics. Finally, implement robust monitoring systems to track fairness metrics over time and detect emerging bias.

Amazon SageMaker AI Clarify provides comprehensive tools for addressing fairness and explainability throughout the ML lifecycle. It offers bias detection capabilities for both data and models, along with explainability features through SHAP (Shapley Additive Explanations) values that provide an understanding of how individual features contribute to predictions.

Implementation steps

1. **Assess ethical implications during problem framing**. Before building an ML model, evaluate whether an algorithmic approach is ethical for your specific use case. Consider potential unintended consequences and whether the benefits outweigh potential risks. Document your assessment and decision-making process for transparency.
2. **Evaluate and prepare representative training data**. Use Amazon SageMaker AI Clarify with enhanced bias detection and new fairness metrics to analyze your training data for potential biases, particularly regarding protected attributes or sensitive groups. Verify that your data accurately represents the population on which the model will be deployed. Apply appropriate sampling or augmentation techniques to address identified representation gaps.
3. **Implement bias detection in the model development process**. Configure SageMaker AI Clarify to evaluate pre-training bias metrics that assess imbalances in your training data. These metrics can identify issues like class imbalance, label imbalance across groups, or feature distribution disparities that could lead to unfair outcomes.
4. **Select appropriate fairness metrics for evaluation**. Depending on your specific use case, choose relevant fairness metrics such as disparate impact, difference in positive proportions, or equal

opportunity difference. These metrics quantify whether your model treats different groups equitably and should be tracked alongside traditional performance metrics.

5. **Apply explainability techniques to understand model decisions**. Implement SHAP (Shapley Additive Explanations) through SageMaker AI Clarify to understand feature importance and how different features influence model predictions. This can identify whether protected attributes or proxies for them are disproportionately influencing outcomes.
6. **Consider model complexity tradeoffs with explainability requirements**. If explainability is a critical requirement, consider using simpler model architectures (like linear models or decision trees) that are inherently more interpretable. For complex models like deep neural networks, implement robust explainability tools.
7. **Implement bias monitoring in production environments**. Configure SageMaker AI Model Monitor to continuously track fairness metrics for deployed models. Set up alerts for drift in fairness metrics that might indicate emerging bias issues requiring intervention.
8. **Establish governance processes for addressing detected bias**. Create clear procedures for when bias is detected, including who is responsible for review, what remediation steps should be taken, and how stakeholders should be informed. Document these processes to verify that they are consistently applied.
9. **Train teams on fairness and explainability concepts**. Verify that data scientists, ML engineers, and other stakeholders understand fairness concepts, bias mitigation techniques, and how to interpret explainability outputs. Regular training sessions build organizational capacity for responsible AI.
10. **Document fairness and explainability considerations**. Maintain comprehensive documentation of fairness evaluations, explainability analyses, and mitigation strategies applied. This documentation supports transparency, aids regulatory adherence efforts, and communicates your responsible AI approach to stakeholders.
11. **Foundation model fairness considerations**. Use foundation models and generative AI to enhance fairness and explainability efforts by generating diverse synthetic data to address representation gaps, creating plain-language explanations of complex model decisions for different stakeholder groups, and automating the generation of comprehensive documentation about model fairness evaluations and mitigations.

Resources

Related documents:

- Model Explainability
- Feature Attributions that Use Shapley Values
- Fairness, model explainability and bias detection with SageMaker AI Clarify
- Amazon SageMaker AI Clarify
- Responsible AI Practices

Data processing

Best practices

- MLOPS03-BP01 Profile data to improve quality
- MLOPS03-BP02 Create tracking and version control mechanisms

MLOPS03-BP01 Profile data to improve quality

Data profiling is essential for understanding data characteristics such as distribution, descriptive statistics, data types, and patterns. By systematically reviewing source data for content and quality, you can filter out or correct problematic data, leading to significant quality improvements in your machine learning workflows.

Desired outcome: You gain comprehensive insights into your data's characteristics, enabling you to identify and remediate quality issues before they impact your machine learning models. Through systematic profiling, you establish a robust data preprocessing pipeline that provides high-quality, consistent data flows to your ML models, resulting in more accurate predictions and better business outcomes.

Common anti-patterns:

- Skipping data profiling and moving directly to model training.
- Manually reviewing data without automated profiling tools.
- Performing one-time data quality checks without continuous monitoring.
- Ignoring data distribution shifts between training and inference data.
- Failing to document data quality issues and their resolutions.

Benefits of establishing this best practice:

- Improved model performance through higher quality training data.

- Earlier detection of data anomalies and inconsistencies.
- Enhanced understanding of data characteristics and limitations.
- Reduced time spent debugging model issues caused by data problems.
- More transparent and reproducible machine learning workflows.
- Increased stakeholder confidence in model outputs.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Data profiling is a critical step in the machine learning workflow. By thoroughly examining your data before model training, you gain valuable insights that improve data quality and ultimately lead to better model performance. Data profiling involves analyzing the statistical properties, distributions, and patterns within your dataset to identify anomalies, missing values, outliers, and other quality issues.

Effective data profiling requires both automated tools and human judgment. While tools can quickly generate statistical summaries and visualizations, subject matter experts should interpret these findings to determine appropriate actions for data cleaning and transformation. For instance, you might discover that a numerical feature has an unexpected distribution that requires normalization, or that categorical variables contain inconsistent values requiring standardization.

Consider a retail company building a customer churn prediction model. Through data profiling, they discover that 15% of customer records have missing age values, 5% have impossibly high transaction amounts, and several categorical fields contain inconsistent formatting. By addressing these issues early, they can significantly improve their model's performance.

Implementation steps

1. **Set up Amazon SageMaker AI Unified Studio for visual data review**. Use Amazon SageMaker AI Unified Studio with enhanced collaborative features and team sharing capabilities to visually review data characteristics and remediate data-quality problems directly in your integrated environment. The unified solution provides improved debugging and monitoring capabilities for data processing workflows, automatically generating charts to identify data quality issues and suggesting transformations to fix common problems.
2. **Implement Amazon SageMaker AI Data Wrangler for comprehensive data preparation**. Import, prepare, transform, visualize, and analyze data with SageMaker AI Data Wrangler with Q integration for interactive analysis. You can integrate Data Wrangler into your ML workflows

to simplify and streamline data pre-processing and feature engineering with little to no coding. Import data from Amazon S3, Amazon Redshift, or other data sources, and then query the data using Amazon Athena. Use Data Wrangler's built-in and custom data transformations and analysis features, including feature target leakage detection and quick modeling, to create sophisticated machine learning data preparation workflows.

3. **Build an automatic data profile and reporting system**. Use AWS Glue Crawler to crawl your data sources and automatically create a data schema. The crawler detects the schema of your data and registers tables in the AWSAWS Glue Data Catalog, providing a comprehensive listing of tables and schemas. Use Amazon Athena for serverless SQL querying to constantly profile your data, and create Quick dashboards for data visualization and monitoring.
4. **Create a baseline dataset with SageMaker AI Model Monitor**. The training dataset used to train your model typically serves as a good baseline dataset. Verify that the training dataset schema and the inference dataset schema exactly match (the number and order of the features). With SageMaker AI Model Monitor, you can automatically detect concept drift in deployed models by comparing production data against this baseline.
5. **Implement continuous data quality monitoring**. Set up automated checks that continuously monitor data quality metrics like completeness, uniqueness, consistency, and validity. Configure alerts to notify relevant stakeholders when data quality issues arise, enabling prompt intervention and resolution. Use Amazon CloudWatch to create dashboards and set up alerts for key data quality metrics.
6. **Document data profiling insights and transformations**. Maintain comprehensive documentation of data profiling findings, quality issues discovered, and the transformations applied to address them. This documentation promotes transparency, facilitates knowledge sharing across teams, and supports regulatory adherence in regulated industries.
7. **Use generative AI for enhanced data profiling**. Use large language models in Amazon Bedrock or Amazon Nova to automatically extract and enrich metadata, identify patterns in your data, and generate natural language summaries of data quality issues. Generative AI can analyze unstructured data fields and provide insights that traditional data profiling tools might miss, though you should validate AI-generated suggestions before implementation.

Resources

Related documents:

- Prepare ML Data with Amazon SageMaker AI Data Wrangler
- Get Started with Data Wrangler

- Data quality
- Create a Baseline
- AWS Glue Data Quality
- Data discovery and cataloging in AWS Glue
- Amazon SageMaker AI notebooks
- What is Amazon Athena?
- What is Amazon Quick Suite?

MLOPS03-BP02 Create tracking and version control mechanisms

Machine learning model development requires robust tracking and version control mechanisms due to its iterative and exploratory nature. By implementing proper tracking systems, you can maintain visibility of your experiments, data processing techniques, and model versions while enabling reproducibility and collaboration.

Desired outcome: You have comprehensive tracking of ML model development with experiment tracking capabilities, version-controlled data processing code, and a model registry that enables you to identify the best performing models. Your development processes are reproducible, collaborative, and automated with CI/CD pipelines for model deployment.

Common anti-patterns:

- Manually tracking experiments in spreadsheets or documents.
- Not documenting data processing steps or model configurations.
- Keeping models and datasets in local environments without version control.
- Starting new experiments from scratch instead of building on previous work.

Benefits of establishing this best practice:

- Enhanced reproducibility of experiments and model training.
- Improved collaboration among data science teams.
- Better visibility into the performance of different model iterations.
- Faster identification of the best performing models.
- Ability to roll back to previous model versions if needed.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Machine learning development involves experimenting with multiple combinations of data, algorithms, and parameters while observing how incremental changes impact model accuracy. Without proper tracking and version control, you risk losing valuable insights and the ability to reproduce successful experiments.

To address these challenges, you need a systematic approach to track experiments, version control your code and data processing techniques, and manage model deployment. AWS SageMaker AI provides integrated tools for experiment tracking, version control, and model registry that can streamline your machine learning operations.

By implementing proper tracking and versioning mechanisms, you can document your model development journey, track performance metrics across experiments, and reliability reproduce and deploy your models. This creates a foundation for continuous improvement of your machine learning applications.

Implementation steps

1. **Track your ML experiments with SageMaker AI Experiments**. Use Amazon SageMaker AI Experiments to you create, manage, analyze, and compare your machine learning experiments. SageMaker AI Experiments automatically tracks inputs, parameters, configurations, and results of your iterations as runs. You can assign, group, and organize these runs into experiments. SageMaker AI Experiments integrates with Amazon SageMaker AI Studio, providing a visual interface to browse active and past experiments, compare runs on key performance metrics, and identify the best-performing models.
2. **Process data with SageMaker AI Processing**. For analyzing data, documenting processing, and evaluating ML models, use Amazon SageMaker AI Processing. This capability can be used for feature engineering, data validation, model evaluation, and model interpretation. SageMaker AI Processing provides a standardized way to run your data processing workloads, fostering consistency and reproducibility.
3. **Use SageMaker AI Unified Studio for enhanced collaboration**. Use Amazon SageMaker AI Unified Studio with enhanced collaborative features and team sharing capabilities for integrated data and AI workflows. The unified solution provides improved debugging and monitoring capabilities, VS Code server integration, and enhanced project sharing. This approach keeps your data processing code and documentation accessible while facilitating better collaboration and version tracking across teams.

4. **Use SageMaker AI Model Registry**. Catalog, manage, and deploy models using SageMaker AI Model Registry. Create a model group and, for each run of your ML pipeline, create a model version that you register in the model group. The Model Registry provides a centralized repository for model versions, making it more straightforward to track model lineage, compare model performance, and promote models to production.
5. **Implement CI/CD for model deployment**. Automate your model deployment process using CI/CD pipelines for consistent and reliable deployments. SageMaker AI Pipelines can be used to create end-to-end workflows that include model building, evaluation, and deployment steps. Implement CI/CD for model deployment to thoroughly test your models before they are deployed to production, reducing the risk of deployment-related issues.
6. **Integrate data version control**. Use tools like Data Version Control (DVC) in conjunction with SageMaker AI Experiments to track both your model code and the datasets used for training and evaluation. With these tools, you can completely reproduce your machine learning experiments.
7. **Create model cards for documentation**. For each model version in your registry, create comprehensive SageMaker AI Model Cards with enhanced documentation and governance capabilities that document the model's purpose, training data, performance metrics, limitations, and usage guidelines. This documentation helps users understand when and how to use specific model versions and supports improved model governance.

Resources

Related documents:

- Amazon SageMaker AI Experiments in Studio Classic
- Accelerate generative AI development using managed MLflow on Amazon SageMaker AI
- Git repositories with SageMaker AI Notebook Instances
- MLOps Automation With SageMaker AI Projects
- Amazon SageMaker AI Model Cards
- Model Registration Deployment with Model Registry
- Data transformation workloads with SageMaker AI Processing
- Pipelines

Related examples:

- SageMaker AI Experiment Examples
- SageMaker AI Model Registry Examples

- Amazon SageMaker AI processing

Model development

Best practices

- MLOPS04-BP01 Automate operations through MLOps and CI/CD
- MLOPS04-BP02 Establish reliable packaging patterns to access approved public libraries

MLOPS04-BP01 Automate operations through MLOps and CI/CD

Automate ML workload operations using infrastructure as code (IaC) and configuration as code (CaC). Select appropriate MLOps mechanisms to orchestrate your ML workflows and integrate with CI/CD pipelines for automated deployments. This approach creates consistency across your staging and production deployment environments. Enable model observability and version control across your hosting infrastructure.

Desired outcome: You establish automated ML operations through MLOps practices and CI/CD pipelines for repeatable, consistent deployments across environments. Your ML workflows are orchestrated through infrastructure as code, providing traceability, version control, and model observability. This enables your teams to deliver ML models faster, with higher quality, and maintain governance throughout the model lifecycle from development to production.

Common anti-patterns:

- Manually deploying ML models to production environments.
- Using different tools and processes across development and production environments.
- Not versioning infrastructure, configuration, or model artifacts.
- Lacking automated testing for ML models before deployment.
- Creating one-off scripts for deployment instead of reusable templates.

Benefits of establishing this best practice:

- Accelerated ML model development and deployment cycles.
- Consistent, reproducible environments across development and production.
- Improved collaboration between data scientists and operations teams.
- Enhanced governance and traceability for model artifacts and infrastructure.
- Improved rollbacks and version management.

Level of risk exposed if this best practice is not established: High

Implementation guidance

MLOps combines machine learning, DevOps practices, and data engineering to streamline and automate the end-to-end ML lifecycle. By implementing infrastructure as code and configuration as code principles, you create consistent, reproducible environments while minimizing manual steps that can introduce errors. This approach makes ML operations more reliable, scalable, and maintainable.

Creating automated CI/CD pipelines for ML workloads requires special consideration compared to traditional software applications. You need to track not only code changes but also data, model parameters, and training configurations. Using AWS services for MLOps provides integrated tools to manage these complexities while maintaining proper governance and observability.

When implementing MLOps practices, start by defining your workflow patterns and choosing appropriate orchestration tools based on your specific requirements. AWS offers multiple options for ML workflow orchestration, from purpose-built services like SageMaker AI Pipelines to more general workflow engines like AWS Step Functions. Each provides different levels of abstraction, control, and integration capabilities.

Model observability is crucial for ML systems in production, allowing you to monitor model performance, detect drift, and run retraining when necessary. Implementing comprehensive monitoring assists you to quickly identify and respond to changes in model behavior or data distributions.

Implementation steps

1. **Define your ML workflow architecture**. Begin by mapping out your end-to-end ML workflow, including data preparation, feature engineering, model training, evaluation, deployment, and monitoring stages. Identify which steps can be automated and which require human intervention. Determine the appropriate level of separation between development, testing, and production environments based on your requirements.
2. **Select an infrastructure as code approach**. Choose either AWS CloudFormation or AWS CDK to define your infrastructure. AWS CloudFormation enables you to create and provision AWS deployments predictably and repeatedly using template files. For teams more comfortable with programming languages, AWS Cloud Development Kit (AWS CDK) (AWS CDK) allows you to define cloud resources using familiar languages like Python, TypeScript, or Java.

3. **Implement version control for assets**. Establish a version control strategy for ML assets including code, configurations, infrastructure definitions, and model artifacts. Use AWS CodeCommit or third-party repositories to store and version these assets. Implement branching strategies that allow for experimentation while maintaining stable production environments. Version control enables you to track changes, collaborate effectively, and roll back to previous versions when needed.
4. **Choose an MLOps orchestration strategy**. Based on your workflow needs, select an appropriate orchestration mechanism:
   - Use Amazon SageMaker AI Pipelines to create ML workflows with Python SDK, visualize, and manage them in Amazon SageMaker AI Studio. SageMaker AI Pipelines automatically logs every step, creating an audit trail of model components including training data, configurations, parameters, and learning gradients.
   - Use AWS Step Functions Data Science SDK to automate ML workflows with more complex orchestration requirements or when integrating with other AWS services beyond SageMaker AI.
   - Use third-party tools such as Amazon Managed Workflows for Apache Airflow (MWAA) to orchestrate workflows using Directed Acyclic Graphs (DAGs) written in Python, especially when you need to integrate with existing Apache Airflow deployments.
5. **Build CI/CD pipelines for ML models**. Implement CI/CD pipelines using AWS CodePipeline to automate the building, testing, and deployment of ML models. Include automated tests for data quality, model performance, and API functionality. Configure the pipeline to deploy to staging environments before production and implement approval gates where needed. Integrate model registration and versioning using Amazon SageMaker AI Model Registry.
6. **Set up model monitoring and observability**. Implement comprehensive monitoring for your deployed models using Amazon SageMaker AI Model Monitor. Configure data quality monitoring, model quality monitoring, bias drift monitoring, and feature attribution drift monitoring. Use Amazon CloudWatch to create dashboards and alerts for model performance metrics.
7. **Establish automated rollback mechanisms**. Configure your deployment pipelines to support automated rollbacks when quality thresholds are not met. Implement canary or blue/green deployment strategies using AWS CodeDeploy to gradually shift traffic to new model versions while monitoring for issues. This minimizes the impact of problematic deployments and provides service continuity.
8. **Integrate security and governance controls**. Implement security checks throughout your MLOps pipeline. Use AWS Identity and Access Management (IAM) to control access to resources

and AWS CloudTrail to log API calls for auditing. Configure Amazon SageMaker AI Model Cards to document model information, intended uses, limitations, and performance characteristics.

9. **Create environments for experimentation and testing**. Set up isolated environments for experimentation that don't impact production systems. Use Amazon SageMaker AI Unified Studio to provide data scientists with self-service environments for exploration while maintaining governance through integrated data and AI workflows. Implement environment-specific configurations through parameter files or environment variables managed in your IaC templates.

Resources

Related documents:

- Pipelines overview
- What is AWS CodePipeline?
- Amazon Managed Workflows for Apache Airflow (MWAA)
- AWS CloudFormation Documentation
- What is the AWS CDK?
- Step Functions Data Science SDK
- Infrastructure as code
- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- Model Registration Deployment with Model Registry

Related videos:

- AWS re:Invent 2024 - Accelerate ML workflows with Amazon SageMaker AI Studio

MLOPS04-BP02 Establish reliable packaging patterns to access approved public libraries

Establishing reliable packaging patterns enables data scientists to access approved public libraries efficiently and consistently. By implementing structured access to public libraries and creating separate kernels for common ML frameworks, organizations can have both flexibility and security in their machine learning development environments.

Desired outcome: You create a streamlined workflow where your data scientists have reliable access to approved libraries through internal repositories. You maintain separate kernels for

common ML frameworks like TensorFlow, PyTorch, Scikit-learn, and Keras. This approach improves development productivity while improving security and adherence with organizational requirements.

Common anti-patterns:

- Allowing uncontrolled direct downloads from public repositories.
- Using inconsistent or undocumented container configurations.
- Maintaining duplicate library versions across different environments.
- Not having a centralized strategy for package management.
- Failing to version control dependencies.

Benefits of establishing this best practice:

- Better security through controlled library usage.
- Improved reproducibility of ML workloads.
- Reduced dependency conflicts.
- Simplified adherence with organizational security policies.
- Faster onboarding of new data scientists.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Creating reliable packaging patterns is essential for maintaining consistent, secure, and efficient machine learning workflows. You need to establish infrastructure that gives data scientists access to the libraries they need while maintaining organizational control over those dependencies. Container technologies provide a portable, consistent way to package ML environments with required dependencies, making them ideal for this purpose. Internal artifact repositories allow for centralized management of approved packages so that team members use consistent, vetted versions.

Your packaging strategy should balance flexibility for data scientists with security and regulatory requirements. This means establishing both infrastructure (containers, repositories) and processes (approval workflows, versioning policies) that work together to create a streamlined experience. The use of standardized environments with separate kernels for different ML frameworks enables isolation when needed while providing the specific tools required for different types of projects.

Implementation steps

1. **Set up container infrastructure for ML workloads**. Create a container strategy using Amazon Elastic Container Registry (Amazon ECR) to store and manage your ML container images. Containers provide consistent environments that package dependencies, libraries, and runtime components needed for ML workloads.
2. **Create base container images for common ML frameworks**. Build and maintain separate base container images for frameworks like TensorFlow, PyTorch, Scikit-learn, and Keras using optimized foundation model containers for efficient training and inference. These images should include standard configurations and commonly used libraries for each framework, with support for frameworks like Hugging Face Transformers and quantization techniques, providing consistency across your organization.
3. **Implement versioning and tagging policies**. Establish clear policies for versioning containers and artifacts for reproducibility of ML experiments and models. Use semantic versioning and proper tagging to track container image changes and library updates.
4. **Develop automation for container builds and updates**. Implement CI/CD pipelines using AWS CodePipeline to automatically build, test, and deploy updated container images when dependencies need to be updated or security patches are required.
5. **Document usage patterns and onboarding procedures**. Create comprehensive documentation that explains how data scientists should use the established packaging patterns, including how to access approved libraries and work with containerized environments.

Resources

Related documents:

- What is Amazon Elastic Container Registry?
- What are AWS Deep Learning Containers?
- Docker containers for training and deploying models
- What is AWS CodePipeline?
- "Publish Docker image to an Amazon ECR image repository" sample for CodeBuild

Related examples:

- SageMaker AI Custom Container Examples
- Deep Learning Container Examples

Deployment

Best practices

- MLOPS05-BP01 Establish deployment environment metrics

MLOPS05-BP01 Establish deployment environment metrics

Establish comprehensive monitoring of machine learning operations to gain visibility into your deployed ML environments. By measuring metrics such as memory, CPU/GPU usage, disk utilization, endpoint invocations, and latency, you can provide optimal performance and improve the reliability of your ML systems.

Desired outcome: You gain real-time visibility into your ML deployment environments through systematic collection and analysis of performance metrics. You establish robust monitoring dashboards, alerting systems, and automated responses to potential issues. This enables you to proactively address performance bottlenecks, optimize resource utilization, and maintain reliable ML services while managing costs effectively.

Common anti-patterns:

- Implementing monitoring only after experiencing production issues.
- Focusing only on basic system metrics while ignoring ML-specific performance indicators.
- Creating monitoring dashboards without establishing corresponding alerts.
- Setting static thresholds that don't account for typical usage patterns.
- Collecting metrics without regular review or actionable response plans.

Benefits of establishing this best practice:

- Early detection of performance issues before they impact end users.
- Improved resource optimization and cost efficiency.
- Enhanced ability to scale ML services based on actual usage patterns.
- Faster troubleshooting and reduced mean time to resolution.
- Data-driven capacity planning and infrastructure decisions.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Establishing effective deployment environment metrics requires a systematic approach to monitoring both infrastructure and application-specific ML metrics. You need to focus on both system-level metrics like CPU, memory, and disk usage, as well as ML-specific metrics such as inference latency, throughput, and model accuracy. By using AWS services like Amazon CloudWatch, you can centralize your monitoring approach and gain comprehensive visibility into your ML operations.

Start by identifying the key performance indicators that matter most for your specific ML workloads. For inference endpoints, these typically include latency, throughput, and resource utilization. For batch processing jobs, focus on processing time, error rates, and resource efficiency. With these KPIs established, you can implement automated monitoring and alerting to quickly identify when your systems deviate from expected performance.

Regular review of your metrics can identify trends and potential areas for optimization. For example, you might discover that certain models require more memory than originally allocated, or that specific types of inference requests consistently take longer to process. This information guides your infrastructure decisions and prioritizes optimization efforts.

Implementation steps

1. **Record performance-related metrics.** Use a monitoring and observability service like Amazon CloudWatch to record performance-related metrics. These metrics can include database transactions, slow queries, I/O latency, HTTP request throughput, service latency, and other key data. For SageMaker AI endpoints, collect metrics such as CPUUtilization, MemoryUtilization, DiskUtilization, and model-specific metrics like ModelLatency and InvocationsPerInstance.
2. **Analyze metrics when events or incidents occur.** Use monitoring dashboards and reports to understand and diagnose the impact of an event or incident. Amazon CloudWatch Dashboards provide insight into what portions of the workload are not performing as expected. Correlate metrics, logs, and traces to quickly identify root causes when issues arise.
3. **Establish key performance indicators (KPIs) to measure workload performance.** Identify the KPIs that indicate whether the workload is performing as intended. An API-based ML endpoint might use overall response latency as an indication of overall performance, while a batch processing job might track throughput metrics. For generative AI applications, monitor additional metrics like token usage, cost per request, prompt engineering effectiveness, and use SageMaker AI Training Plans for cost optimization of large-scale AI training workloads.

4. **Use monitoring to generate alarm-based notifications.** Monitor metrics for the defined KPIs and generate alarms automatically when the measurements are outside expected boundaries. Configure Amazon CloudWatch Alarms to send notifications using Amazon SNS when thresholds are breached, which provides for timely responses to potential issues.
5. **Review metrics at regular intervals.** As routine maintenance, or in response to events or incidents, review what metrics are collected and identify the metrics that were key in addressing issues. Identify additional metrics that can identify, address, or avoid issues. Schedule periodic reviews to keep your monitoring strategy effective as your ML applications evolve.
6. **Monitor and alarm proactively.** Use KPIs, combined with monitoring and alerting systems, to proactively address performance-related issues. Configure CloudWatch to initiate automated actions to remediate issues where possible. Escalate the alarm to those able to respond if an automated response is not possible. Use anomaly detection to predict expected KPI values, and generate alerts and automatically halt or roll back deployments if KPIs are outside of the expected values.
7. **Use Amazon CloudWatch for comprehensive monitoring.** Use Amazon CloudWatch metrics for SageMaker AI endpoints to determine the memory, CPU usage, and disk utilization. Set up CloudWatch Dashboards to visualize the environment metrics and establish CloudWatch alarms to initiate a notification through Amazon SNS (email, SMS, or webHook) to notify on events occurring in the runtime environment. For model performance monitoring, implement Amazon SageMaker AI Model Monitor to detect data drift and quality issues.
8. **Use Amazon EventBridge for automated workflows.** Define an automated workflow using Amazon EventBridge to respond automatically to events. These events can include training job status changes, endpoint status changes, and increasing the compute environment capacity after it crosses a defined threshold (such as CPU or disk utilization). Create event rules that run Lambda functions to scale resources, update configurations, or notify specific teams based on the event type.
9. **Use AWS Application Cost Profiler for cost allocation.** Implement AWS Application Cost Profiler to report the cost per tenant (model or user). This assists you in tracking resource usage at a granular level and enables accurate cost attribution across different ML models, teams, or business units. Use SageMaker AI Training Plans for compute reservation and better resource planning of high-demand GPU resources. Use these insights to optimize resource allocation and identify opportunities for cost savings.
10. **Implement SageMaker AI Model Monitor for data drift detection.** Configure SageMaker AI Model Monitor to continuously monitor the quality of your machine learning models in production. Set up automated quality checks to detect data drift, model drift, and bias drift

in your production workloads. This enables you to maintain high quality standards and take corrective actions when models start to deviate from expected behavior.

11. **Use AWS X-Ray for distributed tracing.** Implement AWS X-Ray to trace requests through your ML application components. This provides visibility into request flows, latency bottlenecks, and error points in distributed systems. X-Ray provides an understanding of the dependencies between components so you can optimize end-to-end performance of your ML applications.
12. **Implement Amazon SageMaker AI Clarify for bias monitoring.** Use SageMaker AI Clarify to detect bias in your deployed models. Set up regular monitoring jobs to track bias metrics and alert when they exceed acceptable thresholds so that your ML systems continue to make fair and unbiased predictions in production.

Resources

Related documents:

- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- Metrics in Amazon CloudWatch
- Using Amzon CloudWatch dashboards
- DevOps and AWS

Monitoring

Best practices

- MLOPS06-BP01 Synchronize architecture and configuration, and check for skew across environments
- MLOPS06-BP02 Enable model observability and tracking

MLOPS06-BP01 Synchronize architecture and configuration, and check for skew across environments

Synchronize your systems and configurations across development and deployment phases for consistent machine learning model inference results. By maintaining identical environments, you can avoid discrepancies that arise from architectural differences, leading to more reliable and predictable model performance.

Desired outcome: You have established a systematic approach to foster architectural and configuration consistency across development, staging, and production environments for machine learning models. This includes automated infrastructure deployment, continuous model quality monitoring, and proactive detection of environmental skew. Your machine learning systems deliver the same range of accuracy regardless of which environment they run in, and you can quickly identify and address deviations.

Common anti-patterns:

- Manually configuring each environment, leading to inconsistencies.
- Neglecting to validate model performance across different environments.
- Assuming model behavior will be identical across environments without verification.
- Using different hardware specifications or software versions between environments.
- Making changes only when needed to production environments without documenting or replicating them in other environments.

Benefits of establishing this best practice:

- Consistent model performance across environments.
- Reduced debugging time for environment-specific issues.
- Improved reliability of machine learning applications.
- Straightforward identification of the root causes of performance deviations.
- Streamlined promotion process from development to production.
- Enhanced confidence in deployed models.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Synchronizing your machine learning environments is critical for checking that a model trained in one environment behaves the same way when deployed in another. Differences in system architectures, software dependencies, or configuration settings can cause unexpected variations in model performance, leading to inaccurate predictions or even system failures.

By treating your infrastructure as code and implementing automated monitoring, you can maintain consistency across environments and quickly detect deviations. This approach allows you to focus on improving your models rather than debugging environment-specific issues. It also provides a reliable foundation for continuous delivery of machine learning solutions.

Environmental skew can occur in various ways, such as through differences in hardware capabilities, software versions, or system configurations. For example, a model trained on a development environment with specific CPU architecture might behave differently when deployed to a production environment with different specifications. Similarly, differences in underlying libraries or dependencies can lead to subtle variations in model behavior.

Regular validation of model performance across environments should be part of your standard promotion process. This includes comparing not only the accuracy metrics but also the distribution of predictions and the model's behavior for edge cases.

Implementation steps

1. **Define infrastructure as code using AWS CloudFormation**. Create CloudFormation templates that define resources, configurations, and dependencies for your machine learning environments. With this strategy, each environment is provisioned consistently and can be recreated identically when needed. Include compute resources, networking configurations, security settings, and machine learning-specific components.
2. **Implement version control for infrastructure templates**. Store your CloudFormation templates in a version control system like AWS CodeCommit or GitHub. This allows you to track changes over time, roll back to previous configurations if needed, and verify that your environments are using the same version of the infrastructure definition.
3. **Set up CI/CD pipelines for infrastructure deployment**. Use AWS CodePipeline or AWS CodeBuild to automate the deployment of your infrastructure changes across environments. This reduces manual intervention and the potential for human error when updating environments.
4. **Configure Amazon SageMaker AI Model Monitor for continuous quality evaluation**. Set up Model Monitor to automatically track the quality of your models in production and compare the results with the baseline established during training. This can identify when model performance starts to drift due to environmental factors or data changes.
5. **Implement data quality monitoring**. Use SageMaker AI Model Monitor's data quality monitoring capability to detect changes in the statistical properties of your input data across environments. This capability provides similar input distributions to your models regardless of environment.
6. **Set up model quality monitoring**. Configure SageMaker AI Model Monitor to track model quality metrics such as accuracy, precision, and recall over time. Compare these metrics between your development, staging, and production environments to detect inconsistencies.

7. **Enable bias drift monitoring**. Use SageMaker AI Model Monitor's bias drift monitoring to detect if your models exhibit different biases in different environments, which could indicate environment-specific issues.
8. **Configure alerts for deviations**. Set up Amazon CloudWatch alarms to notify you when SageMaker AI Model Monitor detects significant deviations in model performance or data characteristics across environments. This allows for proactive intervention before small issues become major problems.
9. **Establish a promotion checklist**. Create a formal checklist that includes verifying model performance consistency across environments before promoting a model to the next stage. Document the acceptable thresholds for performance differences between environments.
10. **Implement regular cross-environment validation**. Schedule periodic validation of model performance across your environments, even when no changes are being made. Use this validation to catch gradual drift that might occur due to external factors.
11. **Create environment comparison dashboards**. Use Quick with GenBI capabilities to automatically generate visualizations and dashboards for model performance metrics across different environments, making it more straightforward to spot discrepancies and track trends over time.
12. **Utilize foundation models for anomaly detection**. Implement Amazon Bedrock or SageMaker AI JumpStart with expanded library of foundation models to analyze performance patterns and identify anomalous behavior that might indicate environmental inconsistencies, especially for complex ML systems where traditional monitoring might miss subtle differences.

Resources

Related documents:

- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- What is AWS CloudFormation?
- What is AWS Config?
- Detect unmanaged configuration changes to stacks and resources with drift detection
- AWS Systems Manager Parameter Store
- Amazon SageMaker AI best practices

Related examples:

- Introduction to Amazon SageMaker AI Model Monitor

- Model Monitor Visualization

MLOPS06-BP02 Enable model observability and tracking

Establish model monitoring mechanisms to identify and proactively avoid inference issues. ML models can degrade in performance over time due to drifts. Monitor metrics that are attributed to your model's performance. For real time inference endpoints, measure the operational health of the underlying compute resources hosting the endpoint and the health of endpoint responses. Establish lineage to trace hosted models back to versioned inputs and model artifacts for analysis.

Desired outcome: You can continuously monitor your machine learning models in production to detect and avoid performance degradation over time. You have mechanisms in place to track model lineage, identify various types of drift, and receive alerts when models deviate from expected behavior. Your monitoring solution provides clear visibility into both the technical health of model endpoints and the business performance of the models themselves, enabling you to maintain high-quality predictions and make informed decisions about model updates.

Common anti-patterns:

- Deploying models without monitoring capabilities.
- Failing to establish model lineage tracking for audit and governance.
- Not monitoring for data drift or concept drift in production models.
- Ignoring model bias and fairness considerations after deployment.
- Lacking documentation of model information and performance metrics.

Benefits of establishing this best practice:

- Early detection of model performance degradation.
- Improved model governance and adherence through comprehensive documentation.
- Enhanced ability to explain model predictions and address bias concerns.
- Reduced operational risk through proactive monitoring of model health.
- Streamlined model updates and improvements based on real-world performance data.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Model observability is critical for maintaining the reliability, fairness, and performance of machine learning systems in production. Without proper monitoring mechanisms, ML models can silently degrade over time as the data they process begins to differ from the data they were trained on, a phenomenon known as drift.

You need to implement comprehensive monitoring across several dimensions: data quality to check that inputs remain consistent with training data, model quality to track performance metrics, bias detection to verify fairness, and explainability to understand model decisions. Additionally, model lineage tracking can trace issues back to specific model versions, training datasets, and hyperparameters.

Amazon SageMaker AI provides integrated tools that make implementing these monitoring capabilities straightforward. SageMaker AI Model Monitor can automatically detect deviations in your model's data and performance characteristics, while SageMaker AI Clarify identifies bias and explains predictions. By setting up proper alerts, you can be notified when issues arise and take corrective action before they impact your business.

Documentation is equally important for model governance. SageMaker AI Model Cards provide a centralized location to store important model information, including performance metrics, intended use cases, and potential limitations.

Implementation steps

1. **Set up Amazon SageMaker AI Model Monitor**. Configure Amazon SageMaker AI Model Monitor to automatically monitor the quality of your ML models in production. Create baseline statistics and constraints during model training, then monitor for deviations in production data. Set up the following types of monitoring:
   - Data quality monitoring to detect changes in data distributions
   - Model quality monitoring to track accuracy and other performance metrics
   - Bias drift monitoring to detect changes in model fairness
   - Feature attribution drift monitoring to track changes in feature importance
2. **Integrate with Amazon CloudWatch**. SageMaker AI Model Monitor automatically sends metrics to Amazon CloudWatch, allowing you to track usage statistics for your ML models. Set up CloudWatch dashboards to visualize key metrics and create alarms that go off when metrics exceed predefined thresholds. Configure notifications through Amazon SNS to alert relevant teams when issues are detected.

3. **Implement SageMaker AI Model Dashboard**. Use the SageMaker AI Model Dashboard to gain a centralized view of your models. From the SageMaker AI console, you can search, view, and explore your models in one place. Set up monitors to track the performance of models deployed on real-time inference endpoints and identify models that violate thresholds for data quality, model quality, bias, and explainability.
4. **Enable bias detection with SageMaker AI Clarify**. Deploy SageMaker AI Clarify to identify various types of bias that can emerge during model training or when the model is in production. Configure both pre-training and post-training bias metrics to understand how your model's predictions affect different segments of your user base. Use Clarify's monitoring capabilities to detect bias drift in production models.
5. **Implement model explainability**. Configure SageMaker AI Clarify's feature attribution capabilities to explain how your models make predictions. This builds trust with stakeholders and can identify potential issues with model logic. Set up monitoring to detect when feature attribution patterns drift from baseline, which could indicate underlying problems with the model.
6. **Establish ML lineage tracking**. Implement SageMaker AI ML Lineage Tracking to create and store information about each step in your machine learning workflow, from data preparation to model deployment. This creates a running history of your ML experiments and establishes model governance by tracking model lineage artifacts for auditing and adherence verification.
7. **Create Model Cards for documentation**. Use Amazon SageMaker AI Model Cards with enhanced documentation and governance capabilities to document critical information about your models in a single location. Include business requirements, key decisions, observations during development, performance goals, risk ratings, and evaluation results. This streamlines documentation throughout a model's lifecycle and supports approval workflows, registration, and audits.
8. **Implement shadow testing for model validation**. Before deploying new model versions to production, use Amazon SageMaker AI Shadow Testing to compare the performance of new models against production models using real-world inference request data. Configure SageMaker AI to route copies of production inference requests to the new model variant and generate dashboards displaying performance differences across key metrics in real-time.

Resources

Related documents:

- Data and model quality monitoring with Amazon SageMaker AI Model Monitor

- Amazon SageMaker AI Model Dashboard
- Shadow tests
- Lineage Tracking Entities
- Using CloudWatch outlier detection
- Monitoring Amazon ML with Amazon CloudWatch Metrics
- New for Amazon SageMaker AI – Perform Shadow Tests to Compare Inference Performance Between ML Model Variants
- Integrate Amazon SageMaker AI Model Cards with the model registr
- Improve governance of your machine learning models with Amazon SageMaker AI

Related examples:

- Monitoring bias drift and feature attribution drift with Amazon SageMaker AI Clarify

Security

The security pillar encompasses the ability to protect data, systems, and assets to take advantage of cloud technologies to improve your security posture. This section includes best practices to consider while identifying the business goal.

Each best practice in this section is presented based on its place in the ML lifecycle as detailed in *Machine learning lifecycle*.

Lifecycle phases

- Business goal identification
- ML problem framing
- Data processing
- Model development
- Deployment
- Monitoring

Business goal identification

Best practices

- MLSEC01-BP01 Validate ML data permissions, privacy, software, and license terms

MLSEC01-BP01 Validate ML data permissions, privacy, software, and license terms

Machine learning implementations require careful consideration of data permissions, privacy, and software licensing to adhere to organizational and legal requirements. Validating these elements throughout the ML lifecycle builds trusted ML systems that respect data rights while delivering business value.

Desired outcome: Establish a robust governance process that verifies that ML data usage and software implementations meets your organization's requirements. Maintain clear documentation of data permissions, approved software packages, and license adherence. Operate ML

implementations within a framework that respects data subject rights, follows privacy regulations, and avoids legal complications related to software licensing.

Common anti-patterns:

- Assuming data collected for one purpose can automatically be used for ML training without additional consent.
- Installing ML libraries and packages without reviewing their license terms or data collection practices.
- Failing to document data permissions and consent mechanisms for adherence verification.
- Ignoring the need for a process to handle withdrawn consent from data subjects.
- Using third-party ML models without understanding their privacy implications or license restrictions.

Benefits of establishing this best practice:

- Reduces legal and regulatory risks related to data privacy and software licensing.
- Enhances trust from data subjects and stakeholders through ethical data handling.
- Avoids unexpected limitations on business plans due to restrictive license terms.
- Improves documentation for audits and regulatory requirements.
- Streamlines deployment through pre-validated software and container solutions.

Level of risk exposed if this best practice is not established: High

Implementation guidance

ML libraries and packages handle various aspects of the machine learning lifecycle, from data processing and model development to training and hosting. Each component may come with specific license terms, privacy considerations, and data handling requirements. Validating these elements verifies your organization's ML initiatives adhere to regulatory requirements and don't introduce unexpected limitations.

When implementing ML systems, verify that data being used has proper permissions for ML applications specifically. This often means going beyond general data collection consent to verify explicit permission for ML usage. For example, if you collected customer data for service delivery, you may need additional consent to use that data for training ML models.

Understanding license implications is critical for software components. Some open-source ML libraries may have restrictions that could affect your ability to commercialize models trained with

them. Similarly, third-party models or APIs might include terms that grant the provider certain rights to your data or restrict how you can deploy solutions.

Privacy considerations extend beyond initial data collection to the entire ML lifecycle. Establish mechanisms to handle data subject requests, including the right to withdraw consent or be forgotten. Your ML implementation should respect these requests without compromising the entire system.

Implementation steps

1. **Attain data permissions for ML**. Verify whether your intended data can be used for machine learning specifically. Document the legitimate business purpose for using the data, and determine whether you need additional consent from data owners or subjects. Implement a process to handle data subjects who withdraw their consent, including the ability to remove their data from training sets or models when required. Maintain comprehensive documentation of data permissions for compliance-aligned purposes and potential audits.
2. **Create a software license inventory**. Develop and maintain an inventory of ML libraries, packages, and dependencies used in your ML pipeline. For each component, document the license type, key terms, restrictions, and implications for your business model. Use tools like AWS License Manager to track and manage software licenses across your ML environments, improving adherence to licensing agreements and optimizing license usage.
3. **Bootstrap instances with lifecycle management policies**. Create lifecycle configurations with references to your approved package repositories and scripts to install required packages. This improves consistency across development environments and avoids the introduction of unauthorized packages. Implement Amazon SageMaker AI Lifecycle Configurations to automate the setup of development environments with pre-approved software packages and configurations.
4. **Evaluate package integrations that require external lookup services**. Based on your data privacy requirements, opt out of data collection features when necessary. Minimize data exposure by establishing trusted relationships with service providers and understanding their data handling practices. Evaluate privacy policies and license terms for ML packages that might collect telemetry or other data. For sensitive implementations, consider creating private mirrors of required packages to maintain control over external connections.
5. **Use prebuilt containers**. Start with pre-packaged and verified containers to quickly provide support for commonly used dependencies while improving license adherence. AWS Deep Learning Containers contain several deep learning framework libraries and tools including TensorFlow, PyTorch, and Apache MXNet with pre-validated license terms. These containers

maintain consistency while reducing the risk of introducing unauthorized or incompatible packages.

6. **Establish a privacy-preserving ML workflow**. Implement data minimization principles by using only the data necessary for your ML tasks. Apply anonymization or pseudonymization techniques to sensitive data before using it for training. Consider using privacy-preserving ML techniques such as differential privacy or federated learning for highly sensitive applications. Document your privacy-preserving measures for compliance-aligned purposes and to build trust with stakeholders.
7. **Monitor for license and privacy adherence**. Implement continuous monitoring of your ML environments to detect potential license violations or privacy issues. Create automated checks for package versions and license changes during CI/CD processes. Regularly audit data access patterns to verify that they comply with documented permissions and privacy requirements. Establish a process for addressing issues when they arise.
8. **Consider synthetic data for training and testing**. Create synthetic datasets that preserve the statistical properties of real data while avoiding privacy concerns. Amazon SageMaker AI provides capabilities for generating synthetic data for training and testing ML models when using real data presents privacy or licensing challenges. Document the use of synthetic data in your ML pipelines to demonstrate privacy-preserving practices.

Resources

Related documents:

- Protecting compute
- What is AWS Deep Learning Containers?
- AWS License Manager
- Lifecycle configurations within Amazon SageMaker AI Studio
- Data Privacy in Amazon SageMaker AI
- Best practices for endpoint security and health with Amazon SageMaker AI
- Private package installation in Amazon SageMaker AI running in internet-free mode
- Machine Learning Best Practices in Financial Services

Related videos:

- Machine Learning Best Practices in Financial Services

ML problem framing

Best practices

- MLSEC02-BP01 Design data encryption and obfuscation

MLSEC02-BP01 Design data encryption and obfuscation

Consider how to protect personal data. Use field level encryption or obfuscation to protect personally identifiable data.

Desired outcome: You establish robust protection for sensitive information by implementing data encryption and obfuscation techniques in your machine learning workflows. You identify and secure personally identifiable information (PII) through field-level encryption and data masking, which improves your adherence to privacy regulations while maintaining data utility for ML models.

Common anti-patterns:

- Storing personally identifiable information in plain text format.
- Using the same encryption keys across different environments.
- Implementing inconsistent data protection policies across ML pipelines.
- Overlooking data protection requirements during the design phase.
- Failing to audit data for attributes requiring special treatment.

Benefits of establishing this best practice:

- Enhanced protection of sensitive and personally identifiable data.
- Improves adherence to data privacy regulations.
- Reduced risk of data breaches and unauthorized access.
- Improved trust from users and stakeholders.
- Ability to utilize sensitive data for ML training while maintaining privacy.

Level of risk exposed if this best practice is not established: High

Implementation guidance

When designing machine learning workflows, protect personal and sensitive data throughout the entire data lifecycle. You should evaluate your data early in the process to identify fields containing

PII or other sensitive information requiring protection. Implementing field-level encryption or data obfuscation techniques maintains data utility for machine learning while safeguarding individual privacy.

AWS provides multiple services to identify, classify, and protect sensitive data within your ML workflows. Services like AWS Glue can automatically detect PII, while AWS Key Management Service (KMS) and AWS CloudHSM support robust encryption strategies. You should establish consistent policies for handling sensitive data across your organization and regularly audit your data protection measures to improve your adherence to privacy regulations.

Implementation steps

1. **Audit data for attributes requiring special treatment**. Identify fields containing data requiring special treatment, such as field-level encryption, data masking, or obfuscation. Use automated tools like AWS Glue to identify PII and sensitive data patterns within your datasets.
2. **Establish a data classification framework**. Develop a systematic approach to categorize data based on sensitivity levels. Define which categories require encryption, masking, or other protection techniques, and document these requirements in your organization's security policies.
3. **Implement field-level encryption**. Apply encryption selectively to sensitive fields rather than entire datasets. Use AWS Key Management Service (KMS) to manage encryption keys and integrate with services like Amazon S3 or Amazon DynamoDB for transparent encryption of selected fields.
4. **Apply data obfuscation techniques**. Use methods such as tokenization, data masking, or anonymization to protect sensitive information while preserving data utility for machine learning. Consider using services like AWS Glue DataBrew for data transformation and masking operations.
5. **Establish key rotation policies**. Implement regular rotation of encryption keys to minimize the impact of potential key compromises. Configure AWS KMS to automate key rotation according to your security policies and regulatory requirements.
6. **Secure ML model artifacts**. Verify that trained models and their associated metadata do not inadvertently expose sensitive information. Use Amazon SageMaker AI's security features to encrypt model artifacts and secure API endpoints that serve predictions.
7. **Implement access controls**. Restrict access to sensitive data and encryption keys using AWS Identity and Access Management (IAM) policies. Apply the principle of least privilege to verify that only authorized personnel can access protected information.

8. **Monitor and audit access patterns**. Implement continuous monitoring to detect unauthorized access attempts or unusual patterns that might indicate a security breach. Configure AWS CloudTrail and Amazon CloudWatch to track and alert on suspicious activities.
9. **Implement differential privacy techniques**. When working with AI models, consider implementing differential privacy techniques to add statistical noise to training data, protecting individual privacy while maintaining overall data utility.
10. **Establish mechanisms to stop model memorization**. Implement safeguards to block AI models from memorizing and potentially reproducing sensitive information from training data, especially when using large language models.

Resources

Related documents:

- Detect and process sensitive data
- Security Pillar - AWS Well-Architected Framework
- Data protection in Amazon SageMaker AI Unified Studio
- Data Privacy in Amazon SageMaker AI
- Data Encryption
- Introducing PII Data Identification and Handling Using AWS Glue DataBrew
- 7 ways to improve security of your machine learning workflows
- Secure deployment of Amazon SageMaker AI resources

Related videos:

- Privacy-preserving machine learning
- Data Protection Best Practices in Machine Learning

Data processing

Best practices

- MLSEC03-BP01 Provide least privilege access
- MLSEC03-BP02 Secure data and modeling environment
- MLSEC03-BP03 Protect sensitive data privacy
- MLSEC03-BP04 Enforce data lineage

- MLSEC03-BP05 Keep only relevant data

MLSEC03-BP01 Provide least privilege access

Protect resources across various phases of the ML lifecycle using the principle of least privilege. These resources include: data, algorithms, code, hyperparameters, trained model artifacts, and infrastructure. Provide dedicated network environments with dedicated resources and services to operate individual projects.

Desired outcome: You establish a secure machine learning environment by implementing the principle of least privilege for resources involved in your ML workflows. Your organization controls access to sensitive data, models, and infrastructure based on business roles, maintains clear separation between development, test, and production environments, and uses appropriate governance mechanisms to enforce security policies. This approach minimizes your attack surface and protects valuable ML assets.

Common anti-patterns:

- Granting excessive permissions to data scientists or developers beyond what they need.
- Using a single AWS account for ML workloads without proper separation.
- Not tagging sensitive data and resources for access control purposes.
- Failing to isolate ML environments based on data sensitivity requirements.
- Relying solely on manual access management without proper governance structures.

Benefits of establishing this best practice:

- Reduced risk of unauthorized access to sensitive data and ML assets.
- Clear segregation of duties based on business roles.
- Improves adherence to regulatory requirements for data protection.
- Simplified governance through standardized access patterns.
- Minimized potential impact of security breaches.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Protecting machine learning workflows requires a comprehensive security approach that applies the principle of least privilege to resources involved. By carefully controlling who has access to data, code, and infrastructure, you can reduce the risk of unauthorized access or data breaches.

When implementing least privilege for ML resources, consider the different phases of the ML lifecycle and the types of access needed by various roles. For example, data scientists might need read access to training data but not production systems, while ML engineers may need deployment permissions but limited access to raw data.

Setting up a multi-account architecture with AWS Organizations provides strong isolation between environments with different security requirements. This allows you to maintain separate development, testing, and production environments with appropriate controls for each.

Implementation steps

1. **Define role-based access control for ML teams**. Identify the distinct roles within your ML workflow, such as data scientists, ML engineers, and operations teams. Map these roles to specific access patterns required for their daily tasks. Use Amazon SageMaker AI Role Manager to quickly create persona-based IAM roles with preconfigured templates for common ML roles including data scientists, MLOps engineers, and business analysts. This reduces manual permissions management and facilitates least privilege access by default. Complement with AWS Identity and Access Management (IAM) for custom role-based policies. Implement regular access reviews to verify that permissions remain appropriate as responsibilities change.
2. **Implement account separation with AWS Organizations**. Create a multi-account architecture that segregates workloads between development, test, and production environments. Use AWS Organizations to centrally manage accounts and apply consistent policies. Establish tagging strategies to identify data sensitivity levels and resource ownership. Apply these tags to relevant resources like S3 buckets containing training data or SageMaker AI instances. Use Service Catalog to create pre-provisioned environments that align with security requirements.
3. **Organize ML workloads by access patterns**. Group ML workloads based on common access requirements and security profiles. Create organizational units (OUs) in AWS Organizations that reflect these groupings. Delegate specific access permissions to each group according to their needs. Apply service control policies (SCPs) to enforce security guardrails at the organizational unit level. Limit administrative access to infrastructure to designated administrators only.
4. **Isolate sensitive data environments**. Create dedicated, isolated environments for working with sensitive data. Implement network controls such as security groups and network ACLs to restrict

data flow between environments. Use Amazon VPC endpoints to provide private connectivity to AWS services without traversing the public internet. Configure AWS PrivateLink for secure access to SageMaker AI endpoints from within your VPC.

5. **Implement automated security controls**. Deploy AWS Config rules to continuously monitor resource configurations for adherence to security policies. Use Amazon GuardDuty for threat detection across your ML infrastructure. Implement AWS CloudTrail to log and monitor API calls related to ML resources. Consider using Amazon Macie to automatically discover and protect sensitive data stored in Amazon S3.
6. **Use secure ML development practices**. Implement code repositories with appropriate access controls for ML code and models. Use version control for artifacts including data, code, and model parameters. Apply the principle of least privilege to CI/CD pipelines that deploy ML models. Implement model governance processes that include security reviews before deployment to production.
7. **Deploy ML guardrails with service control policies**. Create SCPs that enforce requirements across your ML environments. Define policies that block storage of sensitive data in unencrypted formats. Restrict network egress from environments containing sensitive data. Limit which AWS Regions can be used for specific types of ML workloads based on requirements.
8. **Implement safeguards for AI systems**. For AI workloads, implement additional security controls to protect against input injection attacks. Implement built-in guardrails for responsible AI use. Apply input validation for user inputs to AI systems. Implement output filtering to avoid inadvertent disclosure of sensitive information. Consider using Amazon SageMaker AI with governance features to enforce compliance-aligned and responsible AI practices.

Resources

Related documents:

- Amazon SageMaker AI Role Manager
- Service Catalog
- Build a Secure Enterprise Machine Learning Platform on AWS
- Protecting data at rest
- Security best practices in IAM
- Building secure Amazon SageMaker AI access URLs with Service Catalog
- Setting up secure, well-governed machine learning environments on AWS
- ML security: Using Amazon SageMaker AI with AWS PrivateLink

Related videos:

- Architectural best practices for machine learning applications
- Secure and compliant machine learning for regulated industries
- Amazon SageMaker AI Model Development in a Highly Regulated Environment (SDD315)

Related examples:

- Build your own Anomaly Detection ML Pipeline
- AWS MLOps Framework
- Secure ML deployment architecture reference
- Secure Data Science Reference Architecture

MLSEC03-BP02 Secure data and modeling environment

Secure your machine learning data and development environments to protect valuable information assets throughout the ML lifecycle. By implementing proper security measures for storage, compute, and network resources, you can maintain data integrity and confidentiality while enabling data scientists to work effectively.

Desired outcome: You have a secure foundation for storing, processing, and utilizing data for machine learning workloads. Your data is encrypted at rest and in transit, with access tightly controlled through identity management, infrastructure isolation, and secure coding practices. Your development environments are protected from unauthorized access while providing the necessary tools for your ML practitioners.

Common anti-patterns:

- Storing unencrypted training data in publicly accessible storage.
- Using default security configurations for ML environments.
- Allowing unrestricted internet access from ML environments.
- Using hard-coded credentials in ML code and notebooks.
- Installing ML packages from untrusted sources without validation.
- Granting excessive permissions to development environments.

Benefits of establishing this best practice:

- Protection of sensitive training data from unauthorized access or exfiltration.

- Reduced risk of compromised ML models and systems.
- Improves adherence to regulatory requirements for data handling.
- Improved governance of ML development environments.
- Enhanced ability to detect and respond to security events.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Securing your ML environments requires a comprehensive approach addressing data storage, compute resources, network isolation, and access controls. The ML lifecycle involves multiple stages where data could be exposed if proper security measures aren't implemented. By establishing secure foundations for your ML infrastructure, you can protect valuable intellectual property while still enabling productivity.

Start by securing your data repositories with encryption and access controls. Then build secure compute environments for model development that maintain isolation through private networking. Implement proper credential management to avoid exposure of secrets. Finally, verify that your package management practices block the introduction of malicious code into your ML pipeline.

Modern ML workloads often involve large datasets and complex algorithms, making security even more critical as the impact of a breach could be substantial. By implementing the measures in this best practice, you create a secure foundation for your ML initiatives.

Implementation steps

1. **Build a secure analysis environment**. During the data preparation and feature engineering phases, leverage secure data exploration options on AWS. Use Amazon SageMaker AI Studio managed environments or Amazon EMR for data processing. Alternatively, use managed services like Amazon Athena and AWS Glue to explore data without moving it from your data lake. For smaller datasets, use Amazon SageMaker AI Studio to explore, visualize, and engineer features, then scale up your feature engineering using managed ETL services like Amazon EMR or AWS Glue.
2. **Create dedicated IAM and KMS resources**. Limit the scope and impact of credentials and keys by creating dedicated AWS IAM roles and AWS KMS keys for ML workloads. Create private Amazon S3 buckets with versioning enabled to protect your data and intellectual property. Implement a centralized data lake using AWS Lake Formation on Amazon S3. Secure your data

lake using a combination of services to encrypt data in transit and at rest. Monitor access with granular AWS IAM policies, S3 bucket policies, S3 Access Logs, Amazon CloudWatch, and AWS CloudTrail.

3. **Use Secrets Manager and Parameter Store to protect credentials**. Replace hard-coded secrets in your code with API calls to programmatically retrieve and decrypt secrets using AWS Secrets Manager. Use AWS Systems Manager Parameter Store to store application configuration variables such as AMI IDs or license keys. Grant permissions to your SageMaker AI IAM role to access these services from your ML environments.
4. **Automate managing configuration**. Use lifecycle configuration scripts to manage ML environments. These scripts run when environments are created or restarted, allowing you to install custom packages, preload datasets, and set up source code repositories. Lifecycle configurations can be reused across multiple environments and updated centrally. Use AWS CloudFormation infrastructure as code and Service Catalog to simplify configuration for end users while maintaining security standards.
5. **Create private, isolated, network environments**. Use Amazon Virtual Private Cloud (Amazon VPC) to limit connectivity to only essential services and users. Deploy Amazon SageMaker AI resources in a VPC to enable network-level controls and capture network activity in VPC Flow Logs. For distributed training workloads, use Amazon SageMaker AI HyperPod which provides managed, resilient clusters with built-in VPC integration and multi-AZ deployment for enhanced security and availability. This deployment model also enables secure queries to data sources within your VPC, such as Amazon RDS databases or Amazon Redshift data warehouses. Use IAM to restrict access to ML environment web UIs so they can only be accessed from within your VPC. Implement AWS PrivateLink to privately connect your SageMaker AI resources with supported AWS services, facilitating secure communication within the AWS network. Use AWS KMS to encrypt data on the Amazon EBS volumes attached to SageMaker AI resources.
6. **Restrict access**. ML development environments provide web-based access to the underlying compute resources, typically with elevated privileges. Restrict this access to remove the ability to assume root permissions while still allowing users to control their local environment. Implement least privilege access controls for ML resources.
7. **Secure ML algorithms**. Amazon SageMaker AI uses container technology to train and host algorithms and models. When creating custom containers, publish them to a private container registry hosted on Amazon Elastic Container Repository (Amazon ECR). Encrypt containers hosted on Amazon ECR at rest using AWS KMS. Regularly scan containers for vulnerabilities and implement a secure container update process.

8. **Enforce code best practices**. Use secure git repositories for storing code. Implement code reviews, automated security scanning, and version control for ML code. Integrate security checks into your ML CI/CD pipeline to detect potential security issues early in the development process.
9. **Implement a package mirror for consuming approved packages**. Evaluate license terms to determine appropriate ML packages for your business across the ML lifecycle phases. Common ML Python packages include Pandas, PyTorch, Keras, NumPy, and Scikit-learn. Build an automated validation mechanism to check packages for security issues. Only download packages from approved and private repos. Validate package contents before importing. SageMaker AI supports modifying package channel paths to a private repository. When appropriate, use an internal repository as a proxy for public repositories to minimize network traffic and reduce overhead.
10. **Implement model security monitoring**. Deploy continuous monitoring solutions to detect unauthorized access attempts, unusual data access patterns, and potential data exfiltration from your ML environments. Use Amazon CloudWatch, AWS Security Hub CSPM, and Amazon GuardDuty to create a comprehensive security monitoring solution for ML resources.
11. **Implement additional security controls for AI workloads**. For AI workloads, implement additional security controls around input validation and data leakage prevention. Implement Amazon SageMaker AI Model Monitor to detect drift in production AI systems. Consider using Amazon SageMaker AI Model Cards to document model security characteristics and limitations.

Resources

Related documents:

- Prerequisites for using SageMaker AI HyperPod
- Storage Best Practices for Data and Analytics Applications
- Configure security in Amazon SageMaker AI
- Protecting compute
- Protecting data in transit
- 7 ways to improve security of your machine learning workflows
- Building secure machine learning environments with Amazon SageMaker AI
- Setting up secure, well-governed machine learning environments on AWS
- Private package installation in Amazon SageMaker AI running internet-free mode
- Secure Deployment of Amazon SageMaker AI resources
- Apply fine-grained data access controls with AWS Lake Formation and Amazon EMR from Amazon SageMaker AI Studio

Related videos:

- Security for AI/ML Models in AWS
- Security best practices the AWS Well-Architected way

Related examples:

- Secure Data Science Reference Architecture
- Amazon SageMaker AI Secure MLOps

MLSEC03-BP03 Protect sensitive data privacy

Protect sensitive data used in training against unintended disclosure by implementing appropriate identification, classification, and handling strategies. This practice improves data privacy while maintaining model utility through techniques such as data removal, masking, tokenization, and principal component analysis (PCA).

Desired outcome: You establish effective protocols to identify, classify, and protect sensitive data throughout your machine learning workflows. Your sensitive data is appropriately secured with encryption, access controls, and data minimization techniques. Your organization maintains clear documentation of governance practices for consistent application across projects.

Common anti-patterns:

- Failing to identify sensitive data before using it for model training.
- Using raw PII or other sensitive data when anonymized data would suffice.
- Not implementing proper encryption for sensitive training data.
- Assuming cloud services automatically protect sensitive data without proper configuration.
- Neglecting to document data handling processes for future reference.

Benefits of establishing this best practice:

- Reduced risk of data breaches and privacy violations.
- Improves adherence to data protection regulations.
- Increased trust from customers and stakeholders.
- Improved ability to use sensitive data for legitimate machine learning purposes.
- Better governance through documented protocols.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Protecting sensitive data privacy in machine learning workflows requires a systematic approach that begins with data identification and classification. You need to understand what data you have and its sensitivity levels before determining appropriate protection mechanisms. Different types of sensitive data may require different handling strategies—some might need complete removal, while others can be effectively masked or tokenized.

When working with sensitive data in ML workflows, you should adopt a defense-in-depth approach. This means implementing multiple layers of protection, including access controls, encryption, data minimization techniques, and monitoring systems. For example, you might use AWS Key Management Service (KMS) to encrypt your training data, implement role-based access controls, and use Amazon Macie to continuously monitor for sensitive data exposure.

Privacy-preserving machine learning techniques are increasingly important as models become more sophisticated. Techniques like differential privacy, federated learning, and secure multi-party computation can allow you to train effective models while minimizing exposure of sensitive data. These approaches maintain privacy while still extracting valuable insights from your data.

Implementation steps

1. **Implement automated data discovery and classification**. Use automated sensitive data discovery in Amazon Macie to gain continuous, cost-efficient, organization-wide visibility into where sensitive data resides across your Amazon S3 environment. Macie automatically inspects your S3 buckets for sensitive data such as personally identifiable information (PII), financial data, and AWS credentials, then builds and maintains an interactive data map of sensitive data locations and provides sensitivity scores for each bucket.
2. **Apply resource tagging for sensitive data tracking**. Tag resources and models that contain or are derived from sensitive elements to quickly differentiate between resources requiring protection and those that do not. Use AWS resource tagging to systematically identify and manage resources containing sensitive data throughout their lifecycle.
3. **Implement comprehensive encryption strategies**. Encrypt sensitive data using services such as AWS Key Management Service (KMS), the AWS Encryption SDK, or client-side encryption. Apply encryption consistently across data at rest and in transit, with appropriate key management practices.

4. **Implement data minimization techniques**. Evaluate and identify data for anonymization or de-identification to reduce sensitivity. Use techniques such as masking, tokenization, or principal component analysis to reduce the risk associated with using sensitive data for training. Consider using Amazon SageMaker AI Feature Store with appropriate transformation techniques to create privacy-preserving feature representations.
5. **Establish governance documentation and processes**. Create comprehensive documentation of your sensitive data handling practices, including classification schemes, protection mechanisms, access control policies, and incident response procedures. Regularly review and update these documents to reflect changes in regulations, technologies, and organizational practices.
6. **Implement differential privacy techniques**. Apply differential privacy methods to add controlled noise to your data or models to block the extraction of individual data points while maintaining overall statistical validity. AWS Clean Rooms assist organizations with collaborating on sensitive data while maintaining privacy and adherence to regulations.
7. **Perform regular privacy impact assessments**. Conduct systematic evaluations of how your ML workflows collect, use, and protect sensitive data. Use the results to identify areas for improvement in your privacy protection mechanisms and adhere to relevant regulations.
8. **Implement safeguards for large language models**. When using large language models, implement safeguards to block memorization and exposure of sensitive training data. Use Amazon SageMaker AI JumpStart with appropriate privacy-preserving configurations and implement proper data filtering and anonymization techniques during model training and fine-tuning.

Resources

Related documents:

- Running sensitive data discovery jobs in Amazon Macie
- Categorizing your storage using tags
- AWS Key Management Service
- Using the AWS Encryption SDK with AWS KMS
- 7 ways to improve security of your machine learning workflows
- Use Macie to discover sensitive data as part of automated data pipelines
- Building a Serverless Tokenization Solution to Mask Sensitive Data

Related videos:

- Security for AI/ML Models in AWS
- Security best practices the AWS Well-Architected way

Related examples:

- Secure Data Science Reference Architecture
- Amazon SageMaker AI Secure MLOps

MLSEC03-BP04 Enforce data lineage

Data lineage tracking allows you to monitor and track data origins and transformations over time, enabling better visibility into your machine learning workflows. By enforcing data lineage, you can trace the root cause of data processing errors and and protect the integrity of your ML models.

Desired outcome: You can trace a data element back to its source, verify the transformations it underwent, and verify data integrity throughout the ML lifecycle. You have visibility into your entire ML workflow from data preparation to model deployment, enabling you to reproduce workflows, establish model governance standards, and demonstrate audit adherence.

Common anti-patterns:

- Treating data lineage as an afterthought rather than a core requirement.
- Failing to maintain records of data transformations during preprocessing.
- Not implementing integrity checks for detecting data manipulation or corruption.
- Neglecting to document code and infrastructure changes that affect the ML pipeline.
- Relying on manual tracking methods that are prone to errors and inconsistencies.

Benefits of establishing this best practice:

- Improved troubleshooting through the ability to trace issues back to their source.
- Improves adherence to regulatory requirements through comprehensive audit trails.
- Greater confidence in model outputs by understanding the provenance of training data.
- Faster iteration cycles by being able to reproduce workflows efficiently.
- Better governance and risk management across ML operations.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Data lineage is a critical component of responsible ML operations. By tracking the journey of your data from its source through various transformations to model deployment, you create accountability and transparency in your ML systems. Enforcing data lineage involves implementing mechanisms to record metadata about data origins, transformations, and access controls throughout the ML lifecycle.

Amazon SageMaker AI provides built-in capabilities to track and maintain data lineage through its MLflow tracking capabilities. This system allows you to record the relationships between various ML artifacts such as datasets, algorithms, hyperparameters, and model artifacts. By utilizing these tracking capabilities, you can establish a clear audit trail that assists with reproducibility, governance, and troubleshooting.

Proper data lineage implementation also requires strict access controls to block unauthorized data manipulation. Your tracking system should record who accessed the data, what changes were made, and when those changes occurred. Additionally, implement integrity checks against your training data to detect unexpected deviations caused by data corruption or malicious manipulation.

Implementation steps

1. **Set up Amazon SageMaker AI MLflow Tracking**. Enable tracking capabilities in your SageMaker AI environment to automatically capture metadata about your ML workflows. Configure SageMaker AI to track artifacts, associations, and context information using Amazon SageMaker AI MLflow. MLflow in SageMaker AI allows you to create, manage, analyze, and compare experiments, providing comprehensive tracking of training runs, model versions, and associated metadata.
2. **Implement automated metadata collection**. Configure your ML pipelines to automatically record metadata at each stage of processing. Use SageMaker AI Processing jobs to track data transformations and record preprocessing steps. Apply SageMaker AI Pipeline steps to document the flow of data from one stage to another, creating a complete record of the data journey.
3. **Establish data access controls**. Implement strict access controls to protect data integrity. Use AWS Identity and Access Management (IAM) roles and policies to restrict access to specific datasets and models. Configure Amazon SageMaker AI Model Monitor to detect unauthorized access or changes to your data.
4. **Create integrity verification mechanisms**. Implement data validation steps in your pipeline to detect anomalies or unexpected changes. Use checksums, statistical analysis, or machine

learning-based anomaly detection to identify potential data corruption. Store integrity verification results as part of your lineage tracking records.

5. **Document code and infrastructure changes**. Track changes to your code repositories and infrastructure configurations that affect the ML workflow. Use version control systems like Git integrated with AWS CodeCommit to maintain a history of code changes, and AWS CloudFormation or AWS CDK to version your infrastructure as code.
6. **Implement end-to-end traceability**. Verify that your lineage tracking system can trace model predictions back to the original data sources used for training. Use SageMaker AI MLflow Model Registry to catalog your models and associate them with their training data lineage. This enables you to understand exactly which data influenced specific model behaviors.
7. **Establish audit and compliance-aligned reporting**. Create automated reports that demonstrate data lineage for compliance-aligned purposes. Use Quick to visualize data lineage graphs and Amazon Athena to query lineage metadata for audit reports. Regularly review these reports to improve your adherence to your governance requirements.
8. **Implement foundation model tracking**. For foundation model workflows, track not only the data but also the foundation models used, their versions, and fine-tuning parameters. Use Amazon SageMaker AI Model Cards to document model characteristics and Amazon SageMaker AI Model Dashboard to monitor model performance. Implement comprehensive traceability features to document model provenance and usage.
9. **Track model input variations**. Maintain a record of input variations used with models, as these influence model outputs. Use Amazon SageMaker AI MLflow tracking server with enhanced MLflow 3.0 capabilities to track different input variations and their effectiveness, treating inputs as critical components of your data lineage system. The managed MLflow service provides robust experiment management at scale for ML projects with comprehensive tracking of training runs, model versions, and associated metadata.

Resources

Related documents:

- Accelerate generative AI development using managed MLflow on Amazon SageMaker AI
- SageMaker AI MLflow Tracking Server
- Amazon SageMaker AI Feature Store
- Amazon SageMaker AI Model Cards
- Accelerating generative AI development with fully managed MLflow 3.0 on Amazon SageMaker AI

- Building, automating, managing, and scaling ML workflows using Amazon SageMaker AI Pipelines

Related videos:

- How To Efficiently Manage ML experiments using Amazon SageMaker AI ML Flow

MLSEC03-BP05 Keep only relevant data

Reduce data exposure risks by preserving only use-case relevant data across computing environments. Implementing data lifecycle management and privacy-preserving techniques maintains data security while enabling effective machine learning workflows.

Desired outcome: You maintain a streamlined dataset across development, staging, and production environments that contains only the data elements needed for your machine learning use cases. You have implemented automated data lifecycle management processes that properly identify data, redact it when necessary, and remove it when no longer needed. This approach reduces your security risk exposure while maintaining data usability for ML operations.

Common anti-patterns:

- Keeping collected data indefinitely in case it might be useful later.
- Failing to implement data redaction for personally identifiable information (PII) in ML datasets.
- Using production data with sensitive information in development environments.
- Not establishing clear timelines for data retention and removal.
- Ignoring privacy regulations when designing ML workflows.

Benefits of establishing this best practice:

- Reduced risk of data breaches and privacy violations.
- Lower storage and computational costs from processing only necessary data.
- Improved adherence to data privacy regulations.
- Enhanced ML model performance through focus on relevant features.
- Streamlined data management processes across environments.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Managing data exposure is crucial for machine learning security. The more data you collect and store, the greater your attack surface and potential for data breaches. By focusing on data minimization principles, you can reduce these risks while still achieving your ML objectives.

Your data lifecycle management strategy should begin with a thorough assessment of what data is truly needed for your ML use cases. This requires close collaboration between data scientists, security professionals, and business stakeholders to identify essential features and acceptable levels of data granularity. Once identified, implement mechanisms to maintain only the necessary data elements across environments.

When working with potentially sensitive information, apply privacy-preserving techniques like anonymization, pseudonymization, or redaction of PII. AWS services like Amazon Comprehend and Amazon Macie can identify sensitive data automatically, while Amazon Transcribe offers automatic redaction capabilities. For more advanced scenarios, consider techniques like differential privacy or federated learning that allow you to derive insights from sensitive data without exposing the raw information.

Regular data audits and automated cleanup processes are essential components of an effective data lifecycle management strategy. By implementing automated policies for data retention and deletion, you can verify that data doesn't linger unnecessarily in your systems after its useful life has ended.

Implementation steps

1. **Assess data requirements**. Begin by thoroughly analyzing your ML use case to determine exactly which data elements are required for model training, validation, and inference. Document the minimum data requirements for each stage of your ML pipeline and justify the need for each attribute. Consider using techniques like feature importance analysis to identify which data elements contribute most to model performance.
2. **Develop a comprehensive data lifecycle plan**. Create a documented plan that defines how data will flow through your ML pipeline, including data collection, processing, storage, usage, and eventual deletion. Identify usage patterns and requirements for debugging and operational tasks. Specify retention periods based on business needs, regulatory requirements, and the purpose of the data.
3. **Implement data minimization techniques**. Design your data collection and preprocessing pipelines to capture only the necessary data attributes identified in your assessment. Use

AWS Glue or similar ETL services to filter out unnecessary fields before storage. Consider implementing record-level filtering in addition to column-level filtering.

4. **Set up automated PII detection and redaction**. Deploy solutions to automatically identify and redact sensitive information. Use Amazon Comprehend for detecting PII in text data and Amazon Rekognition for identifying sensitive elements in images. Implement Amazon Transcribe's automatic redaction feature for audio transcriptions.
5. **Establish data governance controls**. Implement access controls and encryption mechanisms using AWS Identity and Access Management (IAM) and AWS Key Management Service (KMS). Use Amazon Macie to automatically discover, classify, and protect sensitive data in AWS. Apply data classification tags to facilitate appropriate handling of different data types.
6. **Configure automated data lifecycle policies**. Set up S3 Lifecycle configurations to automatically transition or expire data based on your retention policies. Implement similar mechanisms for other storage systems used in your ML pipeline. Create automated jobs to periodically review and remove stale data from environments.
7. **Implement privacy-preserving ML techniques**. Where possible, use privacy-enhancing technologies like differential privacy, federated learning, or encrypted computation. Consider using AWS Lake Formation to centrally define and enforce fine-grained access controls. For sensitive use cases, explore options for machine learning on encrypted data.
8. **Monitor and audit data usage**. Set up logging and monitoring using AWS CloudTrail and Amazon CloudWatch to track data access patterns. Periodically audit data usage against documented requirements to identify and avoid unnecessary data collection. Use Amazon Athena with user-defined functions for analyzing and redacting sensitive information in logs and audit trails.
9. **Implement responsible data practices for AI models**. When using AI models, be especially careful with training data to block memorization of sensitive information. Utilize Amazon SageMaker AI's feature store for centralized feature management with built-in security controls. Consider data poisoning risks and implement appropriate data validation before model training.

Resources

Related documents:

- Reference Guide: Extract More Value from your Data
- Data Privacy Center
- Building a data analytics practice across the data lifecycle
- Detecting and redacting PII using Amazon Comprehend

- Now available in Amazon Transcribe: Automatic Redaction of Personally Identifiable Information
- Redacting sensitive information with user-defined functions in Amazon Athena

Related videos:

- Privacy-preserving machine learning
- Best practices for Amazon S3

Related examples:

- Field Notes: Redacting Personal Data from Connected Cars Using Amazon Rekognition
- How to Create a Modern CPG Data Architecture with Data Mesh
- Building a secure enterprise machine learning platform on AWS

Model development

Best practices

- MLSEC04-BP01 Secure governed ML environment
- MLSEC04-BP02 Secure inter-node cluster communications
- MLSEC04-BP03 Protect against data poisoning threats

MLSEC04-BP01 Secure governed ML environment

Creating a secure and governed ML environment allows you to protect valuable data and models while enabling teams to innovate efficiently. By implementing proper guardrails, monitoring, and security practices, you maintain control while providing the flexibility ML practitioners need to deliver business value.

Desired outcome: You establish a secure ML operational environment using AWS managed services that incorporates best practices for security, governance, and monitoring. You create development environments that allow data scientists to explore data safely while maintaining organizational security standards. Your ML environments are centrally managed with proper access controls, yet offer self-service capabilities to improve productivity. This balance between security and flexibility enables your organization to innovate while protecting sensitive assets.

Common anti-patterns:

- Using a single shared account for ML workloads regardless of sensitivity or access requirements.
- Allowing unrestricted access to ML infrastructure and production environments.
- Implementing manual provisioning processes that create bottlenecks for data scientists.
- Neglecting to isolate environments containing sensitive data.
- Failing to implement continuous monitoring and detection controls for ML operations.

Benefits of establishing this best practice:

- Reduced security risks through proper isolation and access controls.
- Improved governance with enforced security guardrails.
- Enhanced productivity through self-service capabilities.
- Improves adherence to regulatory requirements.
- Simplified management of ML environments.
- Faster time-to-market for ML initiatives.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Securing your ML environment requires thoughtful architecture that balances security with productivity. You need to consider how different teams interact with ML resources and implement controls appropriate to their roles and the sensitivity of data being processed. AWS provides managed services like Amazon SageMaker AI that can be configured with security best practices in mind.

Begin by understanding your organization's access patterns and data sensitivity levels. This knowledge assists you when determining how to structure your AWS accounts and implementing appropriate security controls. For example, you might separate development, testing, and production environments across different accounts with increasing security restrictions. This multi-account strategy allows you to implement tailored security controls for each environment while maintaining proper isolation.

Once you've established your account structure, implement preventive guardrails using AWS Organizations and service control policies (SCPs) to enforce security boundaries. Detective controls using services like AWS Config and Amazon GuardDuty provide continuous monitoring to identify potential security issues. By combining preventive and detective controls, you create defense-in-depth protection for your ML environments.

For environments handling sensitive data, implement additional security measures like network isolation, encryption, and fine-grained access controls. Amazon SageMaker AI can be deployed within a VPC to limit network access, while AWS KMS provides robust encryption capabilities for data at rest and in transit. These measures protect sensitive information throughout the ML lifecycle.

Implementation steps

1. **Break out ML workloads by organizational unit access patterns**. Create a multi-account strategy that aligns with your organization's structure and security requirements. For example, create separate accounts for data science development, model training, and production model deployment. This separation allows you to implement role-based access control (RBAC) with appropriate permissions for each team. Use Amazon SageMaker AI Role Manager to quickly define persona-based IAM roles for different user types (data scientists, MLOps engineers, business analysts) with preconfigured templates that provide least privilege access. Use AWS Organizations to manage your multi-account environment efficiently.
2. **Use guardrails and service control policies (SCPs) to enforce best practices**. Implement SCPs through AWS Organizations to establish preventive guardrails that restrict actions across accounts. For example, create policies that block the disabling of security services, limit the AWS regions that can be used, or restrict the creation of public resources. Complement SCPs with AWS Config rules to detect non-compatible resources and automatically remediate issues. Limit infrastructure management access to administrators while allowing data scientists to focus on model development.
3. **Verify that sensitive data has access through restricted, isolated environments**. Implement Amazon SageMaker AI within a private VPC to control network traffic to and from your ML environment. Configure security groups and network ACLs to restrict access to authorized sources. Use AWS PrivateLink to access AWS services without traversing the public internet. Enable encryption for sensitive data using AWS Key Management Service (KMS) for both data at rest and in transit. Review service dependencies to verify that they meet your security requirements.
4. **Secure ML algorithm implementation using a restricted development environment**. Deploy Amazon SageMaker AI Studio with appropriate security controls to provide data scientists with a secure development environment. Implement AWS Identity and Access Management (IAM) roles with least privilege permissions for each development environment. Use Amazon SageMaker AI Domain configurations to manage user access to resources. Scan container images for vulnerabilities before deploying them for model training or hosting using Amazon ECR image scanning.

5. **Implement centralized management and monitoring**. Use AWS CloudTrail to track API activity across your ML environments. Deploy Amazon CloudWatch for operational monitoring of your ML resources. Implement Amazon GuardDuty to detect suspicious activity. Centralize logs in a dedicated security account for comprehensive visibility across your ML environments. Create automated alerts for security-related events that require investigation.
6. **Enable self-service provisioning with guardrails**. Implement Service Catalog to provide pre-approved, secure templates for ML resources like SageMaker AI environments. Configure lifecycle policies to automatically shut down idle resources and reduce costs. Use AWS CloudFormation or AWS CDK to define infrastructure as code with security best practices built in. This allows data scientists to provision resources quickly while maintaining adherence to organizational standards.
7. **Secure model artifacts and ML pipelines**. Implement version control for models and code using Amazon SageMaker AI MLflow Model Registry. Configure Amazon SageMaker AI Pipelines with appropriate access controls to automate the ML lifecycle. Use AWS CodePipeline and AWS CodeBuild to implement CI/CD for ML applications with security checks built into the deployment process.
8. **Implement foundation model security controls**. When using large language models (LLMs) or other foundation models, implement guardrails to block the generation of harmful content. Implement content filtering to verify responsible AI usage. For enterprise governance of foundation models, implement SageMaker AI JumpStart Private Model Hub to create curated repositories of approved models with centralized access controls and version management. Use SageMaker AI Catalog as a central metadata hub for secure sharing and governed access to ML assets across business units. Implement Amazon SageMaker AI Model Cards to document model limitations, ethical considerations, and intended uses. Monitor model outputs for drift and bias using Amazon SageMaker AI Model Monitor.

Resources

Related documents:

- Amazon SageMaker AI Role Manager
- Private curated hubs for foundation model access control in JumpStart
- Admin guide for private model hubs in Amazon SageMaker AI JumpStart
- Configure security in Amazon SageMaker AI
- Build a secure enterprise machine learning platform on AWS
- Security Pillar - AWS Well-Architected Framework

- Model governance to manage permissions and track model performance
- Setting up secure, well-governed machine learning environments on AWS
- Securing Amazon SageMaker AI Studio connectivity using a private VPC
- Enable self-service, secured data science using Amazon SageMaker AI and Service Catalog
- Accelerating Machine Learning Development with Data Science as a Service from Change Healthcare

Related videos:

- Architectural best practices for machine learning applications
- Secure and compliant machine learning for regulated industries
- Amazon SageMaker AI Model Development in a Highly Regulated Environment (SDD315)

MLSEC04-BP02 Secure inter-node cluster communications

Machine learning frameworks require secure communications between computational nodes to maintain data integrity and protect sensitive information during model training. By implementing encryption for inter-node communications, you safeguard coefficient exchanges and protect synchronized information across distributed environments.

Desired outcome: You establish encrypted communication channels between computational nodes in your machine learning clusters, protecting sensitive model data, parameters, and training information as it traverses networks. This improves data integrity and confidentiality during distributed training operations while maintaining the performance requirements of your machine learning workloads.

Common anti-patterns:

- Assuming internal network communications are inherently secure and don't require encryption.
- Implementing encryption only for external communications but neglecting inter-node traffic.
- Using outdated or weak encryption protocols for performance reasons.
- Neglecting to rotate encryption certificates and credentials regularly.

Benefits of establishing this best practice:

- Protection of proprietary algorithms and model parameters during training.
- Prevention of data leakage and unauthorized access to training data.

- Improves adherence to data protection regulations and security requirements.
- Consistent security posture across your ML infrastructure.

Level of risk exposed if this best practice is not established: High

Implementation guidance

For machine learning frameworks like TensorFlow that rely on distributed computing, secure inter-node communication is essential to protect the integrity and confidentiality of the training process. During distributed training, nodes exchange critical information like model coefficients, gradients, and parameter updates. This information contains valuable intellectual property about your models and potentially sensitive insights derived from your training data.

When implementing distributed machine learning workloads, encrypt that data transmitted between computational nodes using industry-standard protocols. This is particularly important when your infrastructure spans across different networks, availability zones, or even Regions. Encryption in transit stops unauthorized parties from intercepting or tampering with model data as it moves between nodes.

AWS services like Amazon SageMaker AI and Amazon EMR provide built-in capabilities to secure inter-node communications, making it more straightforward to implement this best practice without extensive custom configuration.

Implementation steps

1. **Enable inter-node encryption in Amazon SageMaker AI**. Amazon SageMaker AI provides automatic encryption for inter-container communication during training jobs. When configuring your training job, enable encryption to verify that data passed between containers traverses over an encrypted tunnel. For large-scale distributed training, use Amazon SageMaker AI HyperPod which provides managed, resilient clusters with built-in security features including VPC integration, automatic health checks, and secure node-to-node communication for foundation model training. This protects your model parameters and gradients during the training process without requiring additional configuration.
2. **Configure TLS for distributed TensorFlow workloads**. For TensorFlow-based distributed training, implement Transport Layer Security (TLS) to secure communications between worker nodes. TensorFlow supports TLS configuration through environment variables and configuration parameters. Use properly signed certificates and configure both client and server-side authentication for maximum security.

3. **Enable encryption in transit in Amazon EMR**. When using Amazon EMR for machine learning workloads, implement security configurations that enable encryption in transit. Amazon EMR makes it simple to create security configurations that specify the use of Transport Layer Security (TLS) certificates for encrypting data in transit between cluster nodes. This protects data whether it's stored locally on the cluster or in Amazon S3.
4. **Implement secure key management**. Use AWS Key Management Service (KMS) to manage the encryption keys used for securing inter-node communications. This provides centralized control, auditing, and automatic key rotation, enhancing your security posture while simplifying key management operations.
5. **Configure secure cluster authentication**. Implement strong authentication mechanisms to verify that only authorized nodes can join your cluster and participate in the distributed training process. Use certificate-based authentication where possible and implement node identity verification as part of your security configuration.
6. **Regularly rotate security credentials**. Establish a process for regularly rotating TLS certificates, encryption keys, and other security credentials used in your distributed training environment. This limits the potential impact of compromised credentials and aligns with security best practices.
7. **Monitor encrypted communications**. Implement logging and monitoring for your encrypted communications channels to detect potential security issues. Configure alerts for unusual traffic patterns or authentication failures that might indicate attempted security breaches.
8. **Secure foundation model communication**. When using distributed training for large language models or other foundation models, encrypt parameter server communications, as these contain valuable intellectual property. For AI workloads on Amazon SageMaker AI, enable inter-container encryption to protect model weights and gradients during the training process.

Resources

Related documents:

- Amazon SageMaker AI HyperPod Prerequisites
- Protect Communications Between ML Compute Instances in a Distributed Training Job
- Encryption options for Amazon EMR
- Configure security in Amazon SageMaker AI
- Security Pillar - AWS Well-Architected Framework
- Encrypt data in transit using a TLS custom certificate provider with Amazon EMR
- Building secure machine learning environments with Amazon SageMaker AI

- Amazon SageMaker AI Studio Admin Best Practices

Related videos:

- Architectural best practices for machine learning applications
- Secure and compliant machine learning for regulated industries

Related examples:

- Amazon SageMaker AI secure distributed training examples

MLSEC04-BP03 Protect against data poisoning threats

Protect your machine learning models and data by implementing security measures against data poisoning attacks, which can compromise model performance and accuracy. Data poisoning occurs through data injection (adding corrupt training data) or data manipulation (changing existing data like labels), resulting in inaccurate and weakened predictive capabilities. By identifying and addressing corrupt data using security methods and anomaly detection algorithms, you can maintain data integrity and protect against threats including ransomware and malicious code in third-party packages.

Desired outcome: You have implemented robust protection mechanisms for your machine learning training data and models. These protections include data validation procedures, monitoring for data drift, version control for both data and models, and rollback capabilities. Your ML systems can detect potential poisoning attempts and maintain model performance integrity through security best practices that protect data throughout its lifecycle.

Common anti-patterns:

- Collecting training data from untrusted or unverified sources without validation.
- Neglecting to monitor data distributions for unexpected shifts.
- Deploying updated models without thorough testing against baseline performance.
- Failing to implement version control for both training data and models.
- Not having a rollback strategy for compromised models.

Benefits of establishing this best practice:

- Improved model reliability and accuracy through clean, trusted data.

- Early detection of potential security breaches targeting training data.
- Reduced risk of deploying compromised models to production.
- Ability to quickly recover from poisoning incidents through rollback mechanisms.
- Enhanced overall ML system security and resilience.

Risk level for not implementing this practice: High

Implementation guidance

Data poisoning represents a security threat to machine learning systems. When malicious actors manipulate training data, they can compromise model integrity and cause downstream impacts on decisions or predictions made by those models. You need to implement comprehensive protections throughout your ML pipeline, from data collection to model deployment and monitoring.

Start by establishing strict controls over data sources and implementing validation procedures to detect anomalies before training. During model development, implement monitoring for data drift that could indicate poisoning attempts. Before deployment, thoroughly compare new models against previous versions to identify unexpected behavior changes. Finally, maintain versioned copies of both training data and models to enable rapid recovery from compromise.

By combining these defensive approaches, you create multiple layers of protection that make your ML systems resilient against data poisoning attempts.

Implementation steps

1. **Use only trusted data sources for training data**. Verify the provenance of data used for training and implement audit controls that allow you to track changes to training data. This includes recording who made changes, what changes were made, and when they occurred. Before using data for training, validate its quality to identify potential outliers and incorrectly labeled samples that could indicate poisoning attempts.
2. **Look for underlying shifts in the patterns and distributions in training data**. Implement continuous monitoring for data drift to detect unexpected changes in data distributions. Use tools like Amazon SageMaker AI Model Monitor to track these changes automatically. Deviations from established patterns can serve as early warning signs of unauthorized access or manipulation targeting training data.
3. **Identify model updates that negatively impact the results before moving them to production**. Compare newly trained models against previous versions using consistent test datasets. Look for unexpected performance changes, especially degradations in specific areas

that weren't present in earlier model iterations. Use Amazon SageMaker AI MLflow Model Registry to track model versions and their performance metrics.

4. **Have a rollback plan**. Implement versioning for both training data and models to enable quick recovery from compromised states. Use Amazon SageMaker AI Feature Store to maintain secure, versioned features for your ML models. The Feature Store provides a centralized repository for features with built-in security controls. Configure Amazon SageMaker AI MLflow Model Registry to support rollback capabilities so you can quickly revert to a known good model version if issues are detected with a newly deployed model.
5. **Use low-entropy classification cases**. Establish performance thresholds and monitor for unexpected classification patterns. Define boundaries for acceptable model behavior and create alerts when outputs deviate from expected patterns. This can identify subtle poisoning attempts that might otherwise go undetected through conventional testing.
6. **Implement end-to-end encryption for ML data**. Secure your training data, feature sets, and models using encryption both at rest and in transit. Use AWS Key Management Service (KMS) to manage encryption keys and apply them consistently across your ML pipeline. Encryption protects against unauthorized access that could lead to data poisoning.
7. **Regularly scan for vulnerabilities in ML dependencies**. Use tools like Amazon Inspector to detect vulnerabilities in the software packages and dependencies used in your ML environment. Data poisoning can occur through compromised third-party libraries, so regular scanning can identify potential entry points for bad actors.
8. **Implement input validation for AI systems**. For AI models, validate inputs for potential poisoning attempts. Implement filtering and sanitization of inputs to block adversarial inputs that could manipulate model behavior or extract sensitive information.

Resources

Related documents:

- Bias drift for models in production
- Accelerate generative AI development using managed MLflow on Amazon SageMaker AI
- Create, store, and share features with Feature Store
- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- Automated monitoring of your machine learning models with Amazon SageMaker AI Model Monitor and sending predictions to human review workflows using Amazon A2I
- Amazon SageMaker AI Model Monitor– Fully Managed Automatic Monitoring for Your Machine Learning Models

- 7 ways to improve security of your machine learning workflows
- Building secure machine learning environments with Amazon SageMaker AI

Related videos:

- Detect machine learning (ML) model drift in production
- Inawisdom: Machine Learning and Automated Model Retraining with SageMaker AI

Related examples:

- Amazon SageMaker AI Model Monitor Examples
- Amazon SageMaker AI Feature Store Examples

Deployment

Best practices

- MLSEC05-BP01 Protect against adversarial and malicious activities

MLSEC05-BP01 Protect against adversarial and malicious activities

Machine learning systems must be robust against adversarial inputs designed to manipulate their behavior. Implementing protection mechanisms both inside and outside of your deployed models can detect malicious inputs that could lead to incorrect predictions, allowing you to automatically identify unauthorized changes, repair compromised inputs, and validate data before it's used for further training.

Desired outcome: You can detect, mitigate, and protect your ML models from adversarial exploits that attempt to manipulate inputs, preserving model integrity and providing reliable predictions. Your ML systems incorporate robust validation processes, ensemble approaches, and monitoring capabilities that maintain consistent performance even when faced with deliberately perturbed or malicious inputs.

Common anti-patterns:

- Implementing ML models without considering potential adversarial threats.
- Focusing solely on model performance metrics without evaluating robustness.
- Using single model architectures that are vulnerable to targeted threats.

- Retraining models with unvalidated inputs that may contain adversarial examples.
- Exposing ML models through unsecured endpoints without monitoring capabilities.

Benefits of establishing this best practice:

- Improved model robustness against input manipulation attempts.
- Enhanced detection of potential security threats targeting ML systems.
- Greater reliability in model predictions even under adversarial conditions.
- Protection against data poisoning during model retraining.
- Reduced vulnerability to model extraction and inference threats.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Adversarial threats against machine learning models involve deliberately crafting inputs to cause incorrect predictions or undesirable behaviors. These threats can range from subtle perturbations that are imperceptible to humans but cause model errors, to more sophisticated techniques that exploit model vulnerabilities. Building protection against adversarial activities requires a multi-layered approach combining robust model design, comprehensive monitoring, and secure deployment strategies.

When implementing defenses, you need to understand the specific vulnerabilities in your models and the potential impact of adversarial threats on your business outcomes. This requires conducting thorough evaluations of model behavior under different threats scenarios and implementing appropriate countermeasures based on your risk profile. Both pre-deployment testing and continuous monitoring during production are essential components of a comprehensive protection strategy.

Adversarial robustness should be considered from the beginning of your ML development process rather than as an afterthought. By incorporating adversarial training, ensemble methods, and input validation techniques during model development, you can create systems that are inherently more resistant to threats. Additionally, implementing proper access controls, monitoring systems, and incident response procedures can establish a secure operational environment for your ML models.

Implementation steps

1. **Evaluate the robustness of your algorithm**. Conduct sensitivity analysis to understand how your model responds to perturbed inputs. Test your model with increasingly modified

data points to identify decision boundaries that might be vulnerable to manipulation. Use adversarial testing frameworks to simulate potential threats and measure their impact on model performance. Document the types of perturbations that cause incorrect predictions to inform your defense strategy.

2. **Build for robustness from the start**. Select diverse features during model design to improve resilience against outliers and adversarial examples. Implement ensemble methods by combining multiple models with different architectures or training approaches to increase decision diversity. Consider techniques like adversarial training where you intentionally incorporate adversarial examples into your training data to make your models more resistant to threats.
3. **Identify repeats and suspicious patterns**. Deploy Amazon SageMaker AI Model Monitor to continuously analyze inference data and detect unusual patterns such as repeated similar inputs that may indicate probing threats. Set up alerts for anomalous input distributions that differ from training data. Monitor for evidence of model brute-forcing, where bad actors systematically vary limited sets of input features to determine decision boundaries and derive feature importance.
4. **Implement experiment and model tracking**. Maintain comprehensive records of data provenance and model versions to trace model skew back to potentially compromised data sources. Before retraining models with new data, implement validation processes to identify and remove adversarial examples. Use Amazon SageMaker AI MLflow to document relationships between datasets, algorithms, and model artifacts throughout the ML lifecycle.
5. **Use secure inference API endpoints**. Host models behind properly secured API endpoints that implement authentication, authorization, and input validation. Configure Amazon SageMaker AI endpoints with appropriate security controls including VPC isolation, AWS IAM roles with least privilege, and encryption for data in transit and at rest. Implement rate limiting and request validation to block abuse of model APIs. Monitor API usage patterns to detect potential exploitation attempts.
6. **Implement continuous model monitoring**. Set up Amazon SageMaker AI Model Monitor to track data and concept drift that may indicate adversarial manipulation. Configure automatic alerts when inference data patterns deviate from training baselines. Periodically reevaluate model robustness as new threat techniques emerge in the security landscape.
7. **Establish incident response procedures**. Develop clear protocols for responding to detected adversarial threats against your ML systems. Define procedures for model rollback, data quarantine, and forensic analysis when suspicious activities are identified. Document lessons learned from security incidents to continuously improve protection strategies.

8. **Apply input validation guardrails**. For AI models, implement robust input validation and filtering mechanisms to block injection exploits. Implement custom guardrails to protect against harmful inputs that may manipulate model behavior. Monitor input patterns and responses to detect attempts to bypass security controls.

Resources

Related documents:

- Deep ensembles
- Empirical demonstration of deterministic overconfidence
- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- Accelerate generative AI development using managed MLflow on Amazon SageMaker AI
- Security and compliance
- 7 ways to improve security of your machine learning workflows
- Run ensemble ML models on Amazon SageMaker AI
- Securing Amazon SageMaker AI Studio connectivity using a private VPC

Monitoring

Best practices

- MLSEC06-BP01 Restrict access to intended legitimate consumers
- MLSEC06-BP02 Monitor human interactions with data for anomalous activity

MLSEC06-BP01 Restrict access to intended legitimate consumers

Use least privilege permissions to invoke the deployed model endpoint. For consumers who are external to the workload environment, provide access using a secure API.

Desired outcome: You establish secure inference API endpoints that allow only authorized parties to access your ML models. You create a controlled environment where model access is restricted based on legitimate business needs, while maintaining monitoring capabilities to track interactions with your models. Your ML endpoints are protected using the same security principles applied to other HTTPS APIs, providing data protection in transit and proper authentication.

Common anti-patterns:

- Allowing public access to model endpoints without proper authentication.
- Using overly permissive IAM roles for model endpoint access.
- Failing to implement network controls for inference endpoints.
- Not monitoring or logging model inference activities.
- Using the same credentials for development and production model access.

Benefits of establishing this best practice:

- Reduced risk of unauthorized model access and potential data exfiltration.
- Enhanced control over who can use your ML models and when.
- Improved monitoring and auditability of model usage.
- Protection of intellectual property embedded in models.
- Improves adherence to regulatory requirements for data security.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Securing your ML model endpoints is critical to protect both your intellectual property and the data processed by these models. By treating ML inference endpoints with the same security rigor as other HTTPS APIs, you can maintain a strong security posture while enabling legitimate business use. You need to implement proper authentication, network controls, and monitoring to verify that only authorized parties can access your models.

When deploying machine learning models in production, you should consider the model as a valuable asset that requires protection. This means implementing layers of security controls including network isolation through VPC configuration, strong authentication mechanisms, and continuous monitoring of inference activities. For external consumers, create a dedicated API layer that enforces security policies and controls access.

Implementation steps

1. **Plan your access control strategy**. Define which users or applications need access to your model endpoints and what specific permissions they require. Follow the principle of least privilege, granting only the minimum permissions necessary for each consumer to perform their required tasks.
2. **Set up secure inference API endpoints**. Host your ML models so that consumers can perform inference against them securely. Use Amazon SageMaker AI endpoints with proper

authentication and authorization controls. Use Amazon SageMaker AI Inference Recommender to automatically benchmark and optimize your model deployments for the best security-performance balance, and select optimal compute instances and configurations while maintaining security controls. This approach defines the relationship between the model and its consumers, restricts access to the base model, and provides monitoring capabilities for model interactions.

3. **Implement network security controls**. Configure your SageMaker AI inference endpoints within a VPC to isolate network traffic. Use security groups to define inbound and outbound traffic rules, and consider using AWS PrivateLink for private connectivity to your endpoints. Follow guidance from the AWS Well-Architected Framework to implement proper network controls, such as restricting access to specific IP ranges and implementing bot protection.
4. **Configure authentication and authorization**. Sign HTTPS requests for API calls so that requester identity can be verified. Use AWS Identity and Access Management (IAM) to control who has access to your SageMaker AI resources and what actions they can perform. Consider using Amazon Cognito for managing user identities if your API is accessed by external users.
5. **Deploy endpoints in a secure VPC configuration**. Use VPC endpoints to privately connect your VPC to supported AWS services without requiring an internet gateway. Follow the guidance in Give SageMaker AI Hosted Endpoints Access to Resources in Your Amazon VPC to configure your endpoint for VPC access.
6. **Implement encryption for data in transit and at rest**. Configure your endpoints to use HTTPS for API calls. Encrypt model artifacts and data at rest using AWS Key Management Service (KMS). Use client-side encryption for sensitive data when appropriate.
7. **Set up monitoring and logging**. Configure Amazon CloudWatch to monitor your endpoints and AWS CloudTrail to log API calls. Implement SageMaker AI Model Monitor to detect drift and data quality issues in your production models.
8. **Use model registry for governance**. Implement SageMaker AI MLflow Model Registry to catalog and manage versions of your models, control which model versions are deployed, and maintain an audit trail of model approvals and deployments.
9. **Implement proper API design patterns**. Design your inference API following REST best practices. Include proper input validation, error handling, and rate limiting to protect against abuse. Consider implementing an API Gateway in front of your SageMaker AI endpoint for additional controls.
10. **Conduct regular security reviews**. Periodically review the security configuration of your endpoints, check for over-permissive policies, and validate that access logs show only expected patterns of usage.

11. **Implement guardrails for AI models**. For AI endpoints, implement content filtering and validation controls to provide responsible outputs, stop harmful content generation, and maintain appropriate use of the models.

Resources

Related documents:

- Amazon SageMaker AI Inference Recommender
- Real-time Inference
- Give SageMaker AI Hosted Endpoints Access to Resources in Your Amazon VPC
- Accelerate generative AI development using managed MLflow on Amazon SageMaker AI
- Integrating machine learning models into your Java-based microservices
- How Financial Institutions can use AWS to Address Regulatory Reporting
- Secure deployment of Amazon SageMaker AI resources
- Accelerating Machine Learning Development with Data Science as a Service from Change Healthcare

Related videos:

- End-to-End machine learning using Spark and Amazon SageMaker AI

Related examples:

- Amazon SageMaker AI secure MLOps

MLSEC06-BP02 Monitor human interactions with data for anomalous activity

Implement comprehensive monitoring of data access events to detect unauthorized or suspicious activities. By auditing user interactions with data, you can identify potential security threats such as unusual access patterns, abnormal locations, or activity that exceeds normal baselines. Use specialized AWS services for anomaly detection alongside data classification to assess risks and protect your machine learning assets.

Desired outcome: You have comprehensive visibility into human interactions with your data, with logging enabled for create, read, update, and delete operations. You can identify who accessed

specific data elements, what actions they took, and when those actions occurred. Your monitoring system automatically flags anomalous activities based on established baselines and alerts you to potential security threats. Data classification is integrated with your monitoring approach to prioritize security events based on data sensitivity.

Common anti-patterns:

- Implementing logging without monitoring or analysis capabilities.
- Focusing only on system-level access without tracking specific data interactions.
- Failing to establish user activity baselines for anomaly detection.
- Not classifying data to differentiate between access to sensitive and non-sensitive information.
- Monitoring access events without automated alerting mechanisms.

Benefits of establishing this best practice:

- Early detection of potential data breaches or insider threats.
- Improved ability to investigate security incidents with comprehensive audit trails.
- Improves adherence to data protection regulations and requirements.
- Better visibility into how data is being used across your ML systems.
- Reduced risk of unauthorized data access or exfiltration.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Protecting your machine learning data requires visibility into who is accessing it and how it's being used. By monitoring human interactions with your data, you can identify potential security threats before they lead to data breaches or misuse. This involves implementing comprehensive logging for data access events, classifying your data based on sensitivity, and using automated tools to detect anomalous behavior.

Start by enabling logging for data interactions, particularly focusing on human access rather than just system-to-system communications. Your logs should capture details about who accessed the data, what specific elements they accessed, what actions they took, and when those interactions occurred. This creates an audit trail that serves as the foundation for your monitoring strategy.

Next, classify your data based on sensitivity and importance. By knowing which datasets contain personally identifiable information (PII), intellectual property, or other sensitive information, you

can prioritize monitoring efforts and apply appropriate security controls. This classification details the potential impact of unauthorized access to different datasets.

Finally, implement anomaly detection to identify unusual patterns that might indicate security threats. These anomalies could include access from unusual locations, outside normal working hours, excessive access volume, or access to data that's not typically needed for an employee's role. When anomalies are detected, your system should generate alerts to prompt investigation.

Implementation steps

1. **Enable data access logging**. Verify that you have data access logging for human CRUD (create, read, update, and delete) operations, including the details of who accessed what elements, what action they took, and at what time. Leverage AWS CloudTrail to capture API calls and user activities across your AWS environment. Configure CloudTrail to log data events for Amazon S3 buckets containing your training and inference data. For SageMaker AI environments, use Amazon SageMaker AI Logging and Monitoring capabilities to track access to ML models and datasets.
2. **Classify your data**. Use Amazon Macie for protecting and classifying training and inference data in Amazon S3. Amazon Macie is a fully managed security service that uses ML to automatically discover, classify, and protect sensitive data in AWS. The service recognizes sensitive data, such as personally identifiable information (PII) or intellectual property. Configure Macie to perform regular automated scans of your S3 buckets to identify and tag sensitive data. Create custom data identifiers in Macie to recognize organization-specific sensitive data patterns beyond the standard patterns Macie detects.
3. **Monitor and protect**. Use Amazon GuardDuty to monitor for malicious and unauthorized activities. This will enable protecting AWS accounts, workloads, and data stored in Amazon S3. Configure GuardDuty to analyze CloudTrail logs, VPC flow logs, and DNS logs to detect suspicious activities. Pay special attention to the GuardDuty S3 Finding Types which can detect anomalous access patterns to your S3-stored data.
4. **Set up anomaly detection**. Implement automated anomaly detection for data access patterns using Amazon CloudWatch Anomaly Detection. Create CloudWatch metrics for access frequency, data volume transferred, access times, and other relevant metrics. Configure CloudWatch alarms to alert when anomalies are detected based on these metrics.
5. **Establish data access baselines**. Create baseline profiles of normal user access patterns using AWS CloudWatch to monitor access trends over time. Set up dashboards that visualize normal patterns of data access by team, role, or time period. Use these baselines to fine-tune anomaly detection thresholds and reduce false positives.

6. **Implement alerting mechanisms**. Configure Amazon EventBridge to trigger automated responses when suspicious access events are detected. Route alerts to your security team through notification channels like Amazon SNS for immediate response. Create different alerting thresholds based on data classification and sensitivity.
7. **Centralize logging and monitoring**. Use Amazon OpenSearch Service (formerly Amazon OpenSearch Service) to create a centralized repository for log analysis and visualization. Build comprehensive dashboards to monitor data access patterns across your organization. Implement log retention policies that comply with your regulatory requirements.
8. **Control and audit data exploration activities**. Implement AWS Lake Formation with Amazon SageMaker AI Studio to provide fine-grained access controls for data exploration. Configure Lake Formation permissions to restrict data access based on user roles and data classification. Use AWS IAM to enforce least-privilege access to sensitive data.
9. **Monitor access to AI training data**. Implement specialized monitoring for datasets used to train AI models, as these may contain particularly sensitive information or be subject to greater privacy concerns. Use Amazon SageMaker AI Model Monitor to detect drift in model behavior that might indicate data access issues. Implement enterprise-ready security and privacy controls for foundation models.

Resources

Related documents:

- CloudWatch Logs for Amazon SageMaker AI
- GuardDuty S3 Protection finding types
- AWS CloudTrail Documentation
- Configure security in Amazon SageMaker AI
- Building a Self-Service, Secure, & Continually Compliant Environment on AWS
- How to Use New Advanced Security Features for Amazon Cognito user pools
- Best practices for setting up Amazon Macie with AWS Organizations

Related videos:

- Protect Your Data in S3 with Amazon Macie and Amazon GuardDuty - AWS Online Tech Talks
- Protecting sensitive data with Amazon Macie and Amazon GuardDuty

Related examples:

- AWS Security Hub CSPM automated response and remediation

Reliability

The reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it's expected to.

Each best practice in this section is presented based on its place in the ML lifecycle as detailed in *Machine learning lifecycle*.

Lifecycle phases

- Business goal identification
- ML problem framing
- Data processing
- Model development
- Deployment
- Monitoring

Business goal identification

There are no reliability best practices specific to the business goal identification phase.

ML problem framing

Best practices

- MLREL01-BP01 Use APIs to abstract change from model consuming applications
- MLREL01-BP02 Adopt a machine learning microservice strategy

MLREL01-BP01 Use APIs to abstract change from model consuming applications

APIs abstract changes from model-consuming applications, keeping machine learning solutions flexible and resilient. Establishing an abstraction layer between ML models and consuming applications enables model updates, replacements, or enhancements without disrupting existing workloads.

Desired outcome: You have a flexible application and API design that isolates machine learning model implementations from consuming applications. You make changes to ML models with minimal or no disruption to existing applications. Your ML endpoints are well-documented and accessible, and changes to underlying models do not require extensive modifications to downstream applications.

Common anti-patterns:

- Directly embedding model code within applications.
- Hardcoding model versions or parameter specifications in client applications.
- Lacking proper API documentation and version control.
- Designing rigid interfaces that break when model inputs or outputs change.
- Creating tight coupling between ML models and consuming applications.

Benefits of establishing this best practice:

- Reduces downtime when updating or replacing ML models.
- Simplifies model deployment and versioning processes.
- Increases agility and flexibility when evolving ML capabilities.
- Lowers maintenance costs for applications using ML models.
- Enhances ability to A/B test different model versions.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Abstracting changes from model-consuming applications requires thoughtful API design and implementation. Create a well-designed API layer between ML models and applications so that you can make modifications to models without disrupting services. This approach involves developing stable interfaces that hide underlying complexity and implementation details of ML models.

When designing these APIs, focus on creating contracts that are flexible enough to accommodate model evolution while maintaining backward compatibility. Document APIs thoroughly so developers consuming models understand how to interact with them correctly. Consider implementing versioning strategies that allow introducing new model capabilities while supporting existing clients.

Implementation steps

1. **Adopt best practices in API design**. Expose ML endpoints through APIs so changes to the model can be introduced without disrupting upstream communications. Create a well-designed API contract that focuses on business capabilities rather than technical implementation details. Document your API in a central repository or documentation site so calling services can understand API routes and flags. Communicate changes to your API with calling services.
2. **Implement API versioning**. Use versioning strategies for APIs to enable backward compatibility while supporting new features. Consider using URL path versioning (for example, /v1/predict), header-based versioning, or query parameter versioning depending on organizational standards. This allows introducing new model versions without breaking existing client applications.
3. **Deploy models in Amazon SageMaker AI**. After training your model, deploy it using Amazon SageMaker AI to get predictions. To establish a persistent endpoint for one prediction at a time, use SageMaker AI hosting services. For predictions on entire datasets, use SageMaker AI batch transform. SageMaker AI provides flexibility in deployment options, including multi-model endpoints, serverless inference, and asynchronous inference.
4. **Use Amazon API Gateway to create APIs**. Amazon API Gateway is a fully managed service that enables developers to create, publish, maintain, monitor, and secure APIs. Using API Gateway, you can create RESTful APIs and WebSocket APIs that enable real-time two-way communication applications. API Gateway supports containerized and serverless workloads, as well as web applications.
5. **Implement request and response transformations**. Use API Gateway's mapping templates to transform client requests to match your model's input format and transform model responses to maintain a consistent API contract. This allows changing model implementations without requiring client applications to change how they format requests or interpret responses.
6. **Add caching and throttling**. Configure API Gateway's caching capability to improve performance and reduce costs for frequently accessed predictions. Implement throttling to protect ML endpoints from traffic spikes and provide consistent performance. Use SageMaker AI Inference Recommender to optimize endpoint configurations for optimal latency and cost performance.
7. **Monitor and analyze API usage**. Set up monitoring and logging for APIs to understand how they are being used and identify potential issues. Use Amazon CloudWatch metrics and logs to track API performance, errors, and usage patterns. This data can optimize ML endpoints and identify opportunities for improvement.

8. **Consider inference components for shared endpoints**. Use SageMaker AI inference components to deploy multiple models to shared endpoints, improving resource utilization and reducing costs while maintaining API abstraction.

Resources

Related documents:

- Model deployment options in Amazon SageMaker AI
- What is Amazon API Gateway?
- Real-time inference
- Multi-model endpoints
- Deploy models with Amazon SageMaker AI Serverless Inference
- Asynchronous inference
- Deploying ML models using SageMaker AI Serverless Inference
- Optimize deployment cost of Amazon SageMaker AI JumpStart foundation models with Amazon SageMaker AI asynchronous endpoints

Related videos:

- Amazon Sagemaker Serverless Inference

Related examples:

- Amazon SageMaker AI Examples Repository
- Amazon SageMaker AI MLOps Workshop

MLREL01-BP02 Adopt a machine learning microservice strategy

Machine learning systems can be effectively implemented through a microservice architecture that breaks down complex business problems into smaller, loosely coupled components. This approach enables distributed development, improves scalability, and facilitates change management while reducing the impact of single failures on the overall workload.

Desired outcome: You decompose complex business problems into manageable components with clear interfaces. You have a more resilient ML system architecture that can scale individual

components independently, enable faster iterations, and allow specialized teams to work simultaneously on different parts of the solution. Your organization benefits from improved fault isolation, simplified testing, and greater flexibility to update or replace individual ML models without affecting the entire system.

Common anti-patterns:

- Building monolithic ML applications where functionality is tightly coupled in a single runtime.
- Creating overly complex microservices that serve multiple purposes.
- Implementing microservices without clear business domain boundaries.
- Neglecting proper service interfaces and communication patterns between microservices.
- Overlooking the operational complexity introduced by distributed systems.

Benefits of establishing this best practice:

- Improves resilience through isolation of failures to individual components.
- Enhances scalability by allowing independent scaling of individual services.
- Accelerates development cycles through parallel work streams.
- Simplifies testing and deployment of individual components.
- Provides flexibility to use different technologies for different ML model requirements.
- Aligns technical components with business domains.
- Eases integration of new ML models and capabilities.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

When implementing ML systems, adopt a microservice architecture to break down complex problems into manageable components. Rather than creating one large monolithic application that handles the entirety of of your machine learning workflow, you can develop specialized services that handle specific functions like data preprocessing, model training, inference, and business logic integration. This approach is particularly valuable for machine learning applications where different models may have varying resource requirements, development cycles, and deployment frequencies.

Microservices provide the flexibility to use different technologies for different parts of your ML system. For example, you might use Python-based services for data science tasks while

implementing Java-based services for integration with enterprise systems. By establishing clear service interfaces, you verify that these components work together seamlessly while maintaining freedom.

When designing your ML microservices, focus on business domains rather than technical functions. Instead of creating a generic prediction service, you might create more specific services like *customer churn prediction* or *product recommendation* that align with business capabilities. This domain-driven approach makes your architecture more intuitive and adaptable to changing business needs.

Implementation steps

1. **Adopt a microservice strategy**. Implement a service-oriented architecture (SOA) by making software components reusable through service interfaces. Break your monolithic application into separate components along business boundaries or logical domains. Focus on building single-purpose applications that can be composed in different ways to deliver various end-user experiences. Design clear APIs between services to implement proper isolation and communication patterns.
2. **Define domain boundaries**. Analyze your machine learning workflow and identify natural boundaries between different functional areas. Map out the data flow and determine where service boundaries make sense. Consider creating separate microservices for data ingestion, preprocessing, feature engineering, model training, model serving, and business logic integration. Verify that each microservice has a clear, single responsibility within your ML system.
3. **Choose appropriate AWS services**. Select AWS services that best support your microservice architecture. AWS Lambda provides serverless compute that scales automatically and charges only for the compute time consumed. For container-based deployments, use AWS Fargate to run containers without managing infrastructure. Consider Amazon ECS or Amazon EKS for container orchestration needs, and Amazon API Gateway to manage and secure your microservice APIs.
4. **Implement model serving infrastructure**. Set up efficient model serving using services like Amazon SageMaker AI for deploying, monitoring, and scaling ML models. SageMaker AI endpoints provide a managed solution for hosting models and handling inference requests. Alternatively, use AWS Lambda for lightweight, event-driven inferencing or containers on AWS Fargate for more complex requirements.
5. **Establish communication patterns**. Design how your microservices will communicate with each other. Use synchronous REST APIs for direct request-response patterns, and asynchronous communication through Amazon SNS or Amazon SQS for event-driven architectures. Implement

AWS EventBridge to create event-driven workflows between your ML microservices and other AWS services.

6. **Implement monitoring and observability**. Set up comprehensive monitoring for your ML microservices using Amazon CloudWatch. Track operational metrics like latency, throughput, and error rates along with ML-specific metrics such as prediction accuracy or drift. Implement distributed tracing with AWS X-Ray to troubleshoot issues across service boundaries and identify performance bottlenecks.
7. **Automate deployments**. Implement CI/CD pipelines using AWS CodePipeline and AWS CodeBuild to automate the testing and deployment of your ML microservices. Use infrastructure as code with AWS CloudFormation or AWS CDK to define and provision your microservice infrastructure consistently.
8. **Implement security best practices**. Secure your ML microservices by implementing proper authentication and authorization using Amazon Cognito or AWS IAM. Use AWS WAF to protect your APIs from common web exploits, and encrypt sensitive data using AWS KMS to improve data privacy and adhere to regulatory requirements.

Resources

Related documents:

- Implementing Microservices on AWS
- AWS Lambda Documentation
- Architect for AWS Fargate for Amazon ECS
- What is Amazon SageMaker AI?
- What is the AWS Serverless Application Model (AWS SAM)?
- Build and deploy ML inference applications from scratch using Amazon SageMaker AI

Related examples:

- Run a Serverless "Hello, World" with AWS Lambda

Data processing

Best practices

- MLREL02-BP01 Use a data catalog

- MLREL02-BP02 Use a data pipeline
- MLREL02-BP03 Automate managing data changes

MLREL02-BP01 Use a data catalog

Process data across multiple data stores using data catalog technology. An advanced data catalog service can enable ETL process integration. This approach improves reliability and efficiency.

Desired outcome: You establish a centralized system that inventories your ML data assets across the organization. You can track, discover, and manage data transformations, and your stakeholders can find and use appropriate datasets. With a complete data catalog in place, you gain better data governance, reduce duplication of effort, increase data quality, and accelerate ML model development through streamlined data preparation workflows.

Common anti-patterns:

- Maintaining separate, isolated data silos without centralized metadata.
- Manual tracking of data assets using spreadsheets or wiki pages.
- Failing to document data transformations and lineage.
- Rebuilding ETL processes for each new ML project.
- Relying on tribal knowledge for understanding data characteristics.

Benefits of establishing this best practice:

- Provides centralized visibility of available data assets.
- Improves data governance.
- Reduces time spent searching for and understanding data.
- Enhances data quality through consistent transformation processes.
- Strengthens collaboration between data engineers and data scientists.
- Accelerates ML model development with faster data preparation.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

A data catalog is critical infrastructure for machine learning workloads to succeed. Without proper cataloging, data scientists spend excessive time searching for, understanding, and preparing data

rather than analyzing it or building models. A comprehensive data catalog provides a centralized inventory of your data assets, complete with metadata, transformation history, and usage information.

When implementing a data catalog for ML workloads, focus on creating a single source of truth for data discovery. The catalog should document where data resides, what transformations have been applied, and how it can be accessed. This reduces the time spent on data preparation, which typically consumes 60-80% of a data scientist's time.

For AWS environments, the AWS AWS Glue Data Catalog offers a powerful solution as it integrates seamlessly with other AWS analytics and machine learning services. By implementing a data catalog as part of your ML infrastructure, you create a foundation for consistent, reliable data processing that supports both current and future ML initiatives.

Implementation steps

1. **Set up AWS AWS Glue Data Catalog**. Establish the AWSAWS Glue Data Catalog as your central metadata repository. This fully managed service provides a unified view of your data across multiple data stores, making it accessible to various AWS services like Amazon SageMaker AI, Amazon EMR, Amazon Athena, and Amazon Redshift.
2. **Define your metadata strategy**. Determine what metadata to capture about each dataset, including business definitions, data lineage, quality metrics, and usage patterns. Well-documented metadata assists data scientists in quickly understanding if a dataset is appropriate for their modeling needs.
3. **Populate the data catalog**. Use AWS Glue crawlers to automatically discover schemas, data types, and statistics from your data sources. Crawlers can scan various data stores including Amazon S3, Amazon RDS, and other database systems. Systematically add your data sources to build comprehensive coverage.
4. **Implement ETL processes with AWS Glue**. Create ETL jobs that transform raw data into formats optimized for machine learning. AWS Glue can automatically generate Python or Scala code for these transformations, reducing development time and improving consistency across different data processing pipelines.
5. **Establish data lineage tracking**. Configure your data catalog to track transformations and maintain clear lineage information. With data lineage tracking, data scientists can understand data provenance, which builds trust in the datasets used for model training.
6. **Integrate with ML workflows**. Connect your AWS AWS Glue Data Catalog with Amazon SageMaker AI to streamline data access for model training. For enterprise environments,

implement Amazon SageMaker AI Catalog as a central metadata hub for ML and data assets, enabling secure sharing and governed access via Amazon DataZone integration. This integration allows data scientists to discover and use properly prepared datasets directly from their modeling environments.

7. **Implement access controls and governance**. Configure appropriate permissions for your data catalog using AWS IAM roles. Verify that data scientists have access to the metadata and datasets they need while maintaining security and regulatory adherence.
8. **Automate catalog maintenance**. Set up automated processes to keep your data catalog current as new data arrives and transformations occur. Regular updates verify that data scientists have access to the latest information about available datasets.
9. **Monitor and measure impact**. Track key metrics like time saved in data discovery, reduction in redundant data preparation work, and improvements in model development cycles to quantify the benefits of your data catalog implementation.

Resources

Related documents:

- Amazon SageMaker AI and when to use Amazon SageMaker AI vs Amazon DataZone
- The lakehouse architecture of Amazon SageMaker AI
- Amazon SageMaker AI Unified Studio
- Catalog and search
- Getting started with serverless ETL on AWS Glue
- Using crawlers to populate the Data Catalog
- AWS modern data architecture
- Moving from development to automated ML pipelines using Amazon SageMaker AI and AWS Glue
- How Genworth built a serverless ML pipeline on AWS using Amazon SageMaker AI and AWS Glue
- How to build and automate a serverless data lake using AWS Glue

Related videos:

- Amazon SageMaker AI Lakehouse - Unified, open, and secure data lakehouse
- Understanding Amazon S3 Tables: Architecture, performance, and integration
- Accelerate your Analytics and AI with Amazon SageMaker AI LakeHouse

- Unifying data governance with Immuta and AWS Lake Formation

Related examples:

- Explaining Credit Decisions with Amazon SageMaker AI
- How to build an end-to-end Machine Learning pipeline using AWS Glue, Amazon S3, Amazon SageMaker AI and Amazon Athena

MLREL02-BP02 Use a data pipeline

Automate the processing, movement, and transformation of data between different compute and storage services. This automation enables data processing that is fault tolerant, repeatable, and highly available.

Desired outcome: You achieve streamlined and consistent data processing workflows that automatically handle data movement and transformations. You can process your machine learning data with increased reliability, repeatability, and availability, while reducing manual effort and potential errors. Your data processing becomes more efficient and scalable, enabling you to focus on deriving insights rather than managing data logistics.

Common anti-patterns:

- Manually moving and transforming data between systems.
- Creating one-off scripts for data processing tasks.
- Inconsistent data transformation processes across teams.
- Neglecting error handling and recovery mechanisms in data workflows.
- Not versioning data processing code or configurations.

Benefits of establishing this best practice:

- Reduces manual errors and inconsistencies in data processing.
- Increases repeatability and reliability of data transformations.
- Enables fault tolerance and automatic recovery mechanisms.
- Improves scalability of data processing workflows.
- Facilitates collaboration through standardized data processing patterns.
- Enhances traceability and governance of data transformations.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Data is the foundation of a machine learning workload, and how you handle this data directly impacts the quality of your ML models. Data pipelines automate and standardize the process of collecting, cleaning, transforming, and delivering data to your ML workflows. Without proper data pipelines, your ML initiatives can suffer from inconsistent data quality, limited reproducibility, and operational inefficiencies.

Creating effective data pipelines requires careful planning around data sources, transformation logic, error handling, and monitoring capabilities. By implementing automated data pipelines, you verify that your data preparation follows a consistent, repeatable process that can scale with your ML workload demands. This approach leads to more reliable models and faster deployment cycles.

AWS provides a comprehensive set of tools specifically designed to build robust ML data pipelines, with Amazon SageMaker AI offering integrated capabilities for the entire ML lifecycle. These tools enable you to focus on deriving insights from your data rather than managing infrastructure.

Implementation steps

1. **Assess your data processing requirements**. Begin by identifying your data sources, required transformations, and destination systems. Document the flow of data from source to consumption, noting data quality requirements, validation rules, or business logic that must be applied during processing.
2. **Implement data preparation and wrangling processes**. Establish comprehensive data preparation workflows that transform raw data into ML-ready formats. Use a combination of AWS services and tools to:
   - Connect to data sources including Amazon S3, Amazon Athena, Amazon Redshift, and other databases using appropriate connectors
   - Explore and profile your data using Amazon EMR with Apache Spark or AWS Glue interactive sessions to identify patterns, anomalies, and data quality issues
   - Transform your data using AWS Glue ETL jobs, Amazon EMR clusters, or SageMaker AI Processing jobs with custom transformation scripts
   - Generate data quality reports and validation checks using AWS Glue DataBrew or custom validation scripts to identify issues before model training
   - Create reusable data preparation workflows using AWS Glue workflows or SageMaker AI Pipelines that verify consistency across different datasets and projects

3. **Build automated ML workflows with SageMaker AI Pipelines**. After creating your data preparation workflow, export it to Amazon SageMaker AI Pipelines to automate your entire ML workflow. SageMaker AI Pipelines helps you:
   - Create end-to-end ML workflows that combine data preparation, model training, evaluation, and deployment
   - Automate pipeline execution on a schedule or trigger-based approach
   - Track lineage of ML artifacts for governance
   - Implement quality gates to verify that models meet performance criteria
   - Version control your pipelines for reproducibility
4. **Implement error handling and monitoring**. Configure your pipelines to handle errors gracefully and provide visibility into pipeline performance:
   - Set up retry mechanisms for transient failures
   - Create notification systems for critical pipeline failures
   - Implement logging throughout your pipeline steps
   - Use Amazon CloudWatch to monitor pipeline metrics and set up alerts
5. **Version and store your data artifacts**. Maintain traceability of your data processing:
   - Store processed datasets in Amazon S3 with appropriate versioning
   - Use Amazon SageMaker AI Feature Store to create reusable feature repositories
   - Document data transformations and their business logic
   - Implement data lineage tracking to understand data provenance
6. **Integrate with your existing ML workflow**. Connect your data pipelines to other components of your ML environment:
   - Feed processed data directly into model training jobs
   - Integrate with model registries and deployment pipelines
   - Establish feedback loops from model performance back to data preparation
7. **Scale your data processing as needed**. Configure your pipelines to handle growing data volumes:
   - Use distributed processing for large datasets with services like Amazon EMR
   - Implement incremental processing patterns for streaming data
   - Configure compute resources appropriately for each pipeline stage

Resources

Related documents:

- Data transformation workloads with SageMaker AI Processing

- Pipelines
- Get started with Amazon SageMaker AI Feature Store
- Data preparation and cleaning
- Run secure processing jobs using PySpark in Amazon SageMaker AI Pipelines
- Building, automating, managing, and scaling ML workflows using Amazon SageMaker AI Pipelines

Related videos:

- AWS Glue and Amazon SageMaker AI Studio Integration
- Build, automate, manage, and scale ML workflows using Amazon SageMaker AI Pipelines

Related examples:

- Amazon SageMaker AI Examples GitHub Repository
- MLOps Workshop with Amazon SageMaker AI Pipelines

MLREL02-BP03 Automate managing data changes

Effective management of machine learning training data changes is crucial for maintaining model reproducibility and providing consistent performance over time. By implementing automated version control for training data, you can precisely recreate a model version when needed and maintain a clear audit trail of data transformations.

Desired outcome: You establish automated processes for tracking and managing changes to your training data using version control technology. You gain the ability to reproduce model versions exactly as they were originally created, track data lineage through your ML pipeline, and maintain consistent model performance across deployments. Your ML operations become more reliable, transparent, and compatible with governance requirements.

Common anti-patterns:

- Manually tracking data versions in spreadsheets or documentation.
- Storing multiple versions of datasets with inconsistent naming conventions.
- Neglecting to record relationships between datasets and resulting models.
- Not preserving feature engineering transformations applied to training data.
- Relying on ad-hoc backup processes instead of systematic version control.

Benefits of establishing this best practice:

- Enables reproducible machine learning by maintaining exact data version history.
- Improves troubleshooting by allowing precise recreation of model versions.
- Enhances collaboration among data scientists through shared version control.
- Provides audit trail for governance requirements.
- Reduces errors in model deployment by providing consistent training data.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Managing changes to training data is fundamental to maintaining reproducible machine learning models. As your data evolves through acquisition, cleaning, and feature engineering, implementing automated version control allows you to track these changes systematically. This provides confidence that you can recreate any model version precisely when needed, which is essential for troubleshooting, compliance alignment, and proviidng consistent performance.

By implementing automated data versioning, you create a traceable history of your training data that integrates seamlessly with your ML pipeline. This approach mirrors software development best practices by treating data as a critical asset requiring the same level of version control as code. When data changes occur, whether through new acquisitions or transformations, your versioning system automatically captures these changes, making it possible to track model lineage from training data to deployment.

Implementation steps

1. **Implement a data version control system**. Begin by setting up a data version control system that can handle ML datasets efficiently. Tools like Git LFS, DVC (Data Version Control), or AWS solutions can be used to track changes in your training datasets. These tools provide mechanisms to capture dataset metadata and references without storing the entire dataset in the version control repository.
2. **Establish a data management strategy**. Define clear workflows for how data should be versioned, including naming conventions, branching strategies, and metadata requirements. Document how data should flow through your ML pipeline and how versions will be tracked at each stage.
3. **Use AWS MLOps Framework**. Implement the AWS MLOps Framework to establish a standardized interface for managing ML pipelines. This framework works with both Amazon

Machine Learning services and third-party services, providing a comprehensive solution for ML operations. The framework allows you to upload trained models (bring your own model), configure pipeline orchestration, and monitor operations—all while maintaining version control of data assets.

4. **Integrate with SageMaker AI Model Registry**. Use Amazon SageMaker AI Model Registry to track model versions and their associated artifacts. Model Registry maintains comprehensive records of model lineage, including which datasets were used for training and validation, preserving the connection between models and their source data.
5. **Establish CI/CD for ML pipelines**. Set up continuous integration and continuous deployment (CI/CD) pipelines specifically designed for ML workflows using Amazon SageMaker AI Pipelines. This assists you to version and test changes to both code and data properly before moving to production.
6. **Create reproducible training environments**. Use container technology to package your training environment along with references to specific data versions. Amazon SageMaker AI provides mechanisms to create reproducible training jobs that can reference specific versions of your datasets stored in Amazon S3.
7. **Implement data quality monitoring**. Set up automated monitoring of data quality metrics to detect drift or anomalies in incoming data. Tools like Amazon SageMaker AI Model Monitor can identify when new data differs from the baseline training data, allowing you to make informed decisions about model retraining.
8. **Configure automated testing**. Implement automated tests that validate data consistency and model performance when data versions change. This verifies that new data meets quality standards before being used in training or inference.
9. **Document data versioning procedures**. Create comprehensive documentation that describes your data versioning strategy, including how to retrieve specific versions of datasets and how to match models with their corresponding training data versions.

Resources

Related documents:

- Implement MLOps
- Model Registration Deployment with Model Registry
- Amazon SageMaker AI Feature Store
- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- Amazon SageMaker AI Pipelines Brings DevOps Capabilities to your Machine Learning Projects

- Building, automating, managing, and scaling ML workflows using Amazon SageMaker AI Pipelines
- Fully managed MLflow 3.0 on Amazon SageMaker AI

Related videos:

- Implementing End-to-End MLOps Solutions with Amazon SageMaker AI
- Accelerate production for gen AI using Amazon SageMaker AI MLOps & FMOps
- Deliver high-performance ML models faster with MLOps tools

Related examples:

- Amazon SageMaker AI secure MLOps
- ML Pipelines using Amazon SageMaker AI

Model development

Best practices

- MLREL03-BP01 Enable CI/CD/CT automation with traceability
- MLREL03-BP02 Verify feature consistency across training and inference
- MLREL03-BP03 Validate models with relevant data
- MLREL03-BP04 Establish data bias detection and mitigation

MLREL03-BP01 Enable CI/CD/CT automation with traceability

Enable source code, data, and artifact version control of ML workloads to enable roll back to a specific version. Incorporate continuous integration (CI), continuous delivery (CD), and continuous training (CT) practices to ML workload operations, providing automation with added traceability.

Desired outcome: You establish automated pipelines that handle the entire machine learning lifecycle from development to deployment and continuous training. You gain the ability to track every artifact, model version, dataset, and code change throughout the ML workflow, enabling transparent auditing, reproducibility of experiments, and the capability to quickly roll back to previous versions when needed.

Common anti-patterns:

- Manual deployment and training processes without version control.
- Lack of documentation on model lineage and data provenance.
- Inability to reproduce ML experiments due to missing environment configurations.
- Performing model training only when necessary without automated testing and validation.
- Siloed development and operations teams working separately on ML workflows.

Benefits of establishing this best practice:

- Increases productivity through automation of repetitive ML development tasks.
- Improves reproducibility of ML experiments and model training.
- Enhances collaboration between data scientists and operations teams.
- Accelerates time-to-market for ML-powered features and applications.
- Reduces risk through ability to quickly roll back problematic deployments.
- Improves adherence to audit requirements through comprehensive traceability.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Implementing CI/CD/CT for machine learning workloads requires a different approach than traditional software development due to the data-centric nature of ML systems. While software CI/CD focuses primarily on code, ML pipelines must also track data, model artifacts, and training environments for full reproducibility.

MLOps combines DevOps practices with machine learning to automate and streamline the entire lifecycle of ML systems. By implementing MLOps with traceability, you create a foundation that supports reproducible science, auditability, and operational excellence. This allows your organization to deploy ML models with confidence while maintaining the ability to understand exactly how each model was created and what data influenced its behavior.

Amazon SageMaker AI provides a comprehensive solution to implement MLOps practices with built-in version control, lineage tracking, and pipeline automation. By using SageMaker AI and complementary AWS services, you can establish a robust MLOps framework that makes your ML workflows reproducible, traceable, and maintainable.

Implementation steps

1. **Implement version control for ML artifacts**. Set up repositories for code, data, models, and configurations using version control systems. Use AWS CodeCommit or integrate with GitHub to

version your ML code and configurations. For data and models, use Amazon SageMaker AI Model Registry to version and catalog your models, creating a system of record that tracks the lineage of each model.

2. **Set up data versioning and lineage tracking**. Implement data version control to track changes in your datasets over time. Use Amazon SageMaker AI Feature Store to store, share, and manage features for ML development, which you can use to track and version feature values. Implement Lineage Tracking to track the relationships between ML artifacts including data, models, training jobs, and deployments.
3. **Establish continuous integration practices**. Configure automated tests that verify both code quality and model performance. Set up CI pipelines using AWS CodeBuild that run unit tests, integration tests, and model quality tests when changes are pushed to your repositories. Implement automated code review practices to maintain quality standards across your ML codebase.
4. **Build continuous delivery pipelines for models**. Create automated deployment pipelines for ML models using Amazon SageMaker AI Pipelines or AWS CodePipeline. Configure pipelines to include stages for data preparation, model training, evaluation, and deployment. Implement approval gates for human validation before models are deployed to production environments.
5. **Implement continuous training mechanisms**. Set up automated retraining pipelines that can be triggered by data drift, scheduled intervals, or on-demand. Use Amazon SageMaker AI Pipelines to create end-to-end workflows for model retraining. Implement monitoring for model drift using SageMaker AI Model Monitor and trigger retraining when performance degrades below thresholds.
6. **Establish model governance and approval workflows**. Create a governance framework that requires appropriate reviews and approvals before models move to production. Use SageMaker AI Model Registry to implement model approval workflows with different approval stages. Configure integration with notification services like Amazon SNS to alert stakeholders when models need review.
7. **Implement immutable infrastructure for reproducibility**. Use infrastructure as code (IaC) to define your ML environments consistently. Leverage AWS CloudFormation or AWS CDK to define your SageMaker AI environments, maintaining consistency across development, testing, and production. Create standardized container images for training and inference to improve environment reproducibility.
8. **Set up comprehensive monitoring and logging**. Implement monitoring for your ML pipelines and deployed models. Use CloudWatch to track operational metrics of your pipeline runs and

model endpoints. Configure SageMaker AI Model Monitor to track data drift, model drift, and prediction quality over time.

9. **Create rollback mechanisms for models and pipelines**. Establish automated rollback procedures that can quickly revert to previous model versions when issues are detected. Configure SageMaker AI endpoints with production variants to support blue/green deployments and canary testing, enabling safe rollbacks when needed.

Resources

Related documents:

- Implement MLOps
- Pipelines overview
- AWS DevOps Guidance
- Model Registration Deployment with Model Registry
- Building, automating, managing, and scaling ML workflows using Amazon SageMaker AI Pipelines
- Create Amazon SageMaker AI projects with image building CI/CD pipelines

Related videos:

- Deliver high-performance ML models faster with MLOps tools
- Accelerate production for gen AI using Amazon SageMaker AI MLOps & FMOps

Related examples:

- Amazon Sagemaker MLOps
- Amazon SageMaker AI secure MLOps

MLREL03-BP02 Verify feature consistency across training and inference

Provide consistent, scalable, and highly available features between training and inference using a feature storage. This results in reducing the training-serving skew by keeping feature consistency between training and inference.

Desired outcome: You create a centralized feature repository where feature definitions are stored, versioned, and shared across your organization. This makes the same features used during model

training consistently available during inference, which reduces training-serving skew. You can discover, reuse, and share features across different ML projects, reducing duplicate work and standardizing feature engineering practices.

Common anti-patterns:

- Recreating feature transformations separately for training and inference pipelines.
- Storing features in different formats or locations for training versus production.
- Lack of versioning for features, leading to inconsistencies when models are updated.
- Duplicating feature engineering work across different teams or projects.
- Using non-standardized approaches to feature storage and retrieval.

Benefits of establishing this best practice:

- Reduces training-serving skew, leading to more reliable model performance in production.
- Increases developer productivity through feature reusability.
- Standardizes feature definitions across the organization.
- Improves model governance and auditability through feature versioning.
- Saves costs by avoiding redundant feature computation and storage.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Feature consistency between training and inference is critical for machine learning system reliability. When the features used to train a model differ from those used during inference, model performance can degrade, a problem known as training-serving skew. To avoid this issue, you need a centralized feature repository that provides consistent access to the same feature definitions and transformations across both training and inference environments.

A feature store serves as this centralized repository, enabling you to define, store, and retrieve features consistently. It provides mechanisms for versioning features, which verifies that you can maintain compatibility between models and the features they expect as your data and feature engineering processes evolve. Additionally, a feature store allows for the sharing and reuse of features across multiple ML projects, increasing efficiency and standardizing feature engineering practices across your organization.

Implementation steps

1. **Set up Amazon SageMaker AI Feature Store**. Create and configure a SageMaker AI Feature Store to serve as your centralized repository for ML features. SageMaker AI Feature Store provides both online and offline storage capabilities: the online store supports low-latency, real-time inference use cases, while the offline store supports training and batch inference processes. Create a feature group that defines your feature structure, data types, and storage configurations using the SageMaker AI SDK or console.
2. **Define feature groups and schemas**. Organize your features into logical groups based on business domains, data sources, or ML use cases. Define schemas for your features, including data types, descriptions, and metadata. This organization makes features more discoverable and more straightforward to manage across your organization.
3. **Implement feature ingestion pipelines**. Build automated pipelines to ingest and process raw data into features. Use SageMaker AI Processing, AWS Glue, or Amazon EMR to transform raw data into feature values. Configure both batch ingestion for historical data and streaming ingestion for real-time updates using services like Amazon Kinesis with SageMaker AI Feature Store.
4. **Develop feature retrieval mechanisms**. Create standardized ways to retrieve features for both training and inference. For training datasets, implement code that pulls features from the offline store, while for inference, develop services that query the online store. Verify that both paths use the same feature definitions and transformations.
5. **Integrate with ML workflows**. Connect your feature store to your ML pipelines by integrating it with SageMaker AI Pipelines or your custom ML workflows. This makes feature retrieval consistent throughout the ML lifecycle, from experimentation to production deployment.
6. **Monitor and validate features**. Implement monitoring for your feature store to detect data drift, missing values, or other quality issues. Use SageMaker AI Model Monitor or custom validation scripts for feature consistency and quality over time. Set up alerts for deviations in feature distributions.
7. **Enable feature discovery and sharing**. Document your features with metadata and descriptions to make them discoverable across your organization. Integrate with data catalogs like AWSAWS Glue Data Catalog to enhance discoverability and governance of features.

Resources

Related documents:

- Get started with Amazon SageMaker AI Feature Store
- Feature Store concepts
- Store, Discover, and Share Machine Learning Features with Amazon SageMaker AI Feature Store
- Unlock ML insights using the Amazon SageMaker AI Feature Store Feature Processor

Related videos:

- Amazon SageMaker AI Feature Store: Store, discover, & share features for ML apps
- Deep dive on Amazon SageMaker AI Feature Store

Related examples:

- Introduction to Feature Store
- Amazon SageMaker AI Feature Store SageMaker AI Examples
- Amazon SageMaker AI Feature Store: Streaming Aggregation

MLREL03-BP03 Validate models with relevant data

Testing and validating machine learning models with appropriate data is essential for reliable performance in production. Use real and representative data that covers many possible patterns and scenarios to avoid model failures when deployed in real-world environments.

Desired outcome: You establish processes that validate your machine learning models with real-world, representative data before deployment. You can identify distribution mismatches between training, validation, test, and inference data early, allowing you to address issues before they impact production performance. Your validation approach includes both real-world and engineered data to account for the scenarios your model might encounter.

Common anti-patterns:

- Testing models with only synthetic data that doesn't represent real-world conditions.
- Failing to check for distribution mismatch between training and production data.
- Ignoring edge cases and rare scenarios in validation datasets.
- Using validation data that lacks diversity or has sampling biases.
- Neglecting periodic revalidation after models are deployed.

Benefits of establishing this best practice:

- Reduces risk of model failures in production environments.
- Enables early detection of data drift and model quality degradation.
- Creates more robust models that perform well across expected scenarios.
- Increases trust in model predictions from stakeholders.
- Improves alignment between model performance in testing and production.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Validating your machine learning models with relevant data is a critical step in the ML development lifecycle. Data that fails to represent the full range of scenarios your model will encounter in production can lead to poor performance, biased outputs, or complete failures when deployed. The key challenge lies in obtaining and using data that accurately mirrors your production environment.

Begin by analyzing your data sources to capture the breadth and depth of real-world scenarios. This includes common cases as well as edge cases that might be rare but important. For example, a model designed to detect fraudulent transactions needs exposure to both typical and unusual fraud patterns. Missing these edge cases can create vulnerabilities in your deployed model.

Pay particular attention to distribution mismatches, where the statistical properties of your training, validation, test, and eventual inference data differ. These mismatches often lead to degraded model performance in production. For instance, if you train a product recommendation model on data from one demographic group but deploy it for a different group, the model may make irrelevant recommendations.

Implement continuous monitoring after deployment to detect when the data distribution shifts over time, which is common in real-world applications. This allows you to retrain models before performance degradation impacts business outcomes.

Implementation steps

1. **Establish data quality criteria**. Define what constitutes representative data for your use case. Include requirements for data completeness, diversity, and coverage of edge cases. Document these criteria as part of your ML development process to create consistency across projects.
2. **Implement cross-validation techniques**. Use techniques like k-fold cross-validation to verify that your model generalizes well across different subsets of your data. This can identify

potential overfitting issues before deployment and provides a more robust estimate of how your model will perform in production.

3. **Use Amazon SageMaker AI MLFlow Tracking**. Set up experiments to track and compare different training runs with various data configurations. SageMaker AI MLFlow Tracking allows you to organize, monitor, and evaluate your machine learning experiments systematically. This can identify which data configurations lead to the best performance and provides a historical record for future reference.
4. **Create engineered test data**. Generate synthetic data to supplement real-world data, especially for rare but important edge cases. Verify that this synthetic data maintains the statistical properties of real data while providing coverage for scenarios that might be underrepresented in your original dataset.
5. **Implement data drift detection**. Set up processes to continuously compare the distribution of inference data with your baseline training data. Use this comparison to identify when the real-world data begins to diverge from what the model was trained on, which can signal the need for retraining.
6. **Use Amazon SageMaker AI Model Monitor**. Deploy Model Monitor to automatically track and analyze model behavior in production. Configure alerts for deviations in model quality, data quality, bias drift, and feature attribution drift. SageMaker AI Model Monitor continuously evaluates your deployed models and notifies you when action is needed.
7. **Establish a regular validation cadence**. Schedule periodic evaluations of your model using fresh data, even if drift hasn't been detected. This can catch subtle changes that might not trigger automated alerts but could still affect model performance over time.
8. **Document validation results**. Create detailed reports of validation processes and outcomes for each model version. Include metrics on data representativeness, performance across different data segments, and identified gaps or biases.

Resources

Related documents:

- Evaluating ML Models
- Cross-Validation
- Accelerate generative AI development using managed MLflow on Amazon SageMaker AI
- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- Amazon SageMaker AI Clarify
- Detect data drift using Amazon SageMaker AI Model Monitor

- Monitoring in-production ML models at large scale using Amazon SageMaker AI Model Monitor

Related videos:

- How To Efficiently Manage ML experiments using Amazon SageMaker AI ML Flow
- Detect machine learning (ML) model drift in production

Related examples:

- Amazon SageMaker AI Model Monitor

MLREL03-BP04 Establish data bias detection and mitigation

Detect and mitigate bias to avoid inaccurate model results. Establish bias detection methodologies at data preparation stage before training starts. Monitor, detect, and mitigate bias after the model is in production. Establish feedback loops to track the drift over time and initiate a re-training.

Desired outcome: You can identify and address biases in your machine learning data and models, providing fair and accurate predictions. You have established systematic approaches for detecting bias before training and continuously monitoring bias in production. Your AI systems produce more reliable, fair, and trustworthy results through automated detection and mitigation processes.

Common anti-patterns:

- Waiting until after model deployment to consider bias detection.
- Using a single bias metric for each use case without understanding context-specific requirements.
- Focusing only on training data bias and ignoring bias that may emerge in production.
- Failing to establish feedback mechanisms to monitor and address drift over time.
- Treating bias detection as a one-time activity rather than an ongoing process.

Benefits of establishing this best practice:

- Improves model accuracy and fairness across different demographic groups.
- Reduces risk of deploying models with harmful or discriminatory outcomes.
- Enhances transparency and explainability for model predictions.
- Increases trust from users and stakeholders in AI systems.

- Improves adherence to emerging AI regulations and ethical guidelines.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Bias in machine learning models can lead to unfair or inaccurate outcomes that disproportionately impact certain groups. You need a systematic approach to detect and mitigate bias throughout the machine learning lifecycle. This begins with careful analysis of your training data to identify potential imbalances or historical biases that could be propagated by your models. By implementing bias detection methodologies before training, you can address issues early in the development process.

Once your models are in production, ongoing monitoring is essential as new data patterns may introduce unexpected biases over time. Setting up automated detection systems allows you to continuously evaluate model fairness and take corrective actions when necessary. Building feedback loops provides the data needed to understand how bias manifests in real-world applications and informs model retraining strategies.

Amazon SageMaker AI provides comprehensive tools like SageMaker AI Clarify to implement bias detection and mitigation strategies throughout the ML lifecycle. These tools offer quantitative metrics to measure different types of bias and provide explanations for model predictions to understand and address the root causes of unfairness.

Implementation steps

1. **Understand different types of bias**. Begin by educating your team about various forms of bias that can affect machine learning models, including selection bias, measurement bias, aggregation bias, and evaluation bias. Educate your team members on how bias can be introduced at different stages of the ML lifecycle and the potential impacts on model predictions.
2. **Analyze your training data**. Use Amazon SageMaker AI Clarify to examine your training data for potential bias before model development. Analyze the distribution of sensitive attributes and identify imbalances or correlations that could lead to unfair outcomes. Address data imbalances through techniques like resampling, weighting, or generating synthetic data for underrepresented groups.
3. **Select appropriate bias metrics**. Choose bias metrics that align with your specific use case and fairness requirements. SageMaker AI Clarify provides multiple pre-training bias metrics including

class imbalance, difference in proportions of labels, and conditional demographic disparity. For post-training, metrics like disparate impact, difference in positive proportions across predicted labels, and accuracy difference can evaluate model fairness.

4. **Run SageMaker AI Clarify processing jobs**. Integrate SageMaker AI Clarify processing jobs into your ML pipeline to analyze bias and provide explainability. Configure these jobs to calculate bias metrics on your training data and model predictions, identifying potential issues before deployment.
5. **Implement bias mitigation strategies**. Address identified biases using techniques like preprocessing (modifying training data), in-processing (incorporating fairness constraints during training), or post-processing (adjusting model outputs). Experiment with different approaches and measure their impact on both fairness metrics and model performance.
6. **Set up production monitoring**. Configure Amazon SageMaker AI Model Monitor to continuously track bias metrics on production data. Create alerts that alarm when bias metrics exceed predefined thresholds, enabling prompt investigation and remediation of emerging issues.
7. **Establish feedback loops**. Implement mechanisms to collect and analyze feedback on model predictions, particularly focusing on cases where bias may be present. Use this feedback to improve your understanding of real-world bias patterns and inform model retraining strategies.
8. **Generate model governance reports**. Use SageMaker AI Clarify to create comprehensive reports on model fairness and explainability for stakeholders, including risk and compliance-aligned teams and external regulators. These reports should document your bias detection and mitigation efforts, providing transparency into your responsible AI practices.
9. **Conduct regular model reviews**. Schedule periodic reviews of your models' fairness performance, bringing together cross-functional teams to evaluate bias metrics, examine challenging cases, and decide on necessary interventions or improvements.

Resources

Related documents:

- Run SageMaker AI Clarify Processing Jobs for Bias Analysis and Explainability
- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- Fairness, model explainability and bias detection with SageMaker AI Clarify
- Amazon SageMaker AI Clarify Detects Bias and Increases the Transparency of Machine Learning Models
- Build a secure enterprise machine learning platform on AWS

Related videos:

- Introducing Amazon SageMaker AI Clarify, part 1 - Bias detection
- Introducing Amazon SageMaker AI Clarify, part 2 - Model explainability
- Responsible use of artificial intelligence and machine learning

Related examples:

- Amazon SageMaker AI Clarify: ML Bias Detection and Explainability
- Amazon SageMaker AI Model Monitoring

Deployment

Best practices

- MLREL04-BP01 Automate endpoint changes through a pipeline
- MLREL04-BP02 Use an appropriate deployment and testing strategy

MLREL04-BP01 Automate endpoint changes through a pipeline

Manual change management can be error prone and potentially incur a high effort cost. Use automated pipelines (that integrate with a change management tracking system) to deploy changes to your model endpoints. Versioned pipeline inputs and artifacts allow you to track the changes and automatically rollback after a failed change.

Desired outcome: You establish a reliable and consistent deployment process for your machine learning models. You gain the ability to track changes, perform automatic rollbacks when needed, and improve adherence to change management policies. This approach reduces manual errors, increases operational efficiency, and provides you with better visibility into your ML deployment lifecycle.

Common anti-patterns:

- Making manual updates directly to production endpoints.
- Using different deployment processes across teams or environments.
- Lacking proper version control for model artifacts.
- Not having clear rollback mechanisms for failed deployments.
- Bypassing change management tracking systems.

Benefits of establishing this best practice:

- Reduces human error during deployments.
- Increases deployment speed and reliability.
- Improves traceability and auditability of changes.
- Enables rollback to previous versions.
- Improves adherence to change management policies.
- Supports scalable ML operations across multiple models.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Implementing automated pipelines for model endpoint changes brings consistency and reliability to your ML operations. By using a standardized deployment approach, you can verify that that changes to production endpoints follow the same validated process. This reduces the risk of deployment errors and improves overall operational efficiency.

The pipeline should include steps for testing, validation, and approval of model changes before they reach production. Integration with your existing change management system allows for proper tracking and documentation of changes. The pipeline should also maintain versioned artifacts of models deployed, enabling rollback if issues arise in production.

Implementation steps

1. **Set up a CI/CD pipeline for ML workloads**. Establish a continuous integration and continuous delivery pipeline specifically designed for machine learning workloads. Amazon SageMaker AI Pipelines provides a purpose-built solution for creating end-to-end ML workflows at scale.
2. **Integrate with change management systems**. Connect your pipeline to existing change management tracking systems to document endpoint changes. This improves adherence with your organization's governance requirements and provides a full audit trail of deployment activity.
3. **Implement artifact versioning**. Store model artifacts, code, and configuration files in version control. Use Amazon SageMaker AI Model Registry to catalog and version your models, making it straightforward to track which model versions are deployed to which endpoints.
4. **Define automated testing and validation**. Include automated testing steps in your pipeline to validate model performance before deployment. This should include unit tests, integration tests, and model quality evaluations using metrics specific to your use case.

5. **Establish approval gates**. Configure approval checkpoints in your pipeline where stakeholders can review changes before they are promoted to production. These gates maintain quality control and improve adherence to business requirements.
6. **Implement automated rollback mechanisms**. Create automated procedures to roll back to the previous stable version if a deployment fails or if issues are detected after deployment. This minimizes downtime and impact on users.
7. **Monitor deployment metrics**. Track the success rate of deployments and time-to-deployment metrics to continuously improve your pipeline. Use Amazon CloudWatch to monitor these operational metrics.

Resources

Related documents:

- Pipelines overview
- Model Registration Deployment with Model Registry
- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- Build a CI/CD pipeline for deploying custom machine learning models using AWS services
- Building, automating, managing, and scaling ML workflows using Amazon SageMaker AI Pipelines

Related videos:

- Implementing MLOps practices with Amazon SageMaker AI

Related examples:

- SageMaker AI Pipelines and SageMaker AI Model Registry

MLREL04-BP02 Use an appropriate deployment and testing strategy

Select the right deployment and testing strategy for your machine learning models to create smoother transitions to production, minimize disruption, and allow for careful evaluation of model performance before full implementation.

Desired outcome: You can confidently deploy machine learning models to production using strategies that minimize risk and maximize availability. You have established processes to monitor

model performance, allowing you to make data-driven decisions about when to roll back to previous versions or roll forward with new updates. Your deployment pipelines include appropriate testing, validation, and metrics collection to improve model quality and performance in production environments.

Common anti-patterns:

- Deploying new models directly to production without testing strategies.
- Lacking version control for model artifacts.
- Not implementing monitoring metrics to evaluate model performance.
- Using the same deployment strategy for models without considering specific use case requirements.
- Failing to plan for rollbacks when model performance degrades.

Benefits of establishing this best practice:

- Minimizes disruption to production services during model updates.
- Enables testing models with real production traffic before full deployment.
- Reduces risk through controlled deployment strategies.
- Improves visibility into model performance.
- Provides better mechanisms for recovering from poor-performing model deployments.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Machine learning models require special consideration when deploying to production environments because their behavior can be complex and sometimes unpredictable. Unlike traditional software where functionality is explicitly coded, ML models learn patterns from data, making it crucial to validate their performance with production data before full deployment.

Different deployment strategies provide varying levels of risk mitigation and testing capabilities. Blue/green deployments allow for instantaneous rollback by maintaining two identical environments. Canary deployments reduce risk by exposing only a small portion of traffic to new models. A/B testing enables you to compare performance metrics between model versions. Shadow deployments let you test new models without affecting user experience by running them alongside the production model but not using their predictions.

Your choice of deployment strategy should be guided by your specific requirements for availability, risk tolerance, and the criticality of the ML application. For high-stakes applications like fraud detection or medical diagnostics, more cautious approaches like canary or shadow deployments may be appropriate. For less critical applications, simpler strategies might suffice.

Implementation steps

1. **Perform a deployment strategy trade-off analysis**. Evaluate your business requirements and risk tolerance to determine which deployment strategies are most appropriate for your ML models. Consider factors like required availability, acceptable downtime, criticality of predictions, and monitoring capabilities. Document your analysis and selection criteria for each model deployment.
2. **Implement robust model versioning**. Establish a system to version model artifacts, including the model itself, preprocessing components, and associated configurations. Use Amazon SageMaker AI Model Registry to catalog models and track their lineage, approval status, and deployment history. Then, you can quickly identify what model version is running and roll back if needed.
3. **Set up blue/green deployments with SageMaker AI**. Implement blue/green deployments for your real-time inference endpoints to maximize availability during updates. SageMaker AI automatically provisions new infrastructure (green fleet) before transitioning traffic from the old infrastructure (blue fleet), providing nearly continuous service. Configure the appropriate traffic shifting mode based on your risk tolerance.
4. **Configure canary deployments for higher-risk updates**. For model updates where you want additional safety, implement canary deployments that route a small percentage of traffic to the new model version first. Use SageMaker AI deployment guardrails to set up canary testing with baking periods to monitor model performance before shifting the remaining traffic.
5. **Establish linear traffic shifting for granular control**. For the most controlled deployments, set up linear traffic shifting in SageMaker AI to gradually move traffic from the blue fleet to the green fleet in multiple steps. Define appropriate step sizes and baking periods between shifts to carefully monitor model behavior at each stage.
6. **Implement A/B testing for model comparison**. When you need to validate that a new model performs better than the existing one, set up A/B testing to compare metrics between versions with real production traffic. Use SageMaker AI A/B testing to route defined percentages of traffic to different model variants and collect performance data.
7. **Deploy shadow models to reduce the risk of testing**. For high-risk applications, consider implementing shadow deployments where the new model runs alongside the production model

but doesn't affect customer-facing decisions. This allows you to compare how the new model would have performed using real production data while minimizing the risk.

8. **Define and implement model performance metrics**. Establish clear metrics to evaluate model performance in production, such as prediction accuracy, latency, throughput, and business KPIs. Set up monitoring with Amazon CloudWatch to track these metrics and create alarms that can run automatic or manual interventions when performance degrades.
9. **Create automatic rollback mechanisms**. Implement automated procedures to roll back to previous model versions when performance metrics indicate problems. Define specific thresholds for metrics that would trigger a rollback, and establish the process for rolling back with minimal disruption.
10. **Build comprehensive CI/CD pipelines**. Integrate your deployment strategies into complete CI/CD pipelines that automate the testing, deployment, and monitoring of models. Use AWS CodePipeline and AWS CodeDeploy in conjunction with SageMaker AI to create reliable deployment workflows.

Resources

Related documents:

- Deployment guardrails for updating models in production
- Blue/Green Deployments
- Testing models with production variants
- Model Registration Deployment with Model Registry
- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- Take advantage of advanced deployment strategies using Amazon SageMaker AI deployment guardrails
- A/B Testing ML models in production using Amazon SageMaker AI

Related videos:

- Deliver high-performance ML models faster with MLOps tools
- Canaries in the code mines: Monitoring deployment pipelines

Related examples:

- Amazon SageMaker AI Inference Deployment Guardrails

- Amazon SageMaker AI A/B Testing Pipeline
- Amazon SageMaker AI Safe Deployment Pipeline

Monitoring

Best practices

- MLREL05-BP01 Allow automatic scaling of the model endpoint
- MLREL05-BP02 Create a recoverable endpoint with a managed version control strategy

MLREL05-BP01 Allow automatic scaling of the model endpoint

Implement capabilities that allow the automatic scaling of model endpoints. This improves the reliable processing of predictions to meet changing workload demands. Include monitoring on endpoints to identify a threshold that initiates the addition or removal of resources to support current demand.

Desired outcome: You can efficiently handle varying workload demands by implementing automatic scaling for your model endpoints. Your endpoints dynamically adjust resources based on real-time needs, providing consistent performance and availability without manual intervention. This results in reliable prediction processing, optimal resource utilization, and cost-effective operations.

Common anti-patterns:

- Manually scaling endpoints in response to traffic changes.
- Over-provisioning resources to handle peak loads at non-peak times.
- Neglecting to set up monitoring for endpoint performance.
- Ignoring traffic patterns when configuring scaling policies.
- Using fixed infrastructure that can't adapt to changing workloads.

Benefits of establishing this best practice:

- Improves reliability and availability of prediction services.
- Optimizes costs through dynamic resource allocation.
- Enhances user experience with consistent response times.
- Reduces operational overhead through automation.

- Strengthens ability to handle unexpected traffic spikes.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Automatic scaling of model endpoints is critical for maintaining reliable machine learning services in production. By implementing auto scaling, your endpoints can handle varying loads efficiently without manual intervention. This capability is especially important for applications with fluctuating traffic patterns or those that experience periodic spikes in demand.

When setting up automatic scaling, you need to consider appropriate metrics that trigger scaling actions, such as CPU utilization, memory usage, or request latency. Define appropriate thresholds for these metrics so that your system scale at the right time - not too early (which wastes resources) or too late (which impacts performance).

Monitoring is an essential component of an auto scaling solution. By implementing comprehensive monitoring, you gain visibility into endpoint performance and scaling operations, allowing you to optimize your configuration over time based on real usage patterns.

Implementation steps

1. **Configure automatic scaling for Amazon SageMaker AI endpoints**. Amazon SageMaker AI supports automatic scaling (auto scaling) for your hosted models. SageMaker AI endpoints can be configured with auto scaling to maintain service availability as traffic increases. Automatic scaling automatically provisions new resources horizontally to handle increased user demand or system load.
2. **Set up appropriate scaling policies**. Define target metrics for scaling such as CPU utilization, memory usage, or request count. Configure appropriate minimum and maximum instance counts based on your expected traffic patterns and performance requirements. Consider implementing both scale-out policies (adding capacity when load increases) and scale-in policies (removing capacity when load decreases) to optimize resource utilization.
3. **Implement comprehensive monitoring**. Use Amazon CloudWatch to monitor the performance of your endpoint and collect metrics that can inform scaling decisions. Create dashboards to visualize endpoint performance and scaling activities. Set up alerts to notify you of issues or anomalies with your endpoints.
4. **Leverage SageMaker AI Serverless Inference**. For workloads with intermittent or unpredictable traffic patterns, consider using Amazon SageMaker AI Serverless Inference, which automatically

scales compute capacity up and down based on traffic, avoiding the need to choose instance types or manage scaling policies.

5. **Utilize SageMaker AI Inference Recommender**. Before deploying models to production, use Amazon SageMaker AI Inference Recommender to get recommendations on instance types and configurations that will best meet your performance and cost requirements, assisting you in optimizing your scaling policies.
6. **Implement load testing**. Perform load testing on your endpoints to understand how they behave under different traffic conditions. This information can fine-tune your scaling policies so that they're effective when real traffic increases occur.

Resources

Related documents:

- Automatic scaling of Amazon SageMaker AI models
- Deploy models with Amazon SageMaker AI Serverless Inference
- Amazon SageMaker AI Inference Recommender
- Configuring autoscaling inference endpoints in Amazon SageMaker AI

MLREL05-BP02 Create a recoverable endpoint with a managed version control strategy

Establish a fully recoverable system for model prediction endpoints by implementing proper version control and lineage tracking for components that generate these endpoints.

Desired outcome: You have a robust infrastructure where components related to model deployment, including model artifacts, container images, and endpoint configurations, are version controlled and traceable. You can recover quickly from issues by identifying and reverting to previous stable versions, and you can audit the full lineage of model deployments for governance and regulatory requirements.

Common anti-patterns:

- Storing model artifacts without proper versioning.
- Using deployment processes only when needed without infrastructure as code.
- Failing to track dependencies between model artifacts, containers, and configurations.
- Not maintaining a centralized registry for models.

- Relying on manual processes for endpoint recovery.

Benefits of establishing this best practice:

- Reduces recovery time when endpoint issues occur.
- Improves auditability and adherence through comprehensive lineage tracking.
- Enhances collaboration between data scientists and operations teams.
- Improves the consistency and reliability of model deployments.
- Enables reproduction of model endpoints exactly as they were at specific points in time.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Machine learning model endpoints represent the interface where your business delivers value from AI/ML investments. Verifying that these endpoints are recoverable is critical for business continuity. Recoverability depends on having comprehensive version control for components involved in creating the endpoint.

A properly implemented MLOps framework for endpoint recoverability tracks not just the model artifacts, but components that influence the prediction service—training data, feature transformations, container definitions, and infrastructure configurations. When incidents occur, you need to understand the complete lineage of your model endpoint, including which data trained the model, which code created the container, and which configurations defined the infrastructure.

By implementing proper version control and lineage tracking, you create a system that is both resilient to failures and compatible with governance requirements. You can trace exactly how each endpoint was created and recreate it precisely when needed.

Implementation steps

1. **Implement MLOps with Amazon SageMaker AI Pipelines and Projects**. Amazon SageMaker AI Pipelines automates ML workflows by providing a service for building, running, and managing ML pipelines. It handles every step from data preparation to model deployment in a versioned, predictable manner.
2. **Implement a model registry system**. Use Amazon SageMaker AI Model Registry to catalog your models for production. The registry tracks model versions and their approval status, creating a system of record for models in your organization. Define a clear approval workflow for moving

models from development to production for proper governance at each stage. For each model, register metadata including performance metrics, training datasets, and intended use cases.

3. **Track experiments with SageMaker AI MLflow**. MLflow in SageMaker AI allows you to create, manage, analyze, and compare experiments. This way, data scientists can view and track the experiments for the current project. Each experiment logs every metric and hyperparameter automatically, along with model artifacts and dataset information.
4. **Use infrastructure as code (IaC) tools**. Define and build your infrastructure, including model endpoints, using AWS CloudFormation or AWS CDK. IaC makes your infrastructure version controlled, repeatable, and able to be reverted to previous states if needed. Store your infrastructure code in git repositories alongside your model code, creating a unified version history. This approach reduces configuration drift between environments and verifies that your production deployment exactly matches your tested configuration.
5. **Store containers in Amazon Elastic Container Registry**. Use Amazon ECR to version and store the Docker containers that serve your models. Amazon ECR automatically creates version hashes for containers as you update them, enabling rollbacks to previous versions. Implement image scanning to detect security vulnerabilities, and apply lifecycle policies to manage older versions of your containers.
6. **Implement automated testing and deployment pipelines**. Create CI/CD pipelines using AWS CodePipeline to automate the testing and deployment of your models. These pipelines should validate models before deployment, deploy infrastructure changes through CloudFormation, and update model endpoints with minimal downtime. Integrate automated quality checks to avoid problematic models from reaching production.
7. **Configure automated backups and recovery processes**. Establish automated backup procedures for your model artifacts, container images, and endpoint configurations. Use Amazon S3 versioning for model artifacts and AWS Backup to protect configuration data. Document and test recovery procedures regularly to verify that they work when needed.

Resources

Related documents:

- AWS CloudFormation Documentation
- Infrastructure as code
- Pipelines overview
- What is Amazon Elastic Container Registry?
- Model Registration Deployment with Model Registry

- Fully managed MLflow 3.0 on Amazon SageMaker AI
- Building, automating, managing, and scaling ML workflows using Amazon SageMaker AI Pipelines
- Multi-account model deployment with Amazon SageMaker AI Pipelines
- Automate feature engineering pipelines with Amazon SageMaker AI

Related videos:

- Next-generation CDK development with Amazon Q Developer
- Infrastructure as Code on AWS - AWS Online Tech Talks
- How to create fully automated ML workflows with Amazon SageMaker AI Pipelines

Related examples:

- Amazon SageMaker AI MLOps

Performance efficiency

The performance efficiency pillar focuses on the efficient use of computing resources to meet requirements and the maintenance of that efficiency as demand changes and technologies evolve.

Each best practice in this section is presented based on its place in the ML lifecycle as detailed in *Machine learning lifecycle*.

Lifecycle phases

- Business goal identification
- ML problem framing
- Data processing
- Model development
- Deployment
- Monitoring

Business goal identification

Best practices

- MLPERF01-BP01 Determine key performance indicators

MLPERF01-BP01 Determine key performance indicators

Use guidance from business stakeholders to capture key performance indicators (KPIs) relevant to the business use case. The KPIs should be directly linked to business value to guide acceptable model performance. Consider that machine learning inferences are probabilistic and will not provide exact results. Identify a minimum acceptable accuracy and maximum acceptable error in the KPIs. This enables you to achieve the required business value and manage the risk of variable results.

Desired outcome: By defining direct, measurable KPIs, ML initiatives deliver quantifiable business outcomes, such as cost savings, expanded scale, and faster response times. Clear performance thresholds set realistic stakeholder expectations and enable risk management based on the probabilistic nature of ML.

Common anti-patterns:

- Implementing ML solutions without defining clear business-oriented success metrics.
- Focusing solely on technical metrics (like model accuracy) without connecting them to business outcomes.
- Setting unrealistic expectations for ML performance without accounting for probabilistic results.
- Failing to define acceptable error thresholds for critical business processes.
- Neglecting to quantify the actual business value of ML implementations.

Benefits of establishing this best practice:

- Aligns machine learning (ML) outcomes with business objectives for measurable value.
- Creates clear expectations about model performance that account for ML's probabilistic nature.
- Enables objective evaluation of ML solution success based on business impact.
- Improves prioritization of ML investments based on tangible results.
- Accelerates decision-making by translating ML insights into business actions.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Start by identifying business challenges ML aims to solve and how success translates into specific, quantifiable benefits. Engage stakeholders throughout KPI selection to verify that business priorities drive metric design. Use metrics that reflect business value—such as cost reduction, customer retention rate, or time savings—rather than technical measures alone.

Regularly review KPIs to stay aligned with strategic shifts. Feedback from business results informs necessary adjustments to both models and evaluation metrics. Common pitfalls include proceeding without clear business KPIs, focusing only on technical metrics such as accuracy, or establishing unrealistic expectations that ignore the probabilistic nature of ML results. Failing to set acceptable error thresholds exposes critical business processes to unmanaged risk, and overlooking the business value of ML adoption makes it hard to measure impact or secure stakeholder support.

Implementation steps

1. **Quantify the value of machine learning for the business**. Consider measures of how machine learning and automation will impact the business:
   - How much will machine learning reduce costs?

- How many more users will be reached by increasing scale?
- How much time will the business save by being able to respond faster to changes, such as in demand and supply disruptions?
- How many hours of manual effort will be reduced by automating with machine learning?
- How much will machine learning be able to change user behavior, such as reducing churn?

2. **Evaluate risks and the tolerance for error**. Quantify the impact of machine learning on the business. Rank order the value of impacts to identify the primary KPIs to optimize with machine learning. Define the cost of error for automated inferences that will be performed by ML models in the use case. Determine the tolerance of the business for error. For example, determine how far off a cost reduction estimate would have to be to negatively impact the business goals. Finally, evaluate the risks of machine learning for the business, and whether the benefits of ML solutions are of high enough value to outweigh those risks.
3. **Establish baseline metrics**. Before implementing ML solutions, document current performance metrics to create a baseline against which to measure improvements. Collect data on existing processes, including costs, time requirements, error rates, and other relevant performance indicators. This baseline will serve as a reference point for demonstrating the business value of your ML implementation.
4. **Define predictive and prescriptive KPIs**. Move beyond retrospective metrics to develop KPIs that offer predictive and prescriptive insights. Use Amazon CloudWatch and Quick to create dashboards that visualize these forward-looking KPIs, making them accessible to business stakeholders.
5. **Create a KPI governance framework**. Develop a structured approach for monitoring, reviewing, and refining your KPIs over time. Gather executive alignment on metrics, establish consistent data collection processes, and define protocols for taking corrective actions when negative trends emerge. Regularly analyze trends and periodically refine KPIs to accurately gauge the business impact of ML implementations.
6. **Leverage advanced analytics for insights**. Enhance KPI discovery and accessibility by integrating advanced analytics services, such as Amazon Q. These tools uncover hidden business patterns and translate complex results into conversational analytics for non-technical audiences.

Resources

Related documents:

- Improve Business Outcomes with Machine Learning
- AWS Well-Architected Framework

- Machine Learning (ML) Governance with Amazon SageMaker AI
- Amazon Q for Business Analytics
- AI/ML | AWS Executive Insights
- Thought Leadership | Artificial Intelligence - AWS
- Keys to maximizing AI value

Related videos:

- How to Drive Business Value with AI/ML
- Creating Business Value with AWS AI/ML

ML problem framing

Best practices

- MLPERF02-BP01 Define relevant evaluation metrics
- MLPERF02-BP02 Use purpose-built AI and ML services and resources

MLPERF02-BP01 Define relevant evaluation metrics

Establishing clear, meaningful evaluation metrics is essential for validating machine learning model performance against business objectives. By selecting metrics that directly relate to your key performance indicators (KPIs), you can verify that your ML solutions deliver measurable business value.

Desired outcome: You have a comprehensive set of evaluation metrics that accurately reflect your business requirements and tolerance for errors. These metrics enable you to tune your models directly to business objectives, monitor performance in production, and make data-driven decisions about model improvements.

Common anti-patterns:

- Using the same generic metrics for each model type regardless of business context.
- Focusing only on technical metrics without considering business impact.
- Overlooking the cost implications of different types of errors (false positives and false negatives).
- Failing to establish baseline performance metrics before deployment.
- Neglecting continuous monitoring of metrics after model deployment.

Benefits of establishing this best practice:

- Alignment of ML models with business goals and objectives.
- Better decision-making through quantifiable performance measurement.
- Early detection of model degradation or concept drift.
- Improved ROI from ML investments.
- Clearer communication between technical teams and business stakeholders.

Level of risk exposed if this best practice is not established: High

Implementation guidance

When developing machine learning solutions, establish evaluation metrics that directly connect to your business objectives. These metrics must reflect how well your model performs in the context of your specific use case rather than relying solely on generic technical measures.

Avoid focusing only on technical metrics without considering business impact. Many organizations use the same metrics for each model type regardless of business context, overlook the cost implications of different types of errors, fail to establish baseline performance metrics before deployment, and neglect continuous monitoring after deployment.

For example, in a predictive maintenance scenario, the business impact of false positives (unnecessarily replacing functioning equipment) differs from false negatives (missing actual failures). Understand these business implications to select appropriate metrics like precision (minimizing false positives) or recall (minimizing false negatives) based on which error type is more costly to your business.

Different ML problem types require different evaluation approaches. Classification models benefit from confusion matrices that break down performance by class, while regression models need error measurements that quantify prediction deviations. Custom metrics can be developed when standard metrics don't adequately capture business requirements.

Continuous monitoring of these metrics in production is crucial for detecting model drift and improving ongoing performance. Setting up automated alerts when metrics fall below thresholds allows for timely intervention and model updates.

Implementation steps

1. **Align metrics to business objectives**. Begin by clearly understanding the KPIs established during the business goal identification phase. Determine how ML model performance directly

impacts these KPIs and identify which types of errors are most costly to the business. For example, in fraud detection, false negatives (missed fraudulent transactions) may be more costly than false positives.

2. **Select appropriate evaluation metrics**. Choose metrics based on your ML problem type:
   - **For classification problems:** Implement confusion matrix derivatives (precision, recall, accuracy, F1 score), AUC, or log-loss as appropriate for your use case
   - **For regression problems:** Utilize RMSE, MAPE, or other error measures that align with business sensitivity to prediction errors
   - **For recommendation systems:** Consider metrics like Normalized Discounted Cumulative Gain (NDCG) or precision@k
   - **For time series forecasting:** Apply metrics like Mean Absolute Scaled Error (MASE) or symmetric Mean Absolute Percentage Error (sMAPE)
3. **Develop custom metrics if needed**. When standard metrics don't adequately capture business requirements, create custom evaluation metrics that better reflect the business objectives. Use Amazon SageMaker AI to implement these custom metrics during model training and evaluation.
4. **Establish performance thresholds**. Calculate the maximum acceptable error probability required for the ML model based on business tolerance levels. Document these thresholds as acceptance criteria for model deployment.
5. **Implement comparative experimentation**. Use Amazon SageMaker AI Managed MLFlow 3.0 to organize, track, and compare different models trained with various hyperparameters and approaches. The enhanced MLFlow integration provides robust experiment management at scale for complex ML projects. This structured experimentation identifies models that optimize your selected metrics within acceptable bounds.
6. **Monitor metrics in production**. Deploy Amazon SageMaker AI Model Monitor to track model and concept drift in real time. Configure alerts when metrics deviate from expected performance thresholds, enabling prompt remediation actions.
7. **Incorporate feedback loops**. Establish mechanisms to collect real-world performance data and incorporate it into your evaluation process. This feedback assists you to refine metrics and models over time to better align with evolving business needs.
8. **Balance competing metrics**. When multiple metrics are relevant, establish a weighting system that reflects their relative importance to business outcomes. Document this decision-making framework for consistency in model evaluation.
9. **Implement bias detection and model explainability**. Use Amazon SageMaker AI Clarify to detect bias in your models and provide explanations for model predictions. Your evaluation

framework should include fairness and interpretability considerations alongside performance metrics.

10 **Establish automated model evaluation pipelines**. Create automated evaluation workflows that run consistently across different model versions and training iterations. Use SageMaker AI Processing to standardize your evaluation processes and provide reproducible results.

Resources

Related documents:

- Amazon CloudWatch Metrics for Monitoring and Analyzing Training Jobs
- Accelerate generative AI development using managed MLflow on Amazon SageMaker AI
- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- Fairness, model explainability and bias detection with SageMaker AI Clarify
- Evaluate, explain, and detect bias in models
- Data transformation workloads with SageMaker AI Processing

Related videos:

- Organize, Track, and Evaluate ML Training Runs with Amazon SageMaker AI Managed MLFlow
- Foundation model evaluation with SageMaker AI Clarify
- How to efficiently manage ML and Gen AI experiments

Related examples:

- Scikit-Learn Data Processing and Model Evaluation
- Amazon SageMaker AI Model Monitor Examples

MLPERF02-BP02 Use purpose-built AI and ML services and resources

Consider how the workload could be handled by pre-built AI services or ML resources. Better performance can often be delivered more efficiently by using pre-optimized components included in AI and ML managed services. Select an optimal mix of bespoke and pre-built components to meet the workload requirements.

Desired outcome: You achieve a balanced approach to your machine learning workloads by implementing purpose-built AI and ML services and resources. You leverage pre-built components

where appropriate to accelerate development, reduce management overhead, and improve performance while maintaining the flexibility to create custom solutions where your business needs demand it. This approach optimizes both your team's productivity and the overall effectiveness of your AI/ML solutions.

Common anti-patterns:

- Building ML components from scratch when suitable pre-built solutions exist.
- Failing to evaluate the full range of AWS AI and ML services before starting development.
- Over-customizing solutions when standard services would adequately meet requirements.
- Underutilizing AWS marketplace solutions and pre-trained models.
- Not considering hybrid approaches that combine managed services with custom ML models.

Benefits of establishing this best practice:

- Accelerated time-to-market for ML solutions.
- Reduced operational overhead for maintaining ML infrastructure.
- Lower development costs through leveraging pre-built components.
- Access to continuously improved and updated AI technologies.
- Ability to focus resources on high-value business differentiators.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Implement purpose-built AI and ML services to focus on business outcomes rather than infrastructure management. AWS provides a comprehensive portfolio of AI services, ranging from ready-to-use APIs to fully customizable ML solutions. Each service addresses different levels of complexity and customization requirements.

When evaluating your ML workloads, assess which components could benefit from managed services. Tasks like image classification, regression, clustering, or time series forecasting can often be accomplished with SageMaker AI built-in algorithms without requiring custom algorithm development. For more specialized needs, you can leverage pre-trained models through services like Amazon SageMaker AI JumpStart or develop custom models using Amazon SageMaker AI.

Resist the temptation to over-customize solutions when standard services would adequately meet requirements. Organizations often underutilize AWS marketplace solutions and pre-trained

models, missing opportunities to accelerate development. The key is finding the right balance between using managed services for common ML tasks and building custom solutions for your unique business requirements. Consider hybrid approaches that combine managed services with custom ML models rather than pursuing an all-or-nothing strategy.

Consider your team's capabilities when making these decisions. If you lack specialized ML expertise, starting with fully managed AI services provides immediate value while your team builds skills. As your team's capabilities grow, you can selectively add custom components where they provide strategic advantage.

Implementation steps

1. **Assess your ML use cases and requirements**. Begin by clearly defining your business use cases and understanding the ML capabilities needed. Evaluate whether your requirements can be met by pre-built services or require custom development. Consider factors like accuracy requirements, latency needs, and the availability of training data.
2. **Learn about AWS managed AI services**. Determine whether AWS managed AI services are applicable to the business use case. Understand how managed AWS AI services can relieve the burden of training and maintaining an ML pipeline. Use Amazon SageMaker AI to develop in the cloud and understand the roles and responsibilities needed to maintain the ML workload. Consider combining managed AI services with custom ML models built on Amazon SageMaker AI.
3. **Explore SageMaker AI built-in algorithms and automated ML capabilities**. Learn about SageMaker AI built-in algorithms for supervised learning tasks like classification, regression, and forecasting. Consider SageMaker AI Autopilot for automated machine learning that handles data analysis, feature engineering, algorithm selection, and hyperparameter tuning. Explore Amazon SageMaker AI JumpStart for pre-trained models across various ML domains, including foundation models that can be fine-tuned for your specific ML tasks. For enterprise environments, implement SageMaker AI JumpStart Private Model Hubs to create curated repositories of both prebuilt and custom models with centralized governance and version management.
4. **Investigate marketplace solutions**. Learn about SageMaker AI Algorithms and Models in AWS Marketplace, a curated digital catalog that makes it simple for you to find, buy, deploy, and manage third-party software and services. Explore specialized algorithms or pre-trained models that might be relevant to your use case.
5. **Implement a hybrid approach where appropriate**. Design your ML architecture to leverage the most suitable services for each component. Use AWS managed services for standard ML tasks

and focus custom development on business differentiators. This balanced approach optimizes both development efficiency and solution effectiveness.

6. **Establish a model evaluation framework**. Create a systematic process for evaluating pre-built models against your requirements. Define clear metrics for accuracy, latency, cost, and other relevant factors. Use this framework to make data-driven decisions about which components to build versus buy.
7. **Plan for operational integration**. Verify that your chosen ML services can integrate effectively with your existing systems and workflows. Design appropriate data pipelines, APIs, and monitoring systems to support your hybrid ML architecture. For development flexibility, leverage remote IDE connectivity to securely connect third-party developer environments such as VS Code to SageMaker AI Studio, enabling professional MLOps workflows while maintaining centralized governance. Consider security, regulatory adherence, and governance requirements when implementing these integrations.
8. **Optimize model performance and deployment**. Use SageMaker AI model optimization capabilities including quantization, compilation, and speculative decoding to improve inference performance. Use SageMaker AI Inference Recommender to automatically benchmark and select optimal instance types, configurations, and parameters for your inference endpoints. Deploy using SageMaker AI deployment options such as real-time hosting, serverless inference, or batch transform based on your latency and throughput requirements.
9. **Implement model monitoring and governance**. Establish monitoring for model performance, data drift, and model drift using SageMaker AI Model Monitor. Implement proper model versioning, A/B testing capabilities, and rollback procedures to maintain model quality and reliability in production environments.

Resources

Related documents:

- Built-in algorithms and pretrained models in Amazon SageMaker AI
- SageMaker AI JumpStart Foundation Models
- Private curated hubs for foundation model access control in JumpStart
- Best practices for deploying models on SageMaker AI Hosting Services
- Inference optimization for Amazon SageMaker AI models
- Amazon SageMaker AI Inference Recommender
- Model deployment options in Amazon SageMaker AI

- Distributed training optimization
- SageMaker AI JumpStart pretrained models
- Machine Learning on AWS
- Architecture Best Practices for Machine Learning
- Data and model quality monitoring with SageMaker AI Model Monitor
- SageMaker AI Algorithms and Models in AWS Marketplace
- Docker containers for training and deploying models
- Train a Model with Amazon SageMaker AI

Related videos:

- Building and Deploying ML Models Fast with Amazon SageMaker AI JumpStart

Data processing

Best practices

- MLPERF03-BP01 Use a modern data architecture

MLPERF03-BP01 Use a modern data architecture

Get the best insights from exponentially growing data using a modern data architecture. This architecture enables movement of data between a data lake and purpose-built stores including a data warehouse, relational databases, non-relational databases, ML and big data processing, and log analytics. A data lake provides a single place to run analytics across mixed data structures collected from disparate sources. Purpose-built analytics services provide the speed required for specific use cases like real-time dashboards and log analytics.

Desired outcome: You implement a modern data architecture that enables seamless data movement between storage systems, provides unified governance, and supports diverse ML workloads. This architecture accelerates data preparation, improves data quality, and enables efficient feature engineering for machine learning models.

Common anti-patterns:

- Creating isolated data silos where different teams manage separate data stores without coordination.

- Building data architecture without establishing proper governance, access controls, or compliance-aligned frameworks.
- Using one-size-fits-all storage solutions without considering specific workload requirements.
- Relying on manual processes for data movement, transformation, and quality checks.
- Neglecting to implement proper data cataloging and discovery mechanisms.

Benefits of establishing this best practice:

- Unified data governance and access control across data stores.
- Reduced data preparation time through integrated data services.
- Improved data quality and consistency for ML model training.
- Enhanced scalability for growing data volumes and ML workloads.
- Better cost optimization through purpose-built storage solutions.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

When building machine learning solutions, you need a modern data architecture that can handle diverse data types, support various ML workloads, and provide unified governance across data stores. This architecture enables efficient data movement between storage systems while maintaining security, quality, and cost optimization.

Avoid creating isolated data silos where different teams manage separate data stores without coordination. Many organizations struggle with inconsistent data governance policies across different storage systems, lack unified access controls for ML teams, fail to implement proper data cataloging and discovery mechanisms, and neglect to optimize storage costs based on access patterns. These issues create bottlenecks in ML workflows and increase operational complexity.

For example, in a retail ML use case, you might need to combine customer transaction data from a data warehouse, real-time clickstream data from streaming services, and product catalog information from operational databases. A modern data architecture enables seamless access to these data sources while maintaining consistent security policies and enabling efficient feature engineering for recommendation models.

Different ML workloads require different data access patterns. Batch training jobs benefit from optimized data lake storage with efficient querying capabilities, while real-time inference requires low-latency access to feature stores and streaming data pipelines. Custom data processing

workflows can be developed when standard ETL processes don't adequately support your specific ML requirements.

Continuous monitoring of data quality and pipeline performance is crucial for maintaining reliable ML systems. Setting up automated data quality checks and pipeline monitoring allows for early detection of data issues that could impact model performance.

Implementation steps

1. **Design your data lake foundation**. Begin by establishing a centralized data lake using Amazon S3 as the primary storage layer. Organize data using a logical structure that supports both current and future ML use cases, such as organizing by business domain, data source, and processing stage. Implement data partitioning strategies based on common query patterns to optimize performance and reduce costs. For example, partition time-series data by date and customer data by region to enable efficient querying for ML feature extraction.
2. **Implement unified data governance**. Use AWS Lake Formation to build a scalable and secure data lake with centralized governance. Establish consistent security policies, access controls, and audit trails across data stores. Apply fine-grained permissions that enable self-service access for ML practitioners while maintaining security and and addressing requirements. Create data stewardship roles and processes to improve ongoing data quality and governance.
3. **Integrate purpose-built analytics services**. Build a high-speed analytic layer with purpose-built services selected based on your specific ML workload requirements:
   - Use Amazon Redshift for data warehousing and complex analytical queries
   - Implement Amazon Kinesis for real-time streaming data processing
   - Deploy Amazon Athena for interactive queries and ad-hoc analysis
   - Consider Amazon DynamoDB for high-performance operational workloads
   - Use Amazon Timestream for time-series data applications
4. **Enable seamless data integration**. Use AWS Glue to integrate data across services and data stores. Implement automated data cataloging to maintain metadata and enable data discovery across your organization. Create ETL pipelines that prepare data for ML workloads while maintaining data lineage and enabling reproducibility. Design workflows that can handle both batch and streaming data processing requirements.
5. **Optimize for ML workloads**. Design data pipelines that support both batch and real-time ML training scenarios. Implement feature stores using services like Amazon SageMaker AI Feature Store to manage and share ML features across teams and models. Create standardized feature engineering processes that can be reused across different ML projects and provide consistent data transformations.

6. **Establish data quality monitoring**. Implement automated data quality checks and monitoring for data reliability for ML models. Use AWS Glue DataBrew for data profiling and quality assessment. Set up automated alerts for data quality issues such as missing values, schema changes, or statistical anomalies that could impact ML model performance.
7. **Implement cost optimization strategies**. Use appropriate storage classes in Amazon S3 based on data access patterns. Implement lifecycle policies to automatically transition data to lower-cost storage tiers such as S3 Infrequent Access or Amazon Glacier for archival data. Monitor and optimize query performance to control compute costs, and use reserved capacity where appropriate for predictable workloads.
8. **Enable real-time data processing**. For ML use cases requiring real-time inference, implement streaming data pipelines using Amazon Kinesis and AWS Lambda to process data as it arrives and update feature stores in near real-time. Design architectures that can handle varying data volumes and provide consistent low-latency access to features for real-time ML predictions.
9. **Implement data lineage and versioning**. Establish comprehensive data lineage tracking to understand data flow from source to ML models. Use versioning for both datasets and feature definitions to enable reproducible ML experiments and model rollbacks when necessary. This is crucial for improving regulatory adherence and debugging ML model issues.
10. **Create self-service data access**. Build data catalogs and discovery tools that enable ML practitioners to find and access relevant data without requiring deep technical knowledge of the underlying storage systems. Implement standardized APIs and interfaces that abstract the complexity of the data architecture while providing the flexibility needed for diverse ML workloads.

Resources

Related documents:

- The lakehouse architecture of Amazon SageMaker AI
- Amazon SageMaker AI Unified Studio
- What is AWS Lake Formation?
- AWS Lake Formation: How it works
- Create, store, and share features with Feature Store
- What is AWS Glue?
- Modern Data Architecture on AWS
- Amazon SageMaker AI Lakehouse | AWS Big Data Blogmoving-from-notebooks-to-automated-ml-pipelines-using-amazon-sagemaker-and-aws-glue/)

- Amazon SageMaker AI | AWS Big Data Blog

Related videos:

- Understanding Amazon S3 Tables: Architecture, performance, and integration
- Accelerate your Analytics and AI with Amazon SageMaker AI LakeHouse
- Unifying data governance with Immuta and AWS Lake Formation
- The lake house approach to data warehousing with Amazon Redshift

Related services:

- AWS Glue DataBrew
- Amazon Kinesis Data Streams
- Amazon Athena

Model development

Best practices

- MLPERF04-BP01 Optimize training and inference instance types
- MLPERF04-BP02 Explore alternatives for performance improvement
- MLPERF04-BP03 Establish a model performance evaluation pipeline
- MLPERF04-BP04 Establish feature statistics
- MLPERF04-BP05 Perform a performance trade-off analysis
- MLPERF04-BP06 Detect performance issues when using transfer learning

MLPERF04-BP01 Optimize training and inference instance types

Selecting appropriate instance types for training and inference workloads provides optimal performance, reduced costs, and faster time-to-market for your machine learning models. By understanding your model's specific requirements and data characteristics, you can choose the right computational resources to maximize efficiency.

Desired outcome: You achieve optimal performance and cost-effectiveness for your machine learning workloads by selecting appropriate instance types for both training and inference. You

understand how model complexity and data characteristics influence hardware decisions, enabling you to accelerate model development, improve inference speeds, and manage resources efficiently.

Common anti-patterns:

- Using the same instance type for both training and inference workloads.
- Overprovisioning resources just to be safe without performance testing.
- Selecting expensive GPU instances for inference when CPU instances would suffice.
- Ignoring model-specific hardware requirements when selecting instances.
- Not scaling training across multiple instances for large datasets.

Benefits of establishing this best practice:

- Reduced training time and faster model iterations.
- Lower operational costs through right-sized resources.
- Improved inference latency and throughput.
- Better utilization of available computational resources.
- Enhanced scalability for varying workloads.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Understanding how your model type and data characteristics influence instance selection is essential for optimizing machine learning workloads. For training, the computational requirements depend largely on the model complexity, dataset size, and training approach. Deep learning models, particularly those processing image, video, or language data, often benefit from GPU-accelerated instances due to their parallel processing capabilities. Meanwhile, traditional machine learning algorithms may be efficiently trained on CPU instances.

For inference, requirements vary based on deployment scenarios. Real-time applications with strict latency requirements might need powerful compute-optimized instances, while batch prediction workloads can use more cost-effective options. Generally, CPUs are sufficient for many inference scenarios, though complex models may still benefit from GPU acceleration.

When evaluating instance options, consider memory requirements (especially for large models or datasets), network performance for distributed training, and storage I/O capabilities when working with large datasets. The right balance between performance and cost is key to sustainable machine learning operations.

Implementation steps

1. **Analyze your model and data requirements**. Begin by understanding the computational needs of your machine learning algorithm. Assess memory requirements, model complexity, and dataset size. For deep learning models processing image, video, or language data, GPU instances like P4, G4, or P3 typically offer the best performance. For traditional ML algorithms, CPU instances may be more cost-effective.
2. **Benchmark different instance types for training**. Run small-scale training jobs across various instance types in Amazon SageMaker AI to measure performance and cost metrics. Compare training times, resource utilization, and overall costs to identify the optimal instance type for your model. Track Experiments with Managed MLFlow to track and compare results.
3. **Implement distributed training for large datasets**. For large datasets or complex models, leverage distributed training across multiple instances to reduce training time. Use SageMaker AI distributed training libraries to automatically partition data and optimize communication between nodes, which accelerates training for deep learning models.
4. **Optimize storage configuration for I/O performance**. Configure fast storage options to avoid I/O bottlenecks during training. Consider using Amazon FSx for Lustre for high-performance file systems or optimize your data pipeline to use Amazon S3 efficiently. Proper data formatting and efficient loading strategies can improve GPU utilization.
5. **Select appropriate inference instance types**. Evaluate latency and throughput requirements for your inference needs. For real-time inference with strict latency requirements, consider compute-optimized instances or GPU-accelerated instances for complex models. For batch inference, less expensive CPU instances often suffice. Use Amazon SageMaker AI Inference Recommender to get automated recommendations for optimal deployment configurations.
6. **Monitor and optimize costs**. Implement continuous monitoring of resource utilization and costs. Use AWS Cost Explorer and SageMaker AI Studio resource monitoring to identify inefficiencies. Consider using Amazon SageMaker AI Savings Plans for frequently used instance types to reduce costs.
7. **Consider model optimization techniques**. Implement model optimization techniques like quantization, pruning, or knowledge distillation to reduce computational requirements for both training and inference. Explore using SageMaker AI Neo to automatically optimize models for target hardware.
8. **Explore serverless inference options**. For variable or unpredictable workloads, consider Amazon SageMaker AI Serverless Inference to automatically scale resources based on traffic and avoid the need to choose instance types manually.

9. **Leverage specialized ML hardware**. For large-scale training and inference workloads, consider AWS Trainium instances for training and AWS Inferentia instances for inference to achieve better price-performance ratios compared to traditional GPU instances.

Resources

Related documents:

- Train a Model with Amazon SageMaker AI
- Deploy models for inference
- Model performance optimization with SageMaker AI Neo
- Amazon SageMaker AI Inference Recommender
- Deploy models with Amazon SageMaker AI Serverless Inference
- Recommended Trainium Instances
- What are AWS Deep Learning Containers?
- Learn how to select ML instances on the fly in Amazon SageMaker AI Studio
- Ensure efficient compute resources on Amazon SageMaker AI

Related videos:

- How to choose the right instance type for ML inference
- The right instance type in Amazon SageMaker AI

Related examples:

- Amazon SageMaker AI End-to-End Example
- SageMaker AI Inference Recommender Examples

MLPERF04-BP02 Explore alternatives for performance improvement

Benchmarking your machine learning models allows you to systematically improve performance by evaluating and comparing different algorithms, features, and architectural resources. Use this practice to identify the optimal combination and achieve your desired performance metrics.

Desired outcome: You implement a systematic approach to improving your machine learning model's performance through benchmarking various techniques. You'll establish a baseline model and methodically explore alternatives including data volume increases, feature engineering,

algorithm selection, ensemble methods, and hyperparameter tuning. This results in optimized models that provide higher accuracy and better business value.

Common anti-patterns:

- Selecting a complex algorithm without establishing a baseline.
- Ignoring feature engineering in favor of only trying different algorithms.
- Using more data without understanding its quality or relevance.
- Focusing exclusively on accuracy while ignoring other important metrics.
- Manually testing hyperparameters without a systematic approach.

Benefits of establishing this best practice:

- Improved model accuracy and performance.
- Better understanding of which factors most influence model performance.
- More efficient use of computational resources.
- Systematic approach to model improvement instead of random experimentation.
- Ability to document and reproduce experiments for future reference.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Performance improvement in machine learning requires a structured, iterative approach. Benchmarking assists you in systematically comparing different approaches and determining the most effective path to improved model performance. Start by establishing a baseline with simple algorithms and obvious features, then methodically explore alternatives to improve upon that baseline.

You can explore multiple avenues for improving performance: increasing data volume, engineering better features, selecting more appropriate algorithms, combining models through ensemble methods, and tuning hyperparameters. Each approach provides unique benefits and may be more or less effective depending on your use case. The key is to follow a systematic process, measure results accurately, and document what you learn.

Implementation steps

1. **Establish a baseline model**. Start with a simple architecture, obvious features, and a straightforward algorithm to create your baseline. Use Amazon SageMaker AI built-in algorithms

to quickly develop this initial model. This gives you a reference point for comparing future experiments and improvements.

2. **Set up experiment tracking**. Use Amazon SageMaker AI Managed MLFlow to organize, track, compare, and evaluate your machine learning experiments. Create a structured framework that tracks performance metrics, algorithm choices, features used, and hyperparameter settings so you can effectively compare results across different approaches.
3. **Test different algorithms**. Systematically test various algorithms, starting with simpler ones and progressively trying more complex options. SageMaker AI provides many built-in algorithms that you can compare. Document how each algorithm performs relative to your baseline and identify which ones show the most promise for your data and problem.
4. **Apply feature engineering**. Extract important signals in your data through feature engineering. This may include feature selection, transformation, creation of new features, normalization, and encoding techniques. Use SageMaker AI Feature Store to manage and share features across experiments and teams.
5. **Increase data volume and quality**. Evaluate whether adding more data or improving data quality could assist your model. More data often broadens the statistical range and improve model performance, but only if the additional data is relevant and of good quality.
6. **Implement ensemble methods**. Combine multiple models to leverage different strengths and compensate for individual weaknesses. Techniques like bagging, boosting, and stacking can often improve overall accuracy. SageMaker AI makes it simple to implement ensemble predictions from multiple models.
7. **Perform hyperparameter tuning**. Use Amazon SageMaker AI Automatic Model Tuning to optimize your model's hyperparameters. This service automates the search through different hyperparameter combinations to find optimal values that improve model performance. You can run multiple HPO jobs in parallel to speed up the process.
8. **Evaluate improvements systematically**. For each change, rigorously evaluate whether performance has improved based on relevant metrics for your problem. Use SageMaker AI's evaluation tools to compare results across experiments and determine which approaches deliver the most gains.
9. **Optimize for production**. Once you've identified the best performing approach, optimize it for production deployment. Consider factors like inference latency, model size, and resource requirements alongside pure performance metrics.
10. **Document findings and methodology**. Create comprehensive documentation of your benchmarking process, including what worked, what didn't, and why. This provides valuable information for future model improvements and builds institutional knowledge.

Resources

Related documents:

- Built-in algorithms and pretrained models in Amazon SageMaker AI
- Accelerate generative AI development using managed MLflow on Amazon SageMaker AI
- Automatic model tuning with SageMaker AI
- Use Feature Store with SDK for Python (Boto3)
- Feature Processing with Spark ML and Scikit-learn
- Running multiple HPO jobs in parallel on Amazon SageMaker AI
- Optimizing portfolio value with Amazon SageMaker AI automatic model tuning

Related videos:

- How To Efficiently Manage ML experiments using Amazon SageMaker AI ML Flow
- Train large models on Amazon SageMaker AI for scale and performance

Related examples:

- Feature Engineering Immersion Day Workshop
- Ensemble Predictions From Multiple Models
- Amazon SageMaker AI Examples GitHub Repository

MLPERF04-BP03 Establish a model performance evaluation pipeline

Establish an end-to-end model performance evaluation pipeline that captures key metrics to evaluate your model's success, align with business KPIs, and automatically test performance when models or data are updated.

Desired outcome: You can systematically evaluate model performance through automated pipelines that measure relevant metrics specific to your use case. Your evaluation process runs automatically whenever model or data updates occur, creating a continuous quality assessment. This assists you in maintaining high-performing models that deliver business value while providing transparency into model behavior and performance over time.

Common anti-patterns:

- Relying solely on training accuracy without considering real-world performance metrics.

- Manual evaluation of models that leads to inconsistency.
- Using generic metrics that don't align with business KPIs.
- Waiting until deployment to evaluate model performance.
- Not establishing automated evaluation triggers when models or data change.

Benefits of establishing this best practice:

- Verifies that models maintain expected performance levels over time.
- Provides data-driven decision making for model selection and deployment.
- Aligns machine learning outcomes with business objectives.
- Enables faster identification and resolution of performance degradation.
- Improves regulatory adherence through consistent evaluation protocols.
- Increases stakeholder confidence through transparent performance reporting.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Model performance evaluation is critical to verify that your machine learning solutions deliver on their intended business outcomes. By establishing a robust, automated evaluation pipeline, you can consistently assess how well your models perform against business KPIs and make data-driven decisions about deployment readiness.

Avoid relying solely on training accuracy without considering real-world performance metrics. Many organizations use manual, ad-hoc evaluation of models that leads to inconsistency, use generic metrics that don't align with business KPIs, wait until deployment to evaluate model performance, and fail to establish automated evaluation runs when models or data change.

Your evaluation pipeline should incorporate metrics specific to your use case. For regression problems, this might include Root Mean Squared Error (RMSE). For classification tasks, accuracy, precision, recall, F1 score, and area under the curve (AUC) are common metrics. These technical metrics should tie directly to business KPIs, helping stakeholders understand the model's contribution to business goals.

Automating the evaluation process provides consistency and reduces manual errors. When new data arrives or models are updated, your pipeline should automatically run evaluations, providing continuous feedback on model performance and enabling rapid identification of any degradation issues.

Implementation steps

1. **Define business objectives and evaluation criteria**. Begin by clearly defining what success means for your machine learning use case. Identify relevant business KPIs and determine which technical metrics (like accuracy, precision, recall, F1 score, RMSE, AUC) best align with these business goals. Document these metrics and their target values to establish clear evaluation criteria.
2. **Create an end-to-end workflow with Amazon SageMaker AI Pipelines**. Start with a workflow template to establish an initial infrastructure for model training and deployment. SageMaker AI Pipelines can automate different steps of the ML workflow including data loading, data transformation, training, tuning, and deployment. Within SageMaker AI Pipelines, the SageMaker AI Model Registry tracks model versions and respective artifacts, including metadata and lineage data collected throughout the model development lifecycle.
3. **Implement model evaluation components in your pipeline**. Design dedicated evaluation steps within your pipeline that calculate relevant metrics for your model. Use SageMaker AI Processing jobs or custom Python scripts to perform evaluations on validation datasets. Store evaluation results in a central location for tracking performance over time.
4. **Set up automated prompts for evaluation**. Configure your pipeline to automatically initiate the evaluation process whenever there's a model update or new training data becomes available. This provides continuous quality assessment and identifies performance degradation early.
5. **Create visualization and reporting mechanisms**. Implement dashboards or reports that display model performance metrics in a straightforward format. Stakeholders can use visualizations to quickly assess model performance against business KPIs and make informed decisions about model deployment.
6. **Establish model approval workflows**. Define criteria for model approval based on evaluation results. Implement approval workflows in the SageMaker AI Model Registry that automatically promote models meeting performance thresholds to production, while flagging underperforming models for review.
7. **Implement A/B testing capabilities**. For production models, set up A/B testing infrastructure to compare performance of new models against baseline models using real-world data. This provides additional validation before fully deploying model updates.
8. **Monitor production model performance**. Use Amazon SageMaker AI Model Monitor to continuously monitor deployed models for data drift, model drift, and performance degradation. Set up alerts when performance metrics fall below acceptable thresholds.

9. **Implement bias detection and fairness evaluation**. Use Amazon SageMaker AI Clarify to detect bias in your models and check fairness across different demographic groups. Include bias metrics as part of your evaluation criteria.
10. **Create feedback loops for continuous improvement**. Design mechanisms to capture feedback from production model performance and incorporate these insights into future model iterations. This creates a cycle of continuous improvement based on real-world performance.

Resources

Related documents:

- Amazon SageMaker AI Pipelines
- Define a pipeline
- Model Registration Deployment with Model Registry
- Data and model quality monitoring with SageMaker AI Model Monitor
- Fairness and Explainability with SageMaker AI Clarify
- Data transformation workloads with SageMaker AI Processing
- Building, automating, managing, and scaling ML workflows using Amazon SageMaker AI Pipelines
- Extend Amazon SageMaker AI Pipelines to include custom steps using callback steps

Related videos:

- Amazon SageMaker AI Pipelines
- How to create fully automated ML workflows with Amazon SageMaker AI Pipelines

Related examples:

- Orchestrate your build and train pipeline with SageMaker AI Pipelines
- SageMaker AI Model Evaluation Examples
- MLOps with SageMaker AI Pipelines

MLPERF04-BP04 Establish feature statistics

Establishing key statistics to measure changes in data that affect model outcomes is crucial for maintaining ML model performance. By analyzing feature importance and sensitivity, you can

select the most critical features to monitor and detect when data drifts outside acceptable ranges so you can determine when model retraining is necessary.

Desired outcome: You establish a robust monitoring system that tracks key statistics for the most influential features in your machine learning models. You can detect data drift that could impact model performance, allowing for timely model retraining decisions based on quantitative measures rather than intuition. Your monitoring system alerts you when important features drift outside their expected statistical ranges, providing continuous model reliability and performance.

Common anti-patterns:

- Monitoring features equally without considering their relative importance to model outcomes.
- Failing to establish baseline statistics for important features before deploying models.
- Not setting appropriate thresholds for data drift alerts.
- Monitoring only model outputs without analyzing input feature distributions.
- Neglecting to perform sensitivity analysis to understand model behavior at decision boundaries.

Benefits of establishing this best practice:

- Early detection of data quality issues that could affect model performance.
- Reduced model performance degradation through timely retraining.
- Greater understanding of which features most impact model predictions.
- Improved model reliability in production environments.
- Enhanced ability to explain model behavior and decision boundaries to stakeholders.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Establishing feature statistics is essential for maintaining model performance over time. As real-world data evolves, your model's predictive power can deteriorate if the data drift exceeds certain thresholds. By focusing on the most influential features and understanding your model's sensitivity to changes in these features, you can create an effective monitoring strategy.

Start by analyzing which features have the greatest impact on your model's predictions through feature importance analysis. Then establish baseline statistics for these critical features using your training data. Monitor these statistics in production, comparing them to your baseline, and set up alerts when deviations occur. This approach allows you to proactively address potential model performance issues before they impact your business outcomes.

Implementation steps

1. **Analyze feature distributions with Data Wrangler**. Use Amazon SageMaker AI Data Wrangler to perform exploratory data analysis on your dataset. Examine the distribution of each feature, identify outliers, and understand relationships between features. Data Wrangler provides visualizations such as histograms, scatter plots, and correlation matrices to understand your data's characteristics before training.
2. **Train your model with proper tracking**. When training your model, capture metadata about the training process using SageMaker AI Managed MLFlow. This can establish a baseline for comparison and enables reproducibility of your experiments. Track key metrics, parameters, and the training dataset version to maintain a complete record of model development.
3. **Determine feature importance**. After training your model, analyze which features have the greatest impact on predictions. Use built-in feature importance methods in SageMaker AI, such as SHAP (SHapley Additive exPlanations) values or permutation importance. Alternatively, use model-specific methods like feature importance in tree-based models or coefficient magnitudes in linear models.
4. **Perform sensitivity analysis**. Map out regions in feature space where predictions change abruptly or remain invariant. Focus particularly on features near decision boundaries where small changes can alter model outputs. Use Amazon SageMaker AI Clarify to analyze how variations in input features affect predictions and understand which features require the closest monitoring.
5. **Check for data bias**. Use Amazon SageMaker AI Clarify to analyze your dataset for potential biases. Imbalances or biases in your training data can lead to poor generalization and unfair predictions. Identify and address these issues before deploying your model to create ethical and reliable ML systems.
6. **Establish monitoring baseline**. Configure Amazon SageMaker AI Model Monitor to create a baseline from your training data. This baseline captures the expected statistical properties of your features, including distributions, ranges, and relationships. SageMaker AI automatically analyzes and creates constraints for each feature based on the training data.
7. **Configure data quality monitoring**. Set up SageMaker AI Model Monitor to continuously evaluate production data against your established baseline. Configure monitoring schedules based on your application's requirements—hourly, daily, or weekly. Define thresholds for acceptable deviation from the baseline for each important feature.
8. **Implement data drift detection**. Configure alerts to notify you when important features drift outside their acceptable statistical ranges. Use Amazon CloudWatch to set up alarms that run

when drift metrics exceed thresholds. This enables timely intervention when data quality issues arise.

9. **Create model retraining prompts**. Establish criteria for when to retrain your model based on data drift metrics. For example, if multiple important features show drift, or if a single critical feature drifts beyond a certain threshold, run the model retraining process.
10. **Set up continuous feedback loop**. Implement a system to continuously gather new labeled data for model retraining. This verifies that your model can adapt to legitimate changes in data distribution over time. Use AWS Step Functions to orchestrate workflows that include data collection, preprocessing, model training, and deployment.

Resources

Related documents:

- Pre-training Data Bias
- Data and model quality monitoring with SageMaker AI Model Monitor
- Data quality
- Accelerate generative AI development using managed MLflow on Amazon SageMaker AI

Related videos:

- Prepare data for machine learning with ease, speed, and accuracy
- Detect machine learning (ML) model drift in production

Related examples:

- Lab 1. Feature Engineering
- SageMaker AI Model Monitor Examples
- SageMaker AI Clarify Explainability Examples

MLPERF04-BP05 Perform a performance trade-off analysis

Perform a trade-off analysis to identify optimal ML model configurations that balance competing requirements for your business needs. This practice enables you to maximize both model accuracy and overall business value.

Desired outcome: You develop a structured approach to evaluate and select machine learning models based on multiple dimensions including accuracy, complexity, bias, fairness, and operational constraints. You'll be able to make informed decisions about model selection that align with your business requirements and ethical considerations.

Common anti-patterns:

- Focusing solely on model accuracy without considering other important factors like explainability, fairness, or inference speed.
- Ignoring bias in training data that may lead to unfair model outcomes for certain groups.
- Deploying overly complex models that are difficult to explain and maintain when simpler models could achieve adequate performance.
- Not testing different model configurations against business requirements.

Benefits of establishing this best practice:

- Optimized machine learning models that balance performance with operational constraints.
- Models that can be explained and trusted by stakeholders and end users.
- Reduced risk of unfair or biased model outcomes.
- Better alignment between model performance and business requirements.
- More cost-effective model deployment and maintenance.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Performance trade-off analysis requires careful consideration of your use case and business requirements. You need to determine which aspects of model performance are most important for your application - whether that's accuracy, explainability, fairness, latency, or other factors. Different business contexts may prioritize these dimensions differently.

For example, in a credit scoring application, fairness and explainability might be primary concerns due to regulatory requirements and the need to provide reasons for decisions. In contrast, a real-time product recommendation system might prioritize prediction speed and accuracy over explainability. Understanding these requirements upfront can guide your model development process.

Trade-off analysis is not a one-time activity but should be incorporated throughout the machine learning lifecycle. As you gather more data, refine your models, and receive feedback from

stakeholders, you should continually reassess these trade-offs to verify that your models continue to meet business needs.

Implementation steps

1. **Define performance metrics aligned with business goals**. Start by clearly defining what success looks like for your use case. Identify the key performance indicators (KPIs) that matter most to your business stakeholders. These might include technical metrics like precision, recall, or latency, as well as business metrics like conversion rate or cost reduction.
2. **Establish a baseline for trade-off analysis**. Create a simple model as a baseline to compare against more complex approaches. This provides a reference point for measuring improvements and understanding the minimum acceptable performance for your use case. Techniques like cross-validation can determine if your baseline is robust.
3. **Explore the accuracy versus complexity trade-off**. Test models with different levels of complexity, from simple linear models to more sophisticated deep learning approaches. Use Amazon SageMaker AI Managed MLFlow to track different model architectures and their performance characteristics. Remember that simpler models are often more explainable and simpler to deploy, even if they sacrifice some accuracy.
4. **Analyze bias and fairness implications**. Use Amazon SageMaker AI Clarify to detect potential bias in your data and models. Identify sensitive attributes that might lead to unfair outcomes for certain groups. Implement mitigation strategies such as balanced datasets, regularization techniques, or fairness-aware algorithms to reduce bias while maintaining acceptable performance.
5. **Optimize the bias versus variance trade-off**. Address underfitting (high bias) and overfitting (high variance) through systematic experimentation. Techniques like cross-validation can identify the optimal model complexity for your data. Consider using more training data, implementing regularization techniques, or simplifying your model architecture depending on whether bias or variance is your primary concern.
6. **Evaluate precision versus recall trade-offs**. For classification problems, determine whether false positives or false negatives are more problematic for your use case. Use tools like precision-recall curves to visualize this trade-off and ROC curves to understand the relationship between true positive and false positive rates. Adjust classification thresholds based on the relative costs of different types of errors.
7. **Consider operational constraints**. Evaluate how models perform under real-world constraints like latency requirements, memory limitations, or compute availability. For edge deployment scenarios, use Amazon SageMaker AI Neo to optimize your models for hardware targets while

maintaining accuracy. This is particularly important for applications that need to run in resource-constrained environments.

8. **Implement explainability techniques**. Use Amazon SageMaker AI Clarify to generate feature importance explanations and understand how your model makes predictions. This builds trust with stakeholders and may be necessary to address regulatory adherence in some industries. Consider the trade-off between model complexity and explainability when selecting your final model.
9. **Document trade-off decisions**. Create comprehensive documentation of your trade-off analysis, including the experiments performed, results observed, and the rationale behind your final model selection. This provides transparency for stakeholders and provides an understanding to future teams on why certain decisions were made.
10. **Establish continuous monitoring**. After deployment, use Amazon SageMaker AI Model Monitor to track model performance and detect drift in data or predictions. This allows you to identify when your trade-off assumptions may no longer be valid and when retraining might be necessary.

Resources

Related documents:

- Evaluating ML Models
- AI Fairness and Explainability Whitepaper
- Optimize model performance using Amazon SageMaker AI Neo
- Data and model quality monitoring with SageMaker AI Model Monitor
- Fairness, model explainability and bias detection with SageMaker AI Clarify
- Accelerating generative AI development with fully managed MLflow 3.0 on Amazon SageMaker AI
- Amazon SageMaker AI Clarify Detects Bias and Increases the Transparency of Machine Learning Models
- Unlock near 3x performance gains with XGBoost and Amazon SageMaker AI Neo

Related videos:

- Building explainable AI models with Amazon SageMaker AI

MLPERF04-BP06 Detect performance issues when using transfer learning

Transfer learning can accelerate machine learning development by using pre-trained models for new tasks. Monitoring the performance of these transferred models verifies that they yield accurate results in new contexts and stops inherited weaknesses from affecting your solutions.

Desired outcome: You can effectively identify and address hidden problems in transfer learning applications, which improves the reliability of your model predictions. By implementing proper monitoring and validation techniques, you gain confidence that inherited prediction weights perform as expected for your use case while minimizing risks associated with using pre-trained models.

Common anti-patterns:

- Assuming a pre-trained model will automatically perform well on your task without validation.
- Neglecting to monitor prediction accuracy and model behavior after transfer learning implementation.
- Failing to examine model predictions for subtle but consequential errors.
- Overlooking the need to validate input preprocessing techniques for transferred models.

Benefits of establishing this best practice:

- Early detection of performance issues that might otherwise remain hidden.
- Improved model reliability and prediction accuracy.
- Better understanding of what capabilities are truly inherited from pre-trained models.
- Reduced risk of model failures in production environments.
- More effective fine-tuning strategies based on identified weaknesses.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Transfer learning can dramatically reduce the time and computational resources needed to develop effective machine learning models by leveraging knowledge gained from solving one problem and applying it to a different but related problem. However, this approach comes with unique challenges that require careful monitoring and validation.

When using transfer learning, it's essential to understand that the pre-trained model's performance characteristics may not directly translate to your use case. The underlying patterns and relationships learned by the original model might not fully align with your target domain, leading to subtle but potentially serious performance issues. These problems can be especially challenging to identify because they often don't manifest as obvious failures but rather as biased or suboptimal predictions.

For effective transfer learning implementations, you need comprehensive monitoring and debugging strategies that can detect these hidden issues. This involves validating not just overall model performance but also examining individual predictions, understanding the inherited capabilities, and properly preprocessing the inputs.

Implementation steps

1. **Set up Amazon SageMaker AI Debugger for monitoring**. Configure Amazon SageMaker AI Debugger to monitor your transfer learning model during training and inference. This service can identify hidden issues that might otherwise go undetected by automatically analyzing tensors, tracking model convergence, and detecting anomalies.
2. **Examine model predictions for errors**. Analyze the outputs of your transfer learning model to identify patterns in prediction errors. Look beyond aggregate metrics like accuracy or F1 score to understand what types of inputs are causing the most confusion. Create confusion matrices and error distribution reports to visualize where your model's performance deviates from expectations.
3. **Validate model robustness**. Test your model's performance under various input conditions to determine how much of its robustness comes from the pre-trained weights versus your fine-tuning process. Perform adversarial testing by introducing slight variations to inputs and measuring how the predictions change. Use SageMaker AI Debugger's built-in rules to detect training anomalies, such as vanishing gradients or exploding tensors.
4. **Verify input preprocessing methods**. Align your data preprocessing pipeline with the expectations of the pre-trained model. Inconsistencies in normalization, tokenization, or feature engineering can impact performance. Document and validate the preprocessing steps to maintain consistency between training and inference stages.
5. **Implement continuous performance monitoring**. Deploy systems to continually monitor your model's performance after deployment. Configure automated alerts for deviations in key performance metrics to catch potential issues early. Use Amazon CloudWatch in conjunction with SageMaker AI Model Monitor to set up comprehensive monitoring dashboards and alerting systems.

6. **Leverage foundation models with fine-tuning**. When using foundation models for transfer learning, implement Amazon SageMaker AI JumpStart to access pre-trained models and fine-tune them for your tasks. Monitor the alignment between generated outputs and expected results, particularly for tasks requiring domain-specific knowledge.

Resources

Related documents:

- Amazon SageMaker AI Debugger
- Data and model quality monitoring with SageMaker AI Model Monitor
- SageMaker AI JumpStart pretrained models
- Detecting data drift using SageMaker AI
- Detecting hidden but non-trivial problems in transfer learning models using Amazon SageMaker AI Debugger

Related videos:

- Introduction to Amazon SageMaker AI Debugger
- Detect machine learning (ML) model drift in production

Deployment

Best practices

- MLPERF05-BP01 Evaluate cloud versus edge options for machine learning deployment
- MLPERF05-BP02 Choose an optimal deployment option in the cloud

MLPERF05-BP01 Evaluate cloud versus edge options for machine learning deployment

Evaluate machine learning deployment options to determine if your application requires near-instantaneous inference results or needs to operate without network connectivity. When the lowest possible latency is essential, deploying inference directly on edge devices avoids costly roundtrips to cloud API endpoints. Edge deployments are particularly valuable for use cases like predictive maintenance in factories, where immediate local responses are critical.

Desired outcome: You can make informed decisions about where to deploy your machine learning models based on your business requirements. You understand when to use cloud resources for training and when to deploy optimized models to edge devices for low-latency inference. Your edge deployments can operate autonomously when needed while maintaining security, performance, and the ability to update models as new data becomes available.

Common anti-patterns:

- Defaulting to cloud-based inference without evaluating latency requirements.
- Deploying models to edge devices without proper optimization, resulting in poor performance.
- Neglecting to establish a strategy for model updates and monitoring on edge devices.
- Overlooking security considerations for models deployed at the edge.
- Failing to evaluate hardware constraints of edge devices before deployment.

Benefits of establishing this best practice:

- Dramatically reduced inference latency for time-sensitive applications.
- Ability to operate ML models in environments with limited or no connectivity.
- Lower operational costs by reducing network traffic and cloud compute usage.
- Enhanced privacy by keeping sensitive data local to edge devices.
- Improved resilience against network outages.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Machine learning deployments require careful consideration of where inference should take place based on your use case requirements. While training complex models is computationally intensive and best suited for the cloud, inference operations require less computing power and can often be performed directly on edge devices.

Avoid defaulting to cloud-based inference without evaluating latency requirements. Many organizations deploy models to edge devices without proper optimization, resulting in poor performance, neglect to establish a strategy for model updates and monitoring on edge devices, overlook security considerations for models deployed at the edge, and fail to evaluate hardware constraints of edge devices before deployment.

When evaluating cloud versus edge deployment, consider factors like latency requirements, connectivity constraints, data privacy needs, and the computational capabilities of your edge

devices. For applications requiring real-time responses, such as autonomous vehicles, industrial equipment monitoring, or smart security systems, edge deployment reduces network latency and provides continuous operation even during connectivity disruptions.

AWS provides comprehensive tools to optimize models for edge deployment while maintaining the ability to train and manage those models in the cloud. This hybrid approach gives you the best of both worlds: powerful cloud resources for development and optimization, with efficient edge deployment for operational performance.

Implementation steps

1. **Assess your deployment requirements**. Begin by clearly defining your application's latency, connectivity, and privacy requirements. Determine if your use case needs millisecond-level response times, must function in environments with unreliable connectivity, or needs to process sensitive data locally. These factors will guide your decision between cloud and edge deployment options.
2. **Optimize models for edge deployment**. Training and optimizing machine learning models require massive computing resources, making cloud environments ideal for this phase. Amazon SageMaker AI provides powerful tools for building and training models that can later be optimized for edge deployment. Consider the computational constraints of your target edge devices and select model architectures that balance accuracy with efficiency.
3. **Deploy with Amazon SageMaker AI Neo for cross-solution compatibility**. Amazon SageMaker AI Neo enables ML models to be trained once and run anywhere in the cloud or at the edge. The Neo compiler reads models exported from various frameworks, converts them to framework-agnostic representations, and generates optimized binary code for target hardware. This process makes your models run faster without accuracy loss.
4. **Implement edge ML with AWS IoT Greengrass**. AWS IoT Greengrass provides a robust solution for running ML inferences on edge devices using cloud-trained models. These models can be built using Amazon SageMaker AI, AWS Deep Learning AMIss, or AWS Deep Learning Containers. Models are stored in Amazon S3 before deployment to edge devices.

Resources

Related documents:

- AWS IoT Greengrass
- Set up Neo on Edge Devices

- AWS Internet of Things
- Optimize image classification on AWS IoT Greengrass using ONNX Runtime

Related videos:

- Machine Learning at the Edge
- Getting Started Using Machine Learning at the Edge
- AWS IoT Greengrass and Machine Learning at the Edge

MLPERF05-BP02 Choose an optimal deployment option in the cloud

When deploying machine learning models in the cloud, selecting the right deployment option is crucial for performance efficiency. By matching your deployment method to your use case requirements for request frequency, latency, and runtime, you can optimize both performance and cost.

Desired outcome: You can deploy your machine learning models in a way that meets your application's needs for throughput, response time, and cost efficiency. The selected deployment option provides the optimal balance between performance and resource utilization while accommodating your workload patterns.

Common anti-patterns:

- Deploying models on persistent endpoints regardless of traffic patterns or workload spikes.
- Overlooking payload size and processing time requirements when selecting deployment options.
- Using real-time inference for batch processing use cases that don't require immediate responses.
- Failing to consider cost implications of different deployment options.
- Not monitoring and optimizing deployment configurations after initial setup.

Benefits of establishing this best practice:

- Improved cost efficiency by matching resources to actual usage patterns.
- Enhanced performance through selection of appropriate deployment methods.
- Better scalability to handle varying workloads.
- Reduced operational overhead with managed deployment options.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Selecting the optimal deployment option for your machine learning models involves understanding your use case requirements and matching them to the capabilities of different AWS SageMaker AI deployment services. Consider factors such as request frequency, payload size, processing time, and response latency needs.

Avoid deploying models on persistent endpoints regardless of traffic patterns or workload spikes. Many organizations overlook payload size and processing time requirements when selecting a deployment option, use real-time inference for batch processing use cases that don't require immediate responses, and fail to consider cost implications of different deployment options.

For time-sensitive applications requiring immediate responses, real-time inference provides persistent endpoints, while workloads with inconsistent traffic patterns might benefit from serverless options that scale automatically. For larger payloads or longer processing times, asynchronous inference is appropriate, and for non-time-sensitive bulk processing, batch transformation offers an efficient option.

Your deployment choice should align with your application's operational patterns to balance performance and cost efficiency. A chatbot requiring immediate responses would benefit from real-time inference, while overnight batch processing of transactions might use batch transform.

Implementation steps

1. **Evaluate your model deployment requirements**. Begin by clearly defining your application's requirements for inference frequency, latency needs, payload sizes, and budget constraints. Consider how often model predictions will be requested, how quickly responses must be delivered, and what resource constraints you may have.
2. **Implement Amazon SageMaker AI Real-time Inference for continuous, low-latency needs**. Deploy models that require near-instantaneous responses and consistent availability using SageMaker AI real-time endpoints. These fully managed endpoints support auto-scaling and are ideal for applications like real-time recommendation engines, chatbots, or fraud detection systems where immediate response is critical.
3. **Implement Amazon SageMaker AI Serverless Inference for variable traffic patterns**. For workloads with inconsistent request patterns or idle periods between traffic spikes, use SageMaker AI Serverless Inference. This option automatically provisions and scales compute resources based on traffic, avoiding the need to manage server infrastructure while optimizing costs during periods of low utilization.

4. **Implement Amazon SageMaker AI Asynchronous Inference for large payloads or long processing**. For use cases involving large input files (up to 1GB) or models requiring extended processing time (up to 15 minutes), deploy using SageMaker AI Asynchronous Inference. This option queues incoming requests and processes them when resources are available, making it ideal for tasks like video processing, large document analysis, or complex NLP tasks.
5. **Implement Amazon SageMaker AI Batch Transform for scheduled bulk processing**. For non-time-sensitive workloads where predictions can be processed in batches, such as overnight processing of transactions or weekly sentiment analysis of customer feedback, use SageMaker AI Batch Transform. This option automatically distributes workloads across compute instances and shuts down resources when processing is complete.
6. **Monitor and optimize your deployment**. Once deployed, continuously monitor your model's performance, resource utilization, and costs. Use Amazon CloudWatch metrics to track invocation metrics, errors, latency, and resource utilization. Adjust auto-scaling configurations or switch deployment options if your usage patterns change over time.
7. **Implement security and governance**. Incorporate proper security controls in your model deployments, including IAM roles with least privilege access, network isolation where appropriate, and encryption of data in transit and at rest. Use Amazon SageMaker AI Role Manager to create persona-based IAM roles for different ML user types (data scientists, MLOps engineers, business analysts) with preconfigured templates that follow least-privilege principles. For regulated industries, implement model governance practices to track model versions, approvals, and changes.

Resources

Related documents:

- Real-time inference
- Deploy models with Amazon SageMaker AI Serverless Inference
- Asynchronous inference
- Batch transform for inference with Amazon SageMaker AI
- Amazon SageMaker AI Role Manager
- Deploy models with Amazon SageMaker AI
- Deploying ML models using SageMaker AI Serverless Inference
- Optimize deployment cost of Amazon SageMaker AI JumpStart foundation models with Amazon SageMaker AI asynchronous endpoints
- Announcing managed inference for Hugging Face models in Amazon SageMaker AI

- Run computer vision inference on large videos with Amazon SageMaker AI asynchronous endpoints

Related videos:

- Achieve high performance and cost-effective model deployment
- Amazon SageMaker AI serverless inference

Monitoring

Best practices

- MLPERF06-BP01 Include human-in-the-loop monitoring
- MLPERF06-BP02 Evaluate model explainability
- MLPERF06-BP03 Evaluate data drift
- MLPERF06-BP04 Monitor, detect, and handle model performance degradation
- MLPERF06-BP05 Establish an automated re-training framework
- MLPERF06-BP06 Review for updated data and features for retraining

MLPERF06-BP01 Include human-in-the-loop monitoring

Including human-in-the-loop monitoring is an effective method for efficiently tracking and maintaining model performance. By incorporating human review into automated decision processes, organizations can establish a reliable quality assurance mechanism that validates model inferences and detects performance degradation over time.

Desired outcome: You implement a robust human-in-the-loop monitoring system that enables continuous assessment of your machine learning models. You can compare human labels with model inferences to detect model drift and performance degradation, allowing timely mitigation through retraining or other remediation actions. This creates a feedback loop that maintains high model quality and reliability in production environments.

Common anti-patterns:

- Relying solely on automated metrics without human validation.
- Ignoring edge cases and low-confidence predictions.
- Not establishing a systematic review process for model outputs.

- Failing to incorporate human feedback into model retraining cycles.
- Using untrained reviewers without subject matter expertise.

Benefits of establishing this best practice:

- Early detection of model drift and performance degradation.
- Higher quality assurance for critical model predictions.
- Better understanding of edge cases and model limitations.
- Continuous improvement of model performance through expert feedback.
- Increased trust in AI systems through human oversight.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Human-in-the-loop monitoring provides a crucial safety net for your machine learning systems by adding appropriate human oversight to important decisions. This approach is particularly valuable when automated systems make predictions that impact critical business processes or customer experiences. By establishing a workflow where human experts review model outputs, particularly those with low confidence or selected randomly for quality assurance, you create a reliable mechanism to evaluate model performance in real-world scenarios.

Avoid relying solely on automated metrics without human validation. Many organizations ignore edge cases and low-confidence predictions, don't establish a systematic review process for model outputs, fail to incorporate human feedback into model retraining cycles, and use untrained reviewers without subject matter expertise.

This monitoring approach can identify when models begin to drift or perform poorly on new data. The comparison between human labels and model predictions serves as a key indicator of model health, signaling when retraining or other interventions are necessary. This feedback loop is essential for maintaining high-quality, reliable AI systems over time.

Implementation steps

1. **Design a quality assurance system for model inferences**. Create a comprehensive plan for how human review will integrate with your machine learning workflow. Determine which predictions will be sent for human review (low-confidence predictions, random samples, or high-risk categories) and establish clear guidelines for reviewers to follow when evaluating model outputs.

2. **Establish a team of subject matter experts**. Identify and recruit individuals with domain expertise who can accurately evaluate model inferences. These reviewers should understand both the technical aspects of your models and the business context in which they operate, allowing them to provide valuable feedback on model performance and identify potential issues.
3. **Implement Amazon Augmented AI for human review workflows**. Use Amazon Augmented AI (Amazon A2I) to create and manage human review workflows for your machine learning models. Amazon A2I integrates with other AWS services like IAM, Amazon SageMaker AI, and Amazon S3 to handle the entire review process.
4. **Configure review criteria and thresholds**. Define the conditions that initiate human review, such as confidence score thresholds or types of predictions that require human validation. Set up rules in Amazon A2I to automatically route these cases to your human reviewers while allowing high-confidence, routine predictions to proceed without review.
5. **Develop feedback integration mechanisms**. Create systems to incorporate human feedback into your model improvement cycle. This includes storing human labels alongside model predictions, analyzing disagreement patterns, and using this information to identify areas where your model needs improvement.
6. **Monitor and analyze human-model agreement rates**. Track how often human reviewers agree with model predictions and analyze patterns in disagreements. This data can identify systematic issues with your model so that you can prioritize areas for improvement.
7. **Implement model retraining based on feedback**. Use the labeled data gathered through human review to periodically retrain your models. This creates a continuous improvement loop where your models learn from past mistakes and adapt to changing patterns in your data.
8. **Measure and optimize cost-effectiveness**. Analyze the ROI of your human-in-the-loop system by comparing the costs of human review with the benefits of improved model accuracy. Adjust your review sampling strategy to focus human attention where it provides the most value.

Resources

Related documents:

- Using Amazon Augmented AI for Human Review
- Data and model quality monitoring with SageMaker AI Model Monitor
- Customized model monitoring for near real-time batch inference with Amazon SageMaker AI-
- Human-in-the-loop review of model explanations with Amazon SageMaker AI Clarify and Amazon A2I

Related videos:

- Easily Implement Human in the Loop into Your Machine Learning Predictions with Amazon A2I
- Accelerate foundation model evaluation with Amazon SageMaker AI Clarify

MLPERF06-BP02 Evaluate model explainability

Model explainability allows you to understand and interpret how your machine learning models arrive at their decisions. By evaluating model explainability, you gain insights into the factors that influence predictions to build trustworthy AI systems that meet business requirements and regulatory standards.

Desired outcome: You can demonstrate why your machine learning models make predictions, which enables you to build trust with stakeholders, adhere to regulatory requirements, and identify potential biases in model outcomes. You have the tools to balance model complexity with explainability based on your business needs and can produce documentation that satisfies governance requirements.

Common anti-patterns:

- Treating machine learning models as unknown without understanding their decision-making process.
- Ignoring explainability requirements until after model deployment.
- Prioritizing model performance metrics over interpretability when business or regulations requires explainability.
- Failing to document model explanations for regulatory adherence.
- Using complex models when simpler, more interpretable alternatives would meet business requirements.

Benefits of establishing this best practice:

- Increased trust from stakeholders and end-users in AI systems.
- Improves adherence with regulations requiring transparent AI decision-making.
- Ability to detect and mitigate biases in model predictions.
- Enhanced model debugging and performance improvement.
- Better alignment between model behavior and business objectives.
- More effective model governance and risk management.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Model explainability is a critical aspect of responsible AI development. When you evaluate explainability, you assess how transparently your machine learning models make decisions and whether those decisions can be explained to stakeholders, regulators, and end-users. This transparency is particularly important in regulated industries and for applications where decisions impact individuals.

Avoid treating machine learning models as opaque without understanding their decision-making process. Many organizations ignore explainability requirements until after model deployment, prioritize model performance metrics over interpretability when business or regulatory requirements demand explainability, and fail to document model explanations for regulatory adherence.

The trade-off between model complexity and explainability is a key consideration. Complex models like deep neural networks may deliver higher accuracy but are often harder to interpret. Simpler models like decision trees or linear regression provide more straightforward explanations but might sacrifice some performance. Your choice should be guided by your business context, including regulatory requirements and the importance of building trust with end-users.

For example, a credit approval system may require clear explanations for why applications are denied, while a manufacturing quality control system might prioritize accuracy over explainability. By evaluating these requirements early, you can select appropriate modeling approaches and develop the right metrics for assessing both performance and interpretability.

Implementation steps

1. **Assess explainability requirements**. Begin by understanding the business and compliance-aligned needs that drive your explainability requirements. Consider regulatory constraints (like GDPR, which includes a right to explanation), business transparency goals, and stakeholder expectations. Document these requirements clearly and prioritize them based on their importance to your use case.
2. **Select appropriate model types**. Choose model architectures that align with your explainability needs. If high explainability is required, consider inherently interpretable models like decision trees, rule-based systems, or linear models. For applications where performance takes priority, more complex models with post-hoc explanation techniques may be appropriate.

3. **Implement Amazon SageMaker AI Clarify**. Amazon SageMaker AI Clarify provides tools to explain model predictions using feature attribution methods. It can identify which features contribute most to a prediction, enabling you to understand and communicate model behavior. SageMaker AI Clarify supports various model types and integrates seamlessly with the SageMaker AI environment.
4. **Apply feature attribution techniques**. Use feature attribution methods like SHAP (SHapley Additive exPlanations) values through SageMaker AI Clarify to quantify the contribution of each feature to individual predictions. These techniques explain both global model behavior (which features are most important overall) and local explanations (why a prediction was made).
5. **Establish explainability metrics**. Define quantitative metrics to assess model explainability, such as feature importance stability, explanation fidelity, or consistency. Use these metrics to objectively evaluate explainability alongside traditional performance metrics like accuracy or F1 score. Include these metrics in your model evaluation framework and monitoring systems.
6. **Create model documentation**. Develop comprehensive documentation that describes how your model works, what features influence its decisions, and how explanations are generated. This documentation should be understandable by both technical and non-technical stakeholders. SageMaker AI Clarify can generate reports that contribute to model governance documentation.
7. **Implement bias detection**. Use SageMaker AI Clarify to detect potential bias in your models during development and production. Configure the appropriate bias metrics based on your use case and sensitive attributes. Regularly assess these metrics to verify that your model remains fair across different demographic groups.
8. **Set up continuous monitoring**. Configure SageMaker AI Clarify to monitor production inferences for bias or feature attribution drift. This allows you to detect when model explanations change over time, which might indicate problems with the model or changes in the underlying data. Establish alerts for shifts in explanations or bias metrics.
9. **Integrate human review processes**. For high-stakes decisions, implement human-in-the-loop review of model explanations using Amazon SageMaker AI Clarify in combination with Amazon Augmented AI (A2I). This provides an additional layer of oversight and can build confidence in the model's decisions.

Resources

Related documents:

- Model Explainability
- Feature Attributions that Use Shapley Values

- Run SageMaker AI Clarify Processing Jobs for Bias Analysis and Explainability
- Data and model quality monitoring with SageMaker AI Model Monitor
- ML model explainability with Amazon SageMaker AI Clarify and the SKLearn pre-built container
- Human-in-the-loop review of model explanations with Amazon SageMaker AI Clarify and Amazon A2I

Related examples:

- Fairness and Explainability with SageMaker AI Clarify
- Explainability with Amazon SageMaker AI Debugger
- Amazon SageMaker AI Clarify Processing GitHub Examples

MLPERF06-BP03 Evaluate data drift

Data drift can impact the performance of your machine learning models, leading to inaccurate predictions and diminished business value. By implementing effective monitoring and detection strategies, you can identify when your models are no longer performing optimally due to changes in the underlying data patterns.

Desired outcome: You can detect and mitigate data drift in your machine learning models, providing continued accuracy and reliability over time. By implementing model monitoring capabilities, you gain visibility into changes in data distributions and model performance, allowing for timely interventions and retraining when necessary.

Common anti-patterns:

- Deploying models without drift monitoring mechanisms.
- Ignoring gradual changes in input data distribution until model performance deteriorates.
- Assuming that a model trained on historical data will remain accurate indefinitely.
- Manually checking model performance at arbitrary intervals rather than implementing continuous monitoring.
- Retraining models on fixed schedules without considering actual data drift patterns.

Benefits of establishing this best practice:

- Early detection of model performance degradation before it impacts business outcomes.
- Increased trust in ML model predictions through continuous quality assurance.

- Improved model lifecycle management with data-driven retraining decisions.
- Reduced operational risks associated with inaccurate predictions.
- Enhanced ability to detect and mitigate bias in ML models over time.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Data drift occurs when the statistical properties of model inputs change over time, causing discrepancies between training data and production data. This can happen gradually due to evolving user behaviors, market conditions, or abrupt changes like economic shifts. When your model encounters data drift, it's essentially making predictions on data distributions it wasn't trained on, which can lead to decreased accuracy and potentially harmful business decisions.

Avoid deploying models without drift monitoring mechanisms. Many organizations ignore gradual changes in input data distribution until model performance deteriorates, assume that a model trained on historical data will remain accurate indefinitely, and manually check model performance at arbitrary intervals rather than implementing continuous monitoring.

Implementing a robust data drift monitoring strategy assists you in maintaining high-performing models by identifying when retraining becomes necessary. Rather than retraining models on arbitrary schedules, you can make data-driven decisions based on actual changes in your data distributions and model performance metrics. This approach verifies that you're optimizing both model performance and resource utilization.

Amazon SageMaker AI provides comprehensive tools to monitor your ML models in production environments, allowing you to detect data drift, concept drift, and bias. By setting up automated monitoring pipelines, you can receive alerts when your models require attention, enabling proactive management of your ML systems.

Implementation steps

1. **Set up baseline data for monitoring**. Establish a reference dataset that represents your model's expected input and output distributions. This baseline will be used to compare against your production data to detect drift. Use the training data or a representative subset that performed well during model validation.
2. **Configure SageMaker AI Model Monitor**. Use Amazon SageMaker AI Model Monitor to automatically detect deviations in your model's data quality and performance metrics.

SageMaker AI Model Monitor can be set up to run on a schedule, analyzing incoming production data and comparing it to your baseline.

3. **Define data quality and drift metrics**. Determine which statistical metrics are most relevant for detecting drift in your use case. SageMaker AI Model Monitor supports various statistical measures like KL divergence, Jensen-Shannon divergence, and population stability index to quantify differences between distributions.
4. **Establish threshold values**. Set appropriate threshold values for your metrics that, when exceeded, will initiate alerts. These thresholds should balance sensitivity to meaningful changes while avoiding false alarms from minor variations.
5. **Create automated monitoring schedules**. Configure SageMaker AI Model Monitor to run analysis jobs at regular intervals that match your business requirements. For critical models, consider more frequent monitoring schedules.
6. **Implement bias detection**. Use Amazon SageMaker AI Clarify to detect bias in your ML models both pre-training and during production. SageMaker AI Clarify can identify if your model is developing biases over time as the data distribution changes.
7. **Set up alert mechanisms**. Configure Amazon CloudWatch alarms to notify appropriate stakeholders when data drift exceeds your defined thresholds. Integrate these alerts with your team's communication tools for timely responses.
8. **Develop a retraining strategy**. Establish clear criteria for when to retrain models based on drift detection results. This should include considerations for data collection, feature engineering updates, and model revalidation.
9. **Implement automated retraining pipelines**. Create SageMaker AI pipelines that can be run automatically when drift exceeds critical thresholds, streamlining the retraining process and minimizing the time models operate with degraded performance.
10. **Document drift patterns and interventions**. Maintain records of detected drift incidents, their causes, and the effectiveness of interventions. This documentation builds institutional knowledge that can improve future model development and monitoring strategies.

Resources

Related documents:

- Data and model quality monitoring with SageMaker AI Model Monitor
- Model Explainability
- Detecting bias after training

- Create, store, and share features with Feature Store
- Amazon SageMaker AI Clarify Detects Bias and Increases the Transparency of Machine Learning Models
- Detecting data drift using Amazon SageMaker AI

Related videos:

- Detect machine learning (ML) model drift in production

Related examples:

- Amazon SageMaker AI Clarify
- Amazon SageMaker AI Model Monitor examples

MLPERF06-BP04 Monitor, detect, and handle model performance degradation

Model performance could degrade over time for reasons such as data quality, model quality, model bias, and model explainability. Continuously monitor the quality of the ML model in real time. Identify the right time and frequency to retrain and update the model. Configure alerts to notify and initiate actions if a drift in model performance is observed.

Desired outcome: You establish a comprehensive monitoring system for your machine learning models that detects performance degradation, alerts relevant stakeholders, and takes appropriate remediation actions. Your ML systems maintain high accuracy and reliability over time through automated monitoring, detection, and handling of performance issues.

Common anti-patterns:

- Implementing ML models without ongoing monitoring.
- Relying solely on periodic manual checks of model performance.
- Ignoring data drift or concept drift until model performance severely degrades.
- Not having an established retraining strategy or schedule.
- Missing alert systems for model performance degradation.

Benefits of establishing this best practice:

- Early detection of model performance issues.
- Automated notifications when models start to degrade.
- Improved model reliability and accuracy over time.
- Reduced operational risk from poor model predictions.
- Better understanding of model behavior in production environments.
- Increased trust in ML-powered systems.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Model performance monitoring is critical for maintaining reliable machine learning systems in production environments. As real-world data changes over time, models can experience data drift (changes in the distribution of input data) or concept drift (changes in the relationship between inputs and target variables). Establish a robust monitoring framework to detect these issues early and take appropriate action.

Avoid implementing ML models without ongoing monitoring. Many organizations rely solely on periodic manual checks of model performance, ignore data drift or concept drift until model performance severely degrades, don't have an established retraining strategy or schedule, and miss alert systems for model performance degradation.

When implementing model monitoring, you should establish baseline performance metrics during the training and validation phases. These baselines serve as the foundation for comparison once the model is deployed. Monitor not just accuracy metrics, but also data statistics, feature distributions, and prediction patterns to identify subtle changes that might indicate underlying problems.

Setting up automated alerts notifies your team when key performance indicators fall below acceptable thresholds. These alerts should be configured with appropriate severity levels to reflect the business impact of model degradation. Additionally, implement automated scaling to handle varying workloads efficiently, which keeps your model endpoints responsive regardless of demand.

Implementation steps

1. **Monitor model performance**. Amazon SageMaker AI Model Monitor continually monitors the quality of Amazon SageMaker AI machine learning models in production. Establish a baseline during training before model is in production. Collect data while in production and compare changes in model inferences. Observations of drifts in the data statistics will indicate that the

model may need to be retrained. Use SageMaker AI Clarify to identify model bias. Configure alerting systems with Amazon CloudWatch to send notifications for unexpected bias or changes in data quality.

2. **Perform automatic scaling**. Amazon SageMaker AI includes automatic scaling capabilities for your hosted model to dynamically adjust underlying compute supporting an endpoint based on demand. This capability verifies that that your endpoint can dynamically support demand while reducing operational overhead.
3. **Monitor endpoint metrics**. Amazon SageMaker AI also outputs endpoint metrics for monitoring the usage and health of the endpoint. Amazon SageMaker AI Model Monitor provides the capability to monitor your ML models in production and provides alerts when data quality issues appear. For enhanced observability, leverage one-click metrics and monitoring for HyperPod training jobs, deployments, health, resource usage, and historical job traces to drive faster debugging and operational excellence in foundation model workflows. Create a mechanism to aggregate and analyze model prediction endpoint metrics using services, such as Amazon OpenSearch Service. OpenSearch Service supports dashboards for visualization. Consider integrating third-party AI tools (Comet, Deepchecks, Fiddler AI, Lakera) for extended governance, bias detection, explainable AI, and vertical solutions. The traceability of hosting metrics back to versioned inputs allows for analysis of changes that could be impacting current operational performance.
4. **Establish data quality monitoring**. Configure SageMaker AI Model Monitor to track data quality metrics such as missing values, statistical outliers, and feature distribution shifts. Set up constraints that define acceptable ranges for these metrics and generate alerts when violations occur.
5. **Implement bias detection and tracking**. Use SageMaker AI Clarify to detect bias in your model predictions over time. Monitor for changes in fairness metrics across different segments of your data and create visualizations to track these metrics over time.
6. **Set up model explainability analysis**. Deploy SageMaker AI Clarify to track feature importance and SHAP values over time. These values can determine if the model's decision-making process is changing in unexpected ways that might indicate performance issues.
7. **Create a retraining pipeline**. Develop an automated pipeline that can retrain your models when performance degradation is detected. Use AWS Step Functions to orchestrate the retraining workflow, including data preparation, model training, evaluation, and deployment.
8. **Implement A/B testing for model updates**. When deploying updated models, use SageMaker AI's production variants to perform A/B testing between the current and new model versions. This allows you to validate performance improvements before fully replacing the existing model.

Resources

Related documents:

- Data and model quality monitoring with SageMaker AI Model Monitor
- Fairness and Explainability with SageMaker AI Clarify
- Amazon SageMaker AI Feature Store
- Monitoring in-production ML models at large scale using Amazon SageMaker AI Model Monitor
- ML model explainability with Amazon SageMaker AI Clarify and the SKLearn pre-built container

Related videos:

- Understand ML model predictions & biases with Amazon SageMaker AI Clarify
- Deep Dive on Amazon SageMaker AI Debugger & Amazon SageMaker AI Model Monitor
- Detect machine learning (ML) model drift in production

Related examples:

- Amazon SageMaker AI Model Monitor Examples - Github
- SageMaker AI Clarify Examples - Github

MLPERF06-BP05 Establish an automated re-training framework

Monitor data and model predictions to identify errors due to data and concept drift. By implementing automated model re-training at scheduled intervals or when performance metrics reach defined thresholds, you can maintain model accuracy and effectiveness over time. This approach keeps your machine learning models relevant as data patterns evolve.

Desired outcome: You can detect when your deployed ML models experience data drift or performance degradation, and automatically run retraining processes. You establish mechanisms to monitor data statistics and ML inferences in production, allowing you to maintain high-quality predictions without manual intervention. Your models are consistently updated with new data, and model versions are properly tracked to maintain traceability and reproducibility.

Common anti-patterns:

- Waiting for model performance to fail catastrophically before initiating retraining.

- Manually monitoring model performance without automated alerts or prompts.
- Retraining on a fixed schedule regardless of model performance or data patterns.
- Lacking proper version control for retrained models.
- Not maintaining consistent evaluation metrics across model versions.

Benefits of establishing this best practice:

- Maintains model accuracy and relevance as data patterns evolve.
- Reduces manual intervention required to keep models performing optimally.
- Enables quick response to data drift and concept drift.
- Creates a documented, repeatable process for model updates.
- Provides consistent model quality through automated evaluation.
- Maximizes return on investment for machine learning solutions.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Establishing an automated retraining framework is crucial for maintaining ML model performance over time. As new data becomes available or as the underlying patterns in your data change, your models can drift and become less accurate. By implementing a systematic approach to model monitoring and retraining, you can verify that your ML solutions continue to deliver business value.

Avoid waiting for model performance to fail catastrophically before initiating retraining. Many organizations manually monitor model performance without automated alerts or prompts, retrain on a fixed schedule regardless of model performance or data patterns, lack proper version control for retrained models, and don't maintain consistent evaluation metrics across model versions.

Start by defining clear performance metrics for your models that align with your business objectives. These metrics should be continuously monitored in production to detect performance degradation. Additionally, monitor your input data for statistical changes that may indicate drift from the training distribution. When changes are detected, your automated framework should run retraining workflows.

The process should include data preparation, model training with both existing and new data, thorough evaluation, and controlled deployment. Each retrained model should be versioned appropriately to maintain traceability and allow for rollback if needed.

Implementation steps

1. **Define model performance metrics**. Establish clear metrics that measure how well your model is performing relative to business objectives. These could include accuracy, precision, recall, F1 score, or custom domain-specific metrics. Verify that these metrics can be calculated automatically and regularly in your production environment.
2. **Configure monitoring systems**. Use Amazon SageMaker AI Model Monitor to continuously monitor the quality of your ML models in production. Set up data quality monitoring to detect drift in input features, model quality monitoring to track prediction quality, bias drift monitoring to detect changes in fairness metrics, and feature attribution drift to identify changes in feature importance.
3. **Establish retraining prompts**. Define the conditions that will initiate model retraining. These can include scheduled intervals based on business requirements, performance degradation beyond defined thresholds, detection of data drift above acceptable limits, and availability of new training data. Set up Amazon CloudWatch alerts to notify or automatically run retraining workflows.
4. **Design retraining pipelines**. Create automated pipelines using Amazon SageMaker AI Pipelines that handle the entire retraining workflow, including data preparation, feature engineering, model training, evaluation, and deployment. For large-scale foundation model training or distributed workloads, leverage Amazon SageMaker AI HyperPod which provides managed, resilient high-performance clusters with automatic health checks and PyTorch auto-resume capabilities for long-running training jobs. In your pipeline, include steps for validation against holdout data before deployment.
5. **Implement model versioning**. Use Amazon SageMaker AI Model Registry to track and manage different versions of your models. As a result, you can recreate a model version if needed and provide traceability for your deployed models. Associate metadata with each version to document training data, hyperparameters, and performance metrics.
6. **Automate data processing for new training data**. Set up automated data processing workflows that prepare new data for training. Configure Amazon S3 event notifications to run Lambda functions or AWS Step Functions workflows when new data becomes available. Use Amazon SageMaker AI Feature Store to manage features consistently across training and inference.
7. **Set up orchestration**. Use AWS Step Functions Data Science SDK for SageMaker AI to orchestrate complex ML workflows. Define each step in the workflow and configure alerts to initiate the process. For detecting new training data, combine AWS CloudTrail with Amazon CloudWatch Events to automatically start Step Function workflows.

8. **Implement deployment safeguards**. Use deployment techniques like blue-green deployment or canary releases to safely transition to new model versions. Monitor the performance of new models closely during initial deployment and configure automatic rollback if performance degrades.
9. **Create feedback loops**. Establish mechanisms to collect ground truth data from production to continually evaluate and improve your models. This might involve user feedback, delayed outcomes, or manual labeling processes for a subset of predictions.
10. **Document the retraining process**. Create comprehensive documentation for your retraining framework, including prompts, pipelines, evaluation criteria, and deployment strategies. This process fosters knowledge transfer and consistent application of the process.

Resources

Related documents:

- Pipelines
- Amazon SageMaker AI HyperPod
- Retraining Models on New Data
- Data and model quality monitoring with SageMaker AI Model Monitor
- Train a Machine Learning Model (using AWS Step Functions)
- Amazon SageMaker AI Feature Store
- Model Registration Deployment with Model Registry
- Best practices and design patterns for building machine learning workflows with Amazon SageMaker AI Pipelines
- Create SageMaker AI Pipelines for training, consuming and monitoring your batch use cases
- Launch Amazon SageMaker AI Autopilot experiments directly from within Amazon SageMaker AI Pipelines to easily automate MLOps workflows

Related videos:

- Automating Machine Learning Workflows: Leveraging Amazon SageMaker AI Pipelines and Autopilot for Efficient Model Development and Deployment

Related examples:

- Amazon SageMaker AI MLOps Immersion Day

- Amazon SageMaker AI MLOps

MLPERF06-BP06 Review for updated data and features for retraining

Establishing a framework to regularly review and update your machine learning model's data and features is essential for maintaining model accuracy. As business environments evolve, new data patterns emerge that can impact your model's performance. By systematically reviewing your data and features at appropriate intervals, you can keep your models accurate and reliable.

Desired outcome: You establish a systematic approach to monitor data changes, explore new features, and incorporate updated data into your models. Through regular data exploration and feature engineering, you maintain model accuracy even as underlying data patterns evolve. This creates a proactive rather than reactive approach to model maintenance and verifies that your ML solutions consistently deliver business value.

Common anti-patterns:

- Assuming that data patterns remain stable over time.
- Retraining models only when performance degrades.
- Failing to explore new potential features as business evolves.
- Using the same feature engineering approach regardless of changing data characteristics.
- Not establishing regular review schedules for data and feature updates.

Benefits of establishing this best practice:

- Improved model accuracy through updated training data and features.
- Early detection of data drift and proactive model updates.
- Continuous discovery of new, potentially valuable features.
- Consistent model performance despite changing business conditions.
- Extended model lifecycle and reduced need for complete rebuilds.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Data is the foundation of a machine learning model, and its characteristics can change over time due to various factors such as seasonal variations, market shifts, or changes in customer behavior.

Without a framework to regularly review and update your data and features, models can gradually become less accurate as they fail to account for these changes.

Avoid assuming that data patterns remain stable over time. Many organizations retrain models only when performance degrades, fail to explore new potential features as their business evolves, and use the same feature engineering approach regardless of changing data characteristics.

To implement this practice effectively, you need to understand the volatility of your business environment and establish appropriate review intervals. For example, retail businesses might need more frequent reviews during holiday seasons when consumer behavior changes rapidly. You also need tools to efficiently explore data, identify new patterns, and engineer features that capture these insights.

Amazon SageMaker AI provides comprehensive capabilities for data preparation, feature engineering, and model monitoring. By using these tools, you can create an efficient pipeline for regularly reviewing and updating your model's data and features, providing continued accuracy and relevance.

Implementation steps

1. **Assess data volatility in your business environment**. Analyze how quickly your business data changes by examining historical data patterns and identifying seasonal trends, market shifts, or other factors that affect your data. This assessment can determine how frequently you need to review your model's data and features.
2. **Establish a review schedule**. Based on your data volatility assessment, create a calendar for regular data and feature reviews. For highly volatile environments, you may need monthly reviews, while more stable contexts might require quarterly or biannual reviews.
3. **Set up data monitoring**. Implement Amazon SageMaker AI Model Monitor to continuously track data drift by comparing production data against your model's training data. Configure alerts when deviations occur to run expedited reviews.
4. **Create a data exploration workflow with Amazon SageMaker AI Canvas**. Use Amazon SageMaker AI Canvas to build data exploration workflows. The unified SageMaker AI Studio environment provides seamless integration with S3, Redshift, and EMR for comprehensive data exploration, engineering, training, and deployment workflows. Canvas now includes enhanced no/low-code ML tools with templates, automation, and wizards that enable non-engineering users to train custom models for verticals like sales, fraud, and demand with minimal technical expertise. These workflows should include data visualizations, statistical analyses, and data quality assessments.

5. **Implement feature engineering processes**. Develop standardized feature engineering pipelines in SageMaker AI Data Wrangler that can transform raw data into model features. Include steps to identify potential new features during each review cycle.
6. **Integrate with SageMaker AI Feature Store**. Store engineered features in Amazon SageMaker AI Feature Store to maintain feature consistency between training and inference. This creates a single source of truth for features and simplifies retraining with updated data.
7. **Establish an evaluation framework**. Create a systematic approach to compare model performance using original features versus updated or new features. This quantifies the impact of feature changes and supports data-driven decisions about model updates.
8. **Form a cross-functional review team**. Assemble a team including data scientists, domain experts, and business stakeholders who can collectively evaluate data changes, validate new features, and authorize model retraining when necessary.
9. **Document changes and maintain version control**. Track changes to data sources, feature definitions, and transformation logic using version control systems. This creates an audit trail and supports reproducibility.
10. **Automate the retraining pipeline**. Use Amazon SageMaker AI Pipelines to create automated workflows that can retrain models with updated data and features when approved by the review team.

Resources

Related documents:

- Automate data preparation with Amazon SageMaker AI Canvas
- Data and model quality monitoring with SageMaker AI Model Monitor
- Create, store, and share features with Feature Store
- Pipelines
- No-code data preparation for time series forecasting using Amazon SageMaker AI Canvas

Related videos:

- Introducing Amazon SageMaker AI Canvas
- Automating Machine Learning Workflows with SageMaker AI Pipelines

Cost optimization

The cost optimization pillar includes the continual process of refinement and improvement of a system over its entire lifecycle. From the initial design of your very first proof of concept to the ongoing operation of production workloads, adopting the practices in this document can enable you to build and operate cost-aware systems that achieve business outcomes and minimize costs, thus allowing your business to maximize its return on investment.

Each best practice in this section is presented based on its place in the ML lifecycle as detailed in *Machine learning lifecycle*.

Lifecycle phases

- Business goal identification
- ML problem framing
- Data processing
- Model development
- Deployment
- Monitoring

Business goal identification

Best practices

- MLCOST01-BP01 Define overall return on investment (ROI) and opportunity cost
- MLCOST01-BP02 Use managed services to reduce total cost of ownership (TCO)

MLCOST01-BP01 Define overall return on investment (ROI) and opportunity cost

Machine learning projects require careful evaluation of their business value and resource requirements. By analyzing the ROI and opportunity costs of ML implementations, you can make informed decisions that optimize resource allocation while delivering maximum business impact.

Desired outcome: When you implement this practice, you have a clear understanding of the financial and business implications of your ML projects. You can differentiate between research-

oriented and development-oriented ML initiatives, track costs effectively through tagging mechanisms, and make data-driven decisions about resource allocation. You have established processes to continuously evaluate the cost-benefit ratio of ML initiatives as business conditions evolve, and your investments deliver measurable value while managing risks appropriately.

Common anti-patterns:

- Initiating ML projects without defining clear business objectives or expected outcomes.
- Failing to distinguish between research projects (long-term returns) and development projects (near-term returns).
- Not implementing cost tracking mechanisms for ML projects.
- Overlooking the ongoing operational costs of maintaining ML models in production.
- Failing to reassess the cost-benefit model when business conditions change.

Benefits of establishing this best practice:

- Improved allocation of limited resources to ML initiatives with highest potential returns.
- Clear visibility into project costs and benefits for better budgeting and planning.
- Reduced risk of project failure through upfront analysis and ongoing monitoring.
- Enhanced ability to communicate ML value to stakeholders.
- Accelerated time-to-value through focus on high-impact use cases.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Understanding the financial implications of machine learning initiatives is crucial for making strategic technology investments. Machine learning projects can vary significantly in terms of resource requirements, timeline to value, and overall business impact. By carefully evaluating the ROI and opportunity costs, you can prioritize initiatives that deliver the most significant business value while managing costs effectively.

Start by working with both technical and business teams to clearly define whether an ML project is research-oriented (focused on exploring potential future value) or development-oriented (applying established methods to deliver immediate business value). This distinction assists to set appropriate expectations around timelines, resources, and outcomes. Implement comprehensive cost tracking through tagging mechanisms to maintain visibility into project expenses across data engineering, model development, and production deployment phases.

When assessing ML project costs, consider both direct expenses (infrastructure, tools, services) and indirect costs (staff time, training requirements, maintenance). Factor in potential costs associated with data preparation, model accuracy, and production errors. Develop a comprehensive cost-benefit model that accounts for these elements while considering business-specific factors like competitive advantage and strategic positioning.

Implementation steps

1. **Specify the objectives of the ML project as research or development**. Work with both business stakeholders and data science teams to determine if your ML initiative is exploratory research with long-term returns or development applying established methods for faster ROI. Align between technical teams and business leaders on project classification, timelines, and expected outcomes.
2. **Use tagging to track costs by project and business unit**. Implement comprehensive tagging in your AWS environment using AWS Cost Categories and AWS Tagging strategies to allocate ML-related expenses to specific projects and business functions. Monitor these costs through AWS Cost Explorer to maintain clear visibility of ROI by project.
3. **Evaluate and assess the data pipeline, the ML model, and the expected quality of production inferences**. Analyze the infrastructure requirements, operational costs, and potential business impact of errors in your ML system. Use Amazon SageMaker AI Clarify to assess model quality and identify potential bias that could impact business outcomes and add remediation costs.
4. **Develop a cost-benefit model**. Create a comprehensive financial model that accounts for initial development costs, ongoing operational expenses, and expected business benefits. Regularly reassess this model as business conditions change or when considering new data sources. Use Quick to build dashboards tracking ML costs against business KPIs.
5. **Understand, evaluate, and monitor project risks**. Identify technical, operational, and business risks associated with your ML project. Establish monitoring systems to track these risks through development and production phases. Use Amazon CloudWatch to monitor technical metrics and AWS Budgets to track spending against forecasts.
6. **Estimate the cost of resources needed for production maintenance**. Calculate the ongoing expenses required to maintain your ML model in production, including data engineers, data scientists, infrastructure costs, and monitoring systems. Consider using AWS Application Cost Profiler to attribute costs accurately across your ML applications.
7. **Leverage enhanced cost tracking and optimization tools**. Use AWS Cost Anomaly Detection to automatically identify unusual spending patterns in your ML workloads and receive alerts for unexpected cost increases.

8. **Consider model selection trade-offs for generative AI projects**. When implementing generative AI solutions, carefully evaluate the balance between model size, performance, and cost. Smaller, domain-specific models may be more cost-effective than large foundation models for certain use cases. Consider using Amazon Bedrock for access to multiple foundation models through a single API, allowing for streamlined model selection and optimization.

Resources

Related documents:

- AWS Pricing Calculator
- What is AWS Billing and Cost Management and Cost Management?
- Optimizing cost for building AI models with Amazon EC2 and SageMaker AI
- Analyze Amazon SageMaker AI spend and determine cost optimization opportunities based on usage, Part 4: Training jobs
- AWS Application Cost Profiler
- Getting started with AWS Cost Anomaly Detection
- Managing costs with AWS Budgets
- AWS Cost Explorer
- AWS Cost and Usage Report
- Generative AI Cost Optimization Strategies

Related videos:

- Maximizing ML ROI: Amazon SageMaker AI's High-Performance Inference and Cost Optimization Strategies

MLCOST01-BP02 Use managed services to reduce total cost of ownership (TCO)

Using managed machine learning services enables organizations to operate more efficiently with reduced resources and costs compared to self-managed options. This approach reduces undifferentiated heavy lifting, reduces operational burden, and allows teams to focus on delivering business value.

Desired outcome: By adopting managed services and pay-per-usage models, you significantly reduce your total cost of ownership while gaining access to a comprehensive suite of AI/ML tools.

You can use pre-built capabilities instead of developing custom solutions, automatically scale resources based on demand, and benefit from AWS's continuous innovations without additional investment. Your teams can focus on creating business value rather than managing infrastructure.

Common anti-patterns:

- Building and maintaining custom ML infrastructure on EC2 or Kubernetes.
- Overprovisioning resources for peak ML workloads.
- Failing to use commitment discounts for persistent workloads.
- Developing proprietary AI services when managed services would suffice.
- Not analyzing workload patterns to optimize instance selection.

Benefits of establishing this best practice:

- Significantly lower total cost of ownership compared to self-managed options.
- Reduced operational overhead and simplified management.
- Increased team productivity with focus on core business problems.
- Access to continuously updated and improved AI/ML capabilities.
- Flexibility to scale resources based on actual demand.
- Ability to use commitment-based pricing for additional savings.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Managed services remove the operational burden of maintaining infrastructure, allowing you to concentrate on developing ML models and applications that drive business value. Using AWS's managed ML services provides a comprehensive environment for building, training, and deploying models with significantly lower costs than self-managed options.

When evaluating your ML strategy, consider the total cost of ownership including infrastructure, operational personnel, maintenance, scaling, and upgrades. Amazon SageMaker AI provides a fully managed service that avoids many of these costs while offering advanced ML capabilities. Similarly, AWS's pre-trained AI services can address common use cases without requiring ML expertise, further reducing implementation time and costs.

To maximize cost efficiency, analyze your workload patterns and determine which components would benefit from commitment discounts. By using Savings Plans, you can significantly reduce

your AWS usage costs while maintaining flexibility across instance families, sizes, regions, and components.

Implementation steps

1. **Use Amazon SageMaker AI as your fully managed ML solution.** Amazon SageMaker AI enables building, training, and deploying models at scale with significantly lower costs. The total cost of ownership (TCO) of SageMaker AI over a three-year period is much lower than other self-managed cloud-based ML options, such as Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Elastic Kubernetes Service (Amazon EKS). SageMaker AI includes technologies such as Autopilot, Feature Store, Clarify, Debugger, Studio, Training, Model deployment, Monitoring, and Pipelines.
2. **Use Amazon managed AI services for common use cases.** AWS pre-trained AI services provide ready-made intelligence for your applications and workflows. These services address common use cases such as personalized recommendations, contact center modernization, safety and security improvement, and customer engagement enhancement. They don't require machine learning expertise, are fully managed, and offer pay-as-you-go pricing with no upfront commitment.
3. **Perform pricing model analysis for cost optimization.** Analyze each component of your ML workload to determine if it will run for extended periods, making it eligible for commitment discounts such as AWS Savings Plans. You can use Savings Plans to reduce AWS usage costs by committing to a consistent amount of usage. Amazon SageMaker AI Savings Plans offer flexible attributes such as instance family, instance size, AWS Region, and component for your SageMaker AI instance usage.
4. **Implement right-sizing strategies for ML resources.** Evaluate your actual ML workload resource requirements and adjust instance types and sizes accordingly. This blocks overprovisioning and assists to control costs while maintaining performance. Use SageMaker AI's automatic scaling capabilities to match resources with demand.
5. **Use serverless options when appropriate.** For intermittent workloads or those with variable demand, consider serverless options like Amazon SageMaker AI Serverless Inference to avoid paying for idle resources.
6. **Use Amazon Bedrock for foundation model access.** Amazon Bedrock provides a unified API for accessing various foundation models, making it simple to experiment with and integrate generative AI capabilities without investing in model training infrastructure. This fully managed service assists to reduce costs while allowing flexibility to choose the right model for your use case.

7. **Use Foundation Model Hub for centralized model access**. Use the Foundation Model Hub to access a centralized catalog of popular foundation models with simplified deployment and performance benchmarking tools, reducing the time and cost of model selection and deployment.
8. **Use AI-powered code generation tools.** Use Amazon Q Developer and AI-powered IDEs like Kiro to accelerate ML development through AI-assisted coding, automated code generation, and intelligent troubleshooting, significantly reducing developer time and associated costs.

Resources

Related documents:

- Amazon managed AI services
- AWS AI Services overview
- ML Savings Plans
- AWS Pricing Calculator for SageMaker AI
- Viewing resource recommendations
- What is Amazon Bedrock?
- What is AWS Migration Hub?
- What are AWS Cost and Usage Reports?

ML problem framing

Best practices

- MLCOST02-BP01 Identify if machine learning is the right solution
- MLCOST02-BP02 Perform a tradeoff analysis between custom and pre-trained models

MLCOST02-BP01 Identify if machine learning is the right solution

Evaluating whether machine learning is the appropriate solution for your business problem is crucial for cost optimization. Not every problem requires ML solutions, and sometimes simpler approaches may be more effective and less costly. By thoroughly evaluating alternatives against ML approaches, you can make informed decisions that optimize both your technical resources and business outcomes.

Desired outcome: You identify whether machine learning is truly the optimal solution for your business problem by comparing it against simpler alternatives. You make informed decisions about resource allocation, understanding the cost implications of ML adoption including data preparation, storage, training, hosting, and maintenance. You validate your approach using tools like Amazon SageMaker AI Autopilot and Amazon SageMaker AI Clarify to verify that ML provides measurable benefits over alternative solutions.

Common anti-patterns:

- Jumping directly to ML solutions without evaluating simpler alternatives.
- Underestimating the total cost of implementing ML, including data preparation and maintenance.
- Failing to establish a baseline for comparison with existing or rules-based approaches.
- Overlooking specialized resource constraints such as data scientist availability or model time-to-market.

Benefits of establishing this best practice:

- Avoids unnecessary complexity and cost in solution design.
- Optimizes resource allocation based on actual business value.
- Reduces risk of project failure due to inappropriate technology selection.
- Provides quantifiable metrics for evaluating ML solution effectiveness.

Level of risk exposed if this best practice is not established: High

Implementation guidance

When considering machine learning for a business problem, start by thoroughly evaluating whether ML is truly necessary. Many problems can be effectively solved with simpler rule-based approaches that may be less expensive to develop and maintain. Machine learning requires significant investment in data preparation, specialized hardware, and ongoing maintenance that must be justified by the business value it delivers.

Begin by clearly articulating your problem and determining if it requires the adaptive learning capabilities that ML provides. Consider if the problem involves complex patterns that rules can't simply capture, or if it requires continuous adaptation to changing conditions. For example, fraud detection in financial transactions might benefit from ML due to constantly evolving fraudulent behaviors, while simple inventory management might be better served by a rules-based system.

Evaluate costs associated with an ML solution, including data preparation, storage, compute resources for training, potential data labeling, model hosting, and ongoing maintenance. Compare these costs against the business value gained from using ML versus alternative approaches. Remember that specialized resources like data scientists might be your most constrained resource, making their time allocation a critical consideration.

Implementation steps

1. **Articulate your problem clearly**. Define the business problem you're trying to solve, the desired outcomes, and how success will be measured. Be specific about what decisions need to be made and what data is available to support those decisions.
2. **Identify your data sources**. Evaluate what data you already have, what data you need to collect, and whether the quality and quantity are sufficient for ML applications. Consider Amazon SageMaker AI to catalog and manage your data assets.
3. **Calculate comprehensive cost implications**. Consider the aspects of implementing an ML solution:
   - Data preparation and engineering costs
   - Data storage requirements and associated costs using Amazon S3 or other storage services
   - Model training expenses on various hardware options in Amazon SageMaker AI Model Training
   - Data labeling costs if supervised learning is required
   - Potential retraining costs due to model drift or bias
   - Model hosting and inference costs
   - Ongoing maintenance and monitoring expenses
4. **Establish a baseline solution**. Evaluate how the problem is currently being solved or how it could be solved with a simpler approach. If a rules-based solution exists, use it as a baseline for comparison. For basic ML approaches, consider pre-built solutions from AWS Marketplace or Amazon SageMaker AI JumpStart.
5. **Build and evaluate an ML prototype**. Use Amazon SageMaker AI or Amazon SageMaker AI Autopilot to quickly develop an ML model. Compare the performance metrics of this solution against your baseline approach, including accuracy, inference time, and total cost of operation.
6. **Analyze model explainability**. Use Amazon SageMaker AI Clarify to understand how your ML model makes decisions and evaluate if these explanations align with business expectations and requirements.
7. **Make a data-driven decision**. Based on your comparative analysis, determine if the ML approach demonstrates sufficient improvement over simpler solutions to justify the investment. Consider both quantitative metrics and qualitative factors like flexibility and scalability.

8. **Use no-code ML for rapid validation**. Use SageMaker AI Canvas with natural language support to quickly validate whether ML approaches provide value over simpler solutions, reducing the time and cost of initial feasibility assessment. Export Canvas-generated models and code to notebooks for further customization and integration into production workflows.
9. **Use AI-powered code generation for rapid prototyping**. Use AI-powered development tools like Amazon Q Developer and Kiro to quickly generate ML prototype code, automate data preprocessing scripts, and accelerate the validation process for determining if ML is the right solution.
10. **Assess hybrid approaches**. Consider whether combining rules-based systems with ML or generative AI could provide the optimal balance of cost, performance, and explainability for your specific use case.

Resources

Related documents:

- Amazon SageMaker AI Canvas
- SageMaker AI autopilot
- Amazon SageMaker AI JumpStart
- Machine Learning solutions in AWS Marketplace
- Cost Optimization Pillar - AWS Well-Architected Framework
- What is Amazon SageMaker AI?
- What is Amazon Bedrock?

MLCOST02-BP02 Perform a tradeoff analysis between custom and pre-trained models

Optimize machine learning costs by carefully analyzing the tradeoffs between developing custom models and using pre-trained models. This analysis should maintain security and performance efficiency within acceptable thresholds while minimizing unnecessary expenses.

Desired outcome: You achieve optimal cost efficiency in your machine learning initiatives by making informed decisions about when to use pre-trained models versus developing custom solutions. You balance development costs, time-to-market, model performance, and specific business requirements while maintaining appropriate security standards. This strategic approach allows you to accelerate ML development while optimizing your investment in AI/ML resources.

Common anti-patterns:

- Building custom models for every use case without considering available pre-trained alternatives.
- Using pre-trained models without evaluating if they meet your specific business requirements.
- Ignoring the total cost of ownership including data scientist time, infrastructure, and ongoing maintenance.
- Overlooking security and compliance requirements when selecting pre-trained models.

Benefits of establishing this best practice:

- Reduced time-to-market for ML solutions.
- Lower development and operational costs.
- Ability to use state-of-the-art models without needing extensive expertise.
- More efficient use of data scientist and ML engineer resources.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

When implementing machine learning solutions, the decision between building custom models and using pre-trained ones significantly impacts costs, time-to-market, and solution effectiveness. Custom models offer greater flexibility and potential performance advantages for specialized tasks but require substantial investment in data collection, training infrastructure, and expertise. Pre-trained models provide rapid deployment and reduced initial costs but may not perfectly align with specific business needs.

Your analysis should consider factors including data availability, task specificity, performance requirements, available expertise, and long-term maintenance costs. For many common use cases like sentiment analysis, image classification, or document processing, pre-trained models can deliver excellent results without the overhead of custom development. For highly specialized domains or when competitive advantage depends on model performance, custom development may justify the additional investment.

Implementation steps

1. **Assess your machine learning needs**. Begin by clearly defining your business use case, required model accuracy, latency requirements, and available data. Understand whether your use case is standard (for example, image classification or sentiment analysis) or highly specialized to guide your decision-making process.

2. **Use Amazon SageMaker AI built-in algorithms and AWS Marketplace**. Amazon SageMaker AI provides a suite of built-in algorithms for data scientists and machine learning practitioners to get started on training and deploying machine learning models. Pre-trained ML models are ready-to-use models that can be quickly deployed on Amazon SageMaker AI. By pre-training the ML models for you, solutions in the AWS Marketplace take care of the heavy lifting so that you can deliver AI- and ML-powered features faster and at a lower cost. Evaluate the cost of your data scientists' time and other resource requirements to develop your own custom model vs. bringing a pre-trained model and deploying it on SageMaker AI for inferencing. The advantage of a custom model is the flexibility to fine-tune it to match the needs of your business use case. A pre-trained model can be difficult to modify and you might have to use it as is.
3. **Use Amazon SageMaker AI JumpStart**. Use Amazon SageMaker AI JumpStart to access pre-trained models and accelerate the ML development process. SageMaker AI JumpStart provides a set of solutions for the most common use cases that can be deployed readily with just a few clicks. The solutions are fully customizable and showcase the use of AWS CloudFormation templates and reference architectures so you can accelerate your ML journey. Amazon SageMaker AI JumpStart also supports one-click deployment and fine-tuning of more than 150 popular open-source models such as natural language processing, object detection, and image classification models.
4. **Conduct a cost-benefit analysis**. Calculate the total cost of ownership for both custom and pre-trained approaches, including development time, infrastructure costs, and ongoing maintenance. Consider factors such as data preparation, training resources, and the expertise required. Compare these costs against expected business value and performance requirements to determine the most cost-effective approach.
5. **Implement cost monitoring and optimization**. Use AWS Cost Explorer and AWS Budgets to monitor and manage your ML workload costs. Implement automatic shutdown of idle resources to reduce unnecessary expenses. Consider using AWS Compute Optimizer to get cost optimization recommendations for your ML infrastructure.
6. **Explore model customization options**. When pre-trained models don't fully meet your requirements, explore customization options like fine-tuning or transfer learning before committing to full custom development. This approach can provide a middle ground between cost and performance and access to existing models while adapting them to your specific needs.
7. **Implement a multi-model approach**. For complex use cases, consider using different models for different components of your solution based on their requirements. This allows you to

optimize costs by using simpler, more economical models where appropriate while reserving more powerful models for tasks that require them.

8. **Evaluate foundation models in Amazon Bedrock**. Amazon Bedrock provides a fully managed service that offers foundation models from leading AI companies through a single API. Consider using these models for text, image, and multimodal generative AI applications instead of building custom models. You can customize these models to your specific needs using retrieval-augmented generation (RAG) or fine-tuning while maintaining cost efficiency.
9. **Use expanded pre-trained model libraries**. Use the expanded SageMaker AI JumpStart catalog which now includes broader selection of pre-trained models and industry-specific solutions, reducing the need for custom model development and associated costs.
10. **For generative AI workloads, consider retrieval-augmented generation (RAG)**. For many generative AI applications, implementing RAG can enhance the performance of foundation models by providing them with relevant context from your organization's data. This approach can be more cost-effective than fine-tuning and still provide customized outputs tailored to your business domain.

Resources

Related documents:

- Amazon SageMaker AI JumpStart
- Pre-trained machine learning models available in AWS Marketplace
- Amazon SageMaker AI pricing
- Cloud Financial Management with AWS

Data processing

Best practices

- MLCOST03-BP01 Use managed data labeling
- MLCOST03-BP02 Use no-code or low-code and code generation tools for interactive analysis
- MLCOST03-BP03 Use managed data processing capabilities
- MLCOST03-BP04 Enable feature reusability

MLCOST03-BP01 Use managed data labeling

Use managed labeling tools that provide automation and access to cost-effective teams of human data labelers. These tools should offer flexibility to choose a variable number of labelers for each input, include a user-friendly interface, and incorporate learning capabilities to improve labeling efficiency over time.

Desired outcome: You have access to high-quality labeled datasets for your machine learning models without building and managing your own labeling infrastructure. Your data labeling process is streamlined, cost-efficient, and scales according to your needs, allowing you to focus on model development rather than data preparation logistics.

Common anti-patterns:

- Building custom data labeling infrastructure from scratch.
- Relying solely on in-house teams for labeling tasks regardless of scale.
- Using labeling solutions that don't improve through machine learning.
- Managing inconsistent labeling quality without proper oversight tools.

Benefits of establishing this best practice:

- Reduce time-to-market for ML models by accelerating the data labeling process.
- Lower total labeling costs through efficient automation and on-demand workforce.
- Improve labeling quality and consistency through specialized tools and workflows.
- Scale labeling operations up or down based on project requirements.
- Focus your team's effort on model development rather than labeling infrastructure.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

To build effective machine learning models, you need large, high-quality labeled datasets. Creating these datasets manually is time-consuming, expensive, and difficult to scale. By using managed data labeling services, you can accelerate this critical step in the ML development process while controlling costs and maintaining quality.

Managed data labeling combines human intelligence with machine learning to improve efficiency over time. As your models process more data, they can begin to automate parts of the labeling

process, reducing costs and time required. These services also provide quality control mechanisms through consensus models, where multiple labelers evaluate the same data to check accuracy.

When selecting a managed data labeling solution, consider factors like the types of data you need to label (like images, text, and video), the complexity of your labeling tasks, integration with your existing ML workflow, and cost structure. The right solution will scale with your needs and provide consistent, high-quality labeled data.

Implementation steps

1. **Assess your data labeling requirements**. Define what types of data you need labeled (images, text, audio, or video), the complexity of annotations required, expected volume, and quality standards. Determine whether you need specialized domain expertise for your labeling tasks.
2. **Use Amazon SageMaker Ground Truth**. To train a machine learning model, you need a large, high-quality, labeled dataset. Amazon SageMaker Ground Truth assists you to build high-quality training datasets for your ML models. With Ground Truth, you can use ML along with workers from a vendor company that you choose, or an internal, private workforce to create a labeled dataset. You can use the labeled dataset output from Ground Truth to train your own models. You can also use the output as a training data set for an Amazon SageMaker AI model.
3. **Use Amazon SageMaker Ground Truth Plus**. Ground Truth Plus is a turn-key service that uses an expert workforce to deliver high-quality training datasets fast, and reduces costs by up to 40 percent. Amazon SageMaker Ground Truth Plus enables you to create high-quality training datasets without having to build labeling applications and manage the labeling workforce on your own. By using this approach, you don't need to have deep ML expertise or extensive knowledge of workflow design and quality management. You simply provide data along with labeling requirements and Ground Truth Plus sets up the data labeling workflows and manages them on your behalf in accordance with your requirements.
4. **Configure active learning workflows**. Set up your labeling projects to use active learning, where the system learns from human annotations and begins to automate labeling for similar items. This reduces the number of items requiring manual labeling over time, improving efficiency and reducing costs. Amazon SageMaker Ground Truth provides built-in support for active learning.
5. **Implement quality control mechanisms**. Configure your labeling jobs to use multiple workers per data item and determine consensus approaches based on your quality requirements. Monitor labeling performance and adjust your quality control parameters as needed.
6. **Set up real-time data labeling pipelines**. For ongoing ML projects, establish continuous data labeling pipelines that can process new data as it becomes available. This way, your models can be regularly retrained with fresh data.

7. **Create custom labeling interfaces when needed**. For specialized labeling tasks, use Ground Truth's custom template capabilities to create tailored labeling interfaces that make the process more efficient for your specific use case.
8. **Use enhanced Ground Truth capabilities**. Use improved Ground Truth Plus features that provide up to 40% cost reduction through expert workforce management and automated quality control mechanisms.
9. **Use foundation models for pre-labeling**. Use generative AI models through Amazon Bedrock to assist with initial data labeling, which can then be verified by human labelers. This hybrid approach can significantly accelerate the labeling process while maintaining quality control.

Resources

Related documents:

- Training data labeling using humans with Amazon SageMaker Ground Truth
- Use Amazon SageMaker Ground Truth Plus to Label Data
- Using the Amazon Mechanical Turk Workforce
- Automate data labeling
- Custom labeling workflows
- Annotation consolidation
- Ground Truth streaming labeling jobs
- Implementing a custom labeling GUI with built-in processing logic with Amazon SageMaker Ground Truth

Related examples:

- Bring your own model for SageMaker AI labeling workflows with active learning
- SageMaker AI Ground Truth recipe

MLCOST03-BP02 Use no-code or low-code and code generation tools for interactive analysis

Prepare data through data wrangler tools for interactive data analysis and model building. The no-code/low-code, automation, and visual capabilities improve productivity and reduce the cost for interactive analysis. Integrate with generative AI code generation tools.

Desired outcome: You will be able to streamline your data preparation workflow using visual interfaces with minimal coding required. By implementing no-code or low-code tools like Amazon SageMaker AI Canvas and Data Wrangler, you reduce time spent on data preprocessing tasks and gain improved insights through interactive visualizations. Amazon Q integration provides intelligent assistance for data preparation and code generation, enabling faster iteration cycles on model development while maintaining data quality and consistency across your machine learning projects.

Common anti-patterns:

- Writing custom data preparation scripts for every analysis task.
- Using disjointed tools for data import, transformation, and visualization.
- Manually performing repetitive data cleaning operations.
- Creating non-reproducible data preparation workflows.

Benefits of establishing this best practice:

- Reduced time and cost for data preparation and feature engineering.
- Improved productivity through visual interfaces and automation.
- Streamlined workflow from data import to model deployment.
- Support for code and no-code approaches to accommodate different skill levels.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Data preparation is often cited as the most time-consuming aspect of machine learning projects, typically consuming 60-80% of data scientists' time. Data wrangler tools provide visual interfaces to simplify and accelerate this process through automation and low-code solutions.

Amazon SageMaker AI Data Wrangler offers an end-to-end solution for data preparation that integrates directly with your machine learning workflow. By using a visual interface, you can import data from various sources, identify and fix data quality issues, transform features, and generate insights—all with minimal coding required. The tool provides transparency by generating code for your transformations, fostering reproducibility and allowing customization when needed.

Data wrangler tools are particularly valuable for exploratory data analysis, where quick iteration and visualization are essential. They allow you to rapidly identify patterns, outliers, and

relationships in your data, accelerating the feature engineering process. With built-in data quality and insights features, you can understand your data characteristics and address issues before model training begins.

Implementation steps

1. **Set up Amazon SageMaker AI Canvas or Studio environment**. Access SageMaker AI Canvas for a no-code experience or SageMaker AI Studio for more advanced capabilities through the AWS Management Console. Canvas provides a visual, drag-and-drop interface for business analysts and citizen data scientists, while Studio offers Data Wrangler for more technical users. Both environments support the complete machine learning workflow with varying levels of coding requirements.
2. **Import data from various sources**. Use SageMaker AI Canvas or Data Wrangler to connect to multiple data sources including Amazon S3, Amazon Athena, Amazon Redshift, Snowflake, and various databases. Canvas provides a simplified point-and-click interface for business users, while Data Wrangler offers more advanced data source connectivity options. Both tools avoid the need for custom connector code.
3. **Explore and visualize your data**. USe Data Wrangler's built-in data visualizations to understand distributions, correlations, and outliers. These visualizations assist to identify potential issues early and inform feature engineering decisions without writing complex plotting code.
4. **Use Amazon Q for generative AI-powered data preparation and code generation**. Use Amazon Q integrated within SageMaker AI Canvas and Data Wrangler to get natural language assistance for data preparation tasks, automated code generation, and intelligent suggestions for data transformations. Amazon Q can explain data patterns, suggest optimal preprocessing steps, and generate code snippets for custom transformations, significantly reducing the time needed for data preparation tasks. Additionally, use AI-powered development tools like Kiro for intelligent code generation and optimization of your data processing workflows.
5. **Apply transformations to prepare your data**. Use the visual transformation interface to clean and prepare data through operations like handling missing values, encoding categorical features, scaling numerical values, and feature extraction. Data Wrangler provides over 300 built-in transformations while allowing custom Python transformations when needed.
6. **Analyze data quality and generate insights**. Use the built-in data quality and insights features to detect anomalies, check for imbalanced data, and understand feature importance. These automated analyses identify potential issues before model training begins.
7. **Balance your datasets**. Address imbalanced datasets using built-in techniques like random oversampling, random undersampling, and synthetic minority oversampling (SMOTE).

Data Wrangler provides visual controls to implement these techniques without specialized knowledge.

8. **Scale to larger datasets**. Process larger datasets by configuring instance types and using distributed processing capabilities. Data Wrangler supports processing wide datasets with thousands of columns and large datasets with billions of rows through appropriate resource allocation.
9. **Prepare time series data**. Use specialized time series transformations to handle temporal data, including resampling, lagged feature creation, and time-based aggregations. These operations simplify working with sequential data patterns.
10. **Export your data flow for production**. Deploy your data preparation workflow by exporting to various destinations including Amazon S3, SageMaker AI Feature Store, or directly to model building workflows. Data Wrangler generates Python code that can be integrated into production pipelines. Canvas workflows can also be exported to SageMaker AI notebooks for further customization and integration into production pipelines.
11. **Use enhanced Canvas capabilities**. Use SageMaker AI Canvas's improved natural language support and Q integration for conversational data analysis, enabling business users to perform complex data preparation tasks without technical expertise.
12. **Integrate with the broader machine learning workflow**. Connect your prepared data directly to SageMaker AI's model building capabilities like SageMaker AI Autopilot for automated model development or custom model training. This integration creates a seamless path from data to deployed models.

Resources

Related documents:

- Amazon SageMaker AI Canvas
- Get Started with Data Wrangler
- Prepare ML Data with Amazon SageMaker AI Data Wrangler
- What is Amazon Q Developer?
- What is AWS Glue DataBrew?
- Create, store, and share features with Feature Store
- Accelerate data preparation with data quality and insights in Amazon SageMaker AI Data Wrangler
- Process larger and wider datasets with Amazon SageMaker AI Data Wrangler
- Fuel Your Data with Generative AI

Related examples:

- Prepare ML Data with Amazon SageMaker AI Data Wrangler
- SageMaker AI Data Wrangler Examples GitHub Repository

MLCOST03-BP03 Use managed data processing capabilities

With managed data processing, you can use a simplified, managed experience to run your data processing workloads, such as feature engineering, data validation, model evaluation, and model interpretation.

Desired outcome: By implementing managed data processing capabilities, you can streamline your machine learning workflow with fully managed infrastructure for data preprocessing and postprocessing tasks. You gain the ability to run processing jobs that integrate with popular frameworks while maintaining operational efficiency, allowing your team to focus on creating valuable ML models rather than managing infrastructure.

Common anti-patterns:

- Building and maintaining custom data processing infrastructure.
- Managing your own compute clusters for data processing tasks.
- Manually handling scaling, deployment, and cleanup of processing resources.
- Using inconsistent processing environments across development and production.

Benefits of establishing this best practice:

- Reduced operational overhead with fully managed infrastructure.
- Simplified integration with popular ML frameworks and AWS services.
- Enhanced productivity by focusing on ML development rather than infrastructure management.
- Seamless integration with other SageMaker AI capabilities.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Amazon SageMaker AI Processing provides a managed solution for running these data processing workloads. Instead of provisioning and managing your own infrastructure, SageMaker AI handles the provisioning, scaling, and cleanup of resources. Processing jobs accept data from Amazon S3

as input and store processed results back to S3 as output. You can use AWS-provided container images that come pre-configured with popular data science frameworks, or you can bring your own custom containers for specialized processing needs.

By using SageMaker AI Processing, you can integrate data processing steps seamlessly into your ML pipelines and create consistency between development and production environments while reducing operational overhead. This allows your data scientists and ML engineers to focus on extracting insights from data rather than managing infrastructure.

Implementation steps

1. **Set up your processing job environment**. Create an Amazon SageMaker AI notebook instance or Studio environment from which you'll configure and launch your processing jobs. This provides an interactive environment for development and testing of your data processing scripts before scaling to larger datasets.
2. **Select or create a processing container**. Choose from SageMaker AI's built-in processing containers for frameworks like scikit-learn, PyTorch, TensorFlow, or Apache Spark. Alternatively, create a custom Docker container if you have specialized framework requirements. The container will include the runtime environment and dependencies needed for your processing tasks.
3. **Prepare your processing script**. Develop a script that runs within the processing container to perform your data transformation, feature engineering, model evaluation, or other processing tasks. This script should read input data, process it according to your requirements, and write output to the designated locations.
4. **Configure storage locations**. Set up Amazon S3 buckets to store your input data, processing scripts, and output results. SageMaker AI Processing jobs use S3 as the primary storage mechanism for exchanging data between steps in your ML workflow.
5. **Launch a processing job**. Use the SageMaker AI Python SDK or AWS console to configure and start your processing job. Specify parameters such as instance type, instance count, environment variables, and input and output configurations. SageMaker AI will provision the requested resources, run your processing script, and then automatically clean up the resources when the job completes.
6. **Monitor job progress and analyze results**. Track your processing job through the SageMaker AI console or API. Review logs to debug issues. Once completed, access the processed data in the specified S3 output locations for use in subsequent ML workflow steps.
7. **Integrate with ML pipelines**. Incorporate your processing jobs into SageMaker AI Pipelines to create automated end-to-end ML workflows. This enables you to orchestrate data preprocessing, model training, evaluation, and deployment steps in a repeatable manner.

8. **Optimize resource utilization and costs**. Review processing job metrics to identify opportunities for optimizing instance selection and parallelization strategies. Consider using Spot instances for cost savings on non-time-sensitive processing jobs.
9. **Use enhanced processing capabilities**. Use SageMaker AI Processing with better integration to popular ML frameworks and enhanced monitoring capabilities for more efficient data processing workflows.
10. **Use AI-powered code generation for data processing**. Use AI-powered development tools like Amazon Q Developer and Kiro to generate data processing scripts, automate pipeline creation, and accelerate the development of custom data transformation workflows.
11. **Implement data validation and quality checks**. Incorporate data validation steps in your processing jobs to check data quality before model training. Use SageMaker AI Clarify within processing jobs to detect bias in your datasets and implement model explainability.

Resources

Related documents:

- Data transformation workloads with SageMaker AI Processing
- CreateProcessingJob
- Managed Spot Training in Amazon SageMaker AI
- Amazon SageMaker AI Feature Store
- Amazon SageMaker AI Data Wrangler

Related examples:

- Amazon SageMaker AI Processing jobs
- SageMaker AI Processing with Spark

MLCOST03-BP04 Enable feature reusability

Reduce duplication and the rerunning of feature engineering code across teams and projects by using feature storage. The store should have online and offline storage, and data encryption capabilities. An online store with low-latency retrieval capabilities is ideal for real-time inference. An offline store maintains a history of feature values and is suited for training and batch scoring.

Desired outcome: You gain a centralized repository for storing, sharing, and managing machine learning features that reduces redundant work across teams and projects. You access features with low latency for real-time applications while maintaining a historical record for training purposes. Your feature store integrates seamlessly with your ML workflows, enhancing collaboration and accelerating model development while maintaining data security through robust encryption.

Common anti-patterns:

- Recreating the same features repeatedly across different teams and projects.
- Storing features in isolated data silos that avoid reuse.
- Lacking version control for features, leading to inconsistencies between training and inference.
- Using separate systems for real-time and batch feature access.
- Implementing homegrown feature storage solutions that lack scalability and proper governance.

Benefits of establishing this best practice:

- Reduces redundant work and computational costs.
- Creates consistency between training and inference environments.
- Enables collaboration and knowledge sharing across teams.
- Provides feature governance, lineage, and traceability.
- Reduces time to production for ML models.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Feature engineering is often one of the most time-consuming aspects of machine learning development. When teams work in silos, they frequently recreate the same features, wasting valuable time and resources. By implementing a centralized feature store, you create a single source of truth for ML features that promotes reusability across your organization.

A well-designed feature store addresses the dual requirements of offline storage for training and batch inference and online storage for low-latency real-time inference. This dual-storage paradigm creates consistency between training and serving environments while optimizing for different access patterns. The feature store should also provide capabilities for feature versioning, access control, and monitoring to maintain data quality and governance.

Amazon SageMaker AI Feature Store offers these capabilities as a fully managed service, which reduces the need to build and maintain complex infrastructure. It seamlessly integrates with

your ML pipelines and supports both batch and real-time inference workflows, making it an ideal solution for feature reusability.

Implementation steps

1. **Identify common features across projects**. Begin by analyzing your existing ML workflows to identify frequently used features that would benefit from centralization. Look for redundancies in feature engineering code across different teams and prioritize these for migration to the feature store.
2. **Set up Amazon SageMaker AI Feature Store**. Create feature groups in Amazon SageMaker AI Feature Store to organize related features. Define the schema for each feature group, including feature names, data types, and primary keys. Consider the access patterns for both training and inference when designing your feature groups.
3. **Configure storage options based on requirements**. Determine whether each feature group needs online storage, offline storage, or both. Configure the appropriate storage options:
   - **Online store:** Set up for low-latency access (milliseconds) needed for real-time inference
   - **Offline store:** Configure Amazon S3 storage for training and batch inference workloads
   - **Online and offline:** Implement both for maximum flexibility
4. **Implement data ingestion pipelines**. Develop automated pipelines to ingest data into your feature store. You can use Amazon SageMaker AI Data Wrangler for data preparation and Amazon SageMaker AI Pipelines for orchestration.
5. **Establish feature access patterns**. Create standardized methods for retrieving features for both training and inference. For training, use the offline store with Amazon Athena queries to efficiently access historical data. For real-time inference, implement API calls to the online store for low-latency feature retrieval.
6. **Enable cross-account and cross-team sharing**. Configure resource policies to enable feature sharing across different teams and AWS accounts. This promotes collaboration and maximizes feature reuse across your organization while maintaining appropriate access controls.
7. **Implement feature versioning and lineage tracking**. Track changes to features over time using versioning capabilities. Link features to models through Amazon SageMaker AI Model Registry to maintain full lineage tracking from data source to deployed model.
8. **Monitor feature usage and drift**. Implement monitoring for your feature store to detect data drift and track feature usage patterns. Use Amazon SageMaker AI Model Monitor to detect changes in feature distributions that might impact model performance.

9. **Create documentation and discovery mechanisms**. Document features and their intended use cases to facilitate discovery and reuse. Implement tagging and search capabilities so that data scientists can find relevant features for their projects.
10. **Use enhanced Feature Store capabilities**. Use improved SageMaker AI Feature Store with better performance, enhanced monitoring capabilities, and improved integration with other SageMaker AI services for more efficient feature management.
11. **Use generative AI for feature discovery and documentation**. Use large language models through Amazon Bedrock to automatically generate feature descriptions, identify potential feature relationships, and improve feature discoverability through natural language search capabilities.

Resources

Related documents:

- Create, store, and share features with Feature Store
- Amazon SageMaker AI Feature Store resources
- Understanding the key capabilities of Amazon SageMaker AI Feature Store

Related examples:

- Amazon SageMaker AI Feature Store Notebook Examples

Related videos:

- Training and Tuning State-of-the-Art Models with Amazon SageMaker AI

Model development

Best practices

- MLCOST04-BP01 Select optimal computing instance size
- MLCOST04-BP02 Use managed build environments
- MLCOST04-BP03 Select local training for small scale experiments
- MLCOST04-BP04 Select an optimal ML framework
- MLCOST04-BP05 Use automated machine learning

- MLCOST04-BP06 Use managed training capabilities
- MLCOST04-BP07 Use distributed training
- MLCOST04-BP08 Stop resources when not in use
- MLCOST04-BP09 Start training with small datasets
- MLCOST04-BP10 Use warm start and checkpointing hyperparameter tuning
- MLCOST04-BP11 Use hyperparameter optimization technologies
- MLCOST04-BP12 Set up a budget and use resource tagging to track costs
- MLCOST04-BP13 Enable data and compute proximity
- MLCOST04-BP14 Select optimal algorithms

MLCOST04-BP01 Select optimal computing instance size

Right size your machine learning training instances according to the algorithm and workload requirements to maximize efficiency and reduce costs. By selecting the most appropriate computing resources, you can improve performance while minimizing unnecessary expenses.

Desired outcome: You can identify and select the optimal computing instance types for your machine learning workloads based on actual resource utilization metrics. You can systematically evaluate different instance options, understand their cost implications, and optimize your machine learning infrastructure spending while maintaining or improving performance.

Common anti-patterns:

- Using oversized instances for training jobs regardless of model complexity.
- Ignoring resource utilization metrics during training.
- Failing to experiment with different instance types to find the optimal cost-performance balance.
- Not considering the communication overhead in distributed training scenarios.

Benefits of establishing this best practice:

- Reduced infrastructure costs for machine learning workloads.
- Improved resource utilization and efficiency.
- Better understanding of ML workload performance characteristics.
- Optimization of price-performance ratio for different model types.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Machine learning training workloads vary significantly in their resource requirements based on model complexity, dataset size, and algorithm characteristics. Simple models might not train faster on larger instances because they cannot effectively utilize additional compute resources, and might even train slower due to high GPU communication overhead. By evaluating your workload's resource needs, you can identify the most cost-effective instance configuration.

The key to optimizing instance selection is understanding the actual resource utilization patterns of your machine learning workloads. Start with smaller instances and scale up only when necessary based on performance data. Amazon SageMaker AI provides tools like Debugger to monitor resource utilization and Experiments to compare training performance across different instance configurations. This data-driven approach assists you to avoid paying for unused resources while maintaining optimal training performance.

Implementation steps

1. **Understand your algorithm's resource requirements**. Begin by analyzing whether your machine learning algorithm is compute-bound, memory-bound, or I/O-bound. Different algorithms have different scaling characteristics and resource needs. For deep learning workloads, consider whether GPU acceleration would provide significant benefits or if CPU instances would be more cost-effective for your specific model.
2. **Use Amazon SageMaker AI Experiments**. Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Machine learning workloads can use either a CPU or a GPU instance. Select an instance type from the available EC2 instance types depending on the needs of your ML algorithm. Experiment with both CPU and GPU instances to learn which one gives you the best cost configuration. Amazon SageMaker AI lets you use a single instance or a distributed cluster of GPU instances. Use Amazon SageMaker AI Experiments to evaluate alternative options, and identify the size resulting in optimal outcome. With the pricing broken down by time and resources, you can optimize the cost of Amazon SageMaker AI and only pay for what is needed.
3. **Use Amazon SageMaker AI Debugger**. Amazon SageMaker AI Debugger automatically monitors the utilization of system resources, such as GPUs, CPUs, network, and memory, and profiles your training jobs to collect detailed ML framework metrics. You can inspect resource metrics visually through SageMaker AI Studio and take corrective actions if the resource is under-utilized to optimize cost.

4. **Start small and scale gradually**. Begin with smaller instance sizes for new models and monitor performance. Only increase instance size when you have data showing that your workload can benefit from additional resources. This approach assists you to avoid overprovisioning and unnecessary costs.
5. **Consider the communication overhead**. For distributed training across multiple GPUs or instances, evaluate the communication overhead between nodes. In some cases, adding more compute resources might actually slow down training due to increased coordination requirements.
6. **Monitor and analyze training metrics**. Track key metrics like CPU/GPU utilization, memory usage, I/O patterns, and training throughput across different instance types to identify bottlenecks and optimization opportunities.
7. **Use Spot Instances for cost savings**. For non-critical training jobs, consider using Amazon EC2 Spot Instances through SageMaker AI to reduce costs by up to 90%. Configure your training jobs to checkpoint regularly to minimize the impact of potential interruptions.
8. **Use SageMaker AI Inference Recommender for optimal instance selection**. Use SageMaker AI Inference Recommender with enhanced algorithms and support for multi-model endpoints to get sophisticated cost optimization recommendations for your specific workloads.
9. **For generative AI workloads, use foundation model optimization techniques**. For generative AI workloads, consider techniques like quantization, distillation, and efficient fine-tuning methods to reduce the computational resources needed while maintaining model quality. Amazon SageMaker AI JumpStart provides optimized foundation models that can significantly reduce training time and resource requirements.

Resources

Related documents:

- Amazon SageMaker AI Debugger
- Accelerate generative AI development with Amazon SageMaker AI and MLflow
- Amazon EC2 instance types
- Managed Spot Training in Amazon SageMaker AI
- Amazon SageMaker AI Training Compiler

MLCOST04-BP02 Use managed build environments

Using managed build environments for machine learning development instead of local setups provides significant cost, time, and resource advantages. Managed notebooks come pre-configured with security, networking, storage, and compute capabilities that would otherwise require extensive development and maintenance effort. These environments also offer flexible machine selection, including access to powerful GPUs and high-memory instances that may be impractical in local setups.

Desired outcome: You can quickly start machine learning development work without spending time setting up infrastructure, managing dependencies, or configuring development environments. You gain access to scalable compute resources, including specialized hardware like GPUs, and benefit from built-in security and collaboration features, allowing you to focus on building ML models rather than managing infrastructure.

Common anti-patterns:

- Spending excessive time configuring local development environments for each project.
- Encountering hardware limitations when training complex models locally.
- Struggling with inconsistent development environments across team members.
- Managing security and networking configurations manually.
- Inability to scale resources up or down based on workload requirements.

Benefits of establishing this best practice:

- Reduced time to start development with pre-configured environments.
- Access to powerful compute resources on demand.
- Consistent development environments for team members.
- Built-in security, networking, and storage capabilities.
- Simplified collaboration and sharing of notebooks and models.
- Cost optimization through pay-for-what-you-use model.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

When implementing machine learning projects, your development environment plays a critical role in productivity and efficiency. Local development environments often lead to inconsistencies

between team members, dependency conflicts, and hardware limitations. Managed build environments address these challenges by providing standardized, scalable, and secure solutions for ML development.

Amazon SageMaker AI offers several managed environment options tailored to different user needs and expertise levels. These include SageMaker AI Notebook Instances for individual developers, SageMaker AI Studio for comprehensive ML development, and SageMaker AI Canvas for no-code ML solutions. These environments come pre-configured with the necessary tools and libraries, saving setup time and fostering consistency.

These managed environments integrate seamlessly with other AWS services, making it simple to access data stored in Amazon S3, use specialized hardware like GPUs, and deploy models to production endpoints. They also provide built-in security features, version control, and collaboration capabilities that would be difficult to implement in a local setup.

Implementation steps

1. **Evaluate your ML development needs**. Begin by assessing your team's requirements, including technical expertise, project complexity, and compute resource needs. Identify which SageMaker AI offering best matches these requirements.
2. **Use Amazon SageMaker AI Notebook Instances**. Set up SageMaker AI Notebook Instances which provide a fully managed Jupyter notebook environment. These instances come pre-loaded with popular ML frameworks and libraries, allowing you to start working immediately.
3. **Implement Amazon SageMaker AI Studio**. Deploy SageMaker AI Studio as your comprehensive ML development environment. SageMaker AI Studio provides a web-based visual interface where your team can perform ML development steps from data preparation to model deployment. Access Studio by creating a SageMaker AI domain through the SageMaker AI console, which enables team management and resource sharing capabilities.
4. **Deploy SageMaker AI Canvas for business users**. Implement SageMaker AI Canvas for business analysts and non-technical team members who need to create ML models without coding. Canvas provides an intuitive visual interface for importing data, creating models, and generating predictions.
5. **Set up proper IAM roles and permissions**. Configure appropriate IAM roles for your SageMaker AI environments to provide secure access to AWS resources. Create specific roles that follow the principle of least privilege, granting only the permissions necessary for your ML workflows.

6. **Configure data access and storage**. Set up connections between your SageMaker AI environments and data sources such as Amazon S3, Amazon Redshift, or Amazon RDS. Configure appropriate permissions to access these data sources securely.
7. **Implement version control and collaboration**. Integrate your managed environments with version control systems like Git to track changes to notebooks and code. Use SageMaker AI Studio's built-in collaboration features to share work among team members.
8. **Optimize for cost efficiency**. Configure auto-shutdown policies for notebook instances when they're idle to reduce costs. Monitor resource usage and adjust instance types as needed to balance performance and cost.
9. **Use SageMaker AI HyperPod for large-scale training**. For distributed training of large models, use SageMaker AI HyperPod which provides purpose-built infrastructure with automatic checkpoint storage and recovery, optimizing resource utilization for long-running training jobs.
10. **Enable SageMaker AI JupyterLab 3 features**. Take advantage of the productivity improvements in JupyterLab 3, which is available in both SageMaker AI Studio and Notebook Instances, providing better performance and enhanced features for developers.

Resources

Related documents:

- Amazon SageMaker AI Studio
- Amazon SageMaker AI Canvas
- Amazon SageMaker AI Pricing
- Amazon SageMaker AI notebook instances
- SageMaker AI JumpStart pretrained models
- Pipelines

MLCOST04-BP03 Select local training for small scale experiments

When developing machine learning models, choosing the right training environment is crucial for both cost efficiency and rapid experimentation. By evaluating whether to train your ML model locally or in the cloud, you can optimize your development workflow and appropriately match resources to the scale of your experiment.

Desired outcome: You can rapidly iterate on machine learning experiments with small datasets by training models locally, while having a clear path to scale up to cloud-based training when working

with larger datasets. This approach enables faster development cycles during the experimentation phase and cost-effective scaling when required for production workloads.

Common anti-patterns:

- Deploying cloud-based training clusters regardless of dataset size.
- Using oversized compute instances for small-scale experimentation.
- Not considering the time and cost implications of repeatedly launching training clusters during the experimentation phase.
- Failing to right-size compute resources based on specific workload requirements.

Benefits of establishing this best practice:

- Reduced development costs during experimentation phases.
- Faster iteration cycles when testing various algorithms and configurations.
- Simplified workflow for early-stage development.
- Clear scaling path from local experimentation to production deployment.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

When developing machine learning models, you often need to experiment with multiple algorithms, configurations, and hyperparameters before finding an optimal solution. The choice between local training and cloud-based training significantly impacts both development speed and cost efficiency.

Local training is most advantageous during early experimentation phases when working with small datasets. This approach reduces the overhead of provisioning cloud resources and waiting for training clusters to spin up for each experiment iteration. Your development cycle becomes more agile as you can quickly test hypotheses and make adjustments without incurring additional cloud costs.

As your models and datasets grow in size and complexity, transitioning to cloud-based training becomes necessary. Cloud environments offer scalable computing resources that can handle large datasets and complex models that would be impractical to process on local machines. By right-sizing your compute instances based on your specific workload requirements, you can maintain cost efficiency while gaining the performance benefits of cloud infrastructure.

Implementation steps

1. **Evaluate your training requirements**. Before deciding on local or cloud-based training, assess your dataset size, model complexity, and computational requirements. Small datasets (typically under a few gigabytes) and simpler models are generally good candidates for local training, especially during initial experimentation.
2. **Set up Amazon SageMaker AI local mode**. When experimenting with small datasets, use Amazon SageMaker AI's local mode to train models directly on your notebook instance. This approach allows you to test and iterate on your code without provisioning separate training clusters. To implement local mode:

```
from sagemaker.estimator import Estimator

estimator = Estimator(
    image_uri="your-container-image",
    role="your-sagemaker-role",
    instance_count=1,
    instance_type="local"
)

estimator.fit({"train": "s3://your-bucket/train-data"})
```

3. **Use local development environment with SageMaker AI SDK**. For development outside of SageMaker AI notebooks, install the SageMaker AI Python SDK on your local machine. This allows you to develop and test locally while still having the ability to deploy models to AWS:

```
pip install sagemaker
```

4. **Profile your workloads for cloud deployment**. As your models mature and datasets grow, prepare for cloud deployment by profiling your workloads. Identify memory usage, CPU and GPU requirements, and processing time to determine appropriate instance types for cloud-based training.
5. **Right-size cloud-based training clusters**. When moving to cloud training, select appropriate instance types based on your workload profiling. Consider factors such as:
   - Model architecture (CPU and GPU requirements)
   - Memory needs
   - Dataset size and I/O patterns

- Training time constraints
- Cost constraints

6. **Implement distributed training for large-scale workloads**. For large datasets or complex models, configure distributed training across multiple instances to reduce training time.
7. **Monitor and optimize cloud resource usage**. Regularly review your training job metrics to identify opportunities for optimization. Use SageMaker AI Experiments to track and compare resource utilization across different training configurations.
8. **Use enhanced local development capabilities**. Use improved SageMaker AI local mode with better debugging and monitoring capabilities, allowing for more efficient local experimentation before scaling to cloud resources.
9. **For generative AI workloads, use foundation models efficiently**. When working with generative AI and foundation models, consider using Amazon SageMaker AI JumpStart for local experimentation with smaller, distilled versions of foundation models before fine-tuning larger models in the cloud. This approach allows for rapid prototyping while managing costs effectively.

Resources

Related documents:

- Model training
- AWS Pricing Calculator
- What is Amazon SageMaker AI?
- SageMaker AI Python SDK Documentation
- SageMaker AI JumpStart pretrained models
- Generative AI Cost Optimization Strategies

Related videos:

- Train with Amazon SageMaker AI on your local machine

Related examples:

- SageMaker AI Local Mode Examples

MLCOST04-BP04 Select an optimal ML framework

Selecting the most cost-effective machine learning (ML) framework for your requirements can significantly impact your operational efficiency and return on investment. By systematically comparing frameworks like TensorFlow, PyTorch, and Scikit-learn, you can determine which delivers the best performance for your specific use cases at the optimal cost.

Desired outcome: You establish a systematic approach for evaluating ML frameworks and instance types, and you can select the optimal combination based on performance, cost, and use case requirements. You can track, compare, and analyze experiments across different frameworks, leading to informed decisions that maximize performance while minimizing costs.

Common anti-patterns:

- Selecting ML frameworks based on popularity rather than suitability for your specific use case.
- Using a single framework for ML projects regardless of workload characteristics.
- Not tracking experiment metrics systematically across different frameworks.
- Failing to benchmark performance and cost metrics before moving to production.

Benefits of establishing this best practice:

- Reduced operational costs through optimized infrastructure selection.
- Improved model performance by selecting the most suitable framework.
- Enhanced productivity by streamlining experiment tracking and comparison.
- Faster iteration and deployment of ML models.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Selecting the optimal ML framework involves evaluating different options against your specific requirements and constraints. Consider factors such as model complexity, data volume, performance requirements, and team expertise when choosing between frameworks. Tracking experiments systematically assists you to compare approaches objectively and make data-driven decisions.

When implementing this best practice, use AWS' comprehensive ML infrastructure, which supports major frameworks and provides tools for experiment tracking and resource optimization. Regular

performance benchmarking and cost analysis should become standard procedures in your ML development process.

Implementation steps

1. **Implement systematic experiment tracking with SageMaker AI Experiments**. Amazon SageMaker AI Experiments enables you to organize, track, compare, and evaluate your machine learning experiments. Create experiments to group related trials, assign parameters, metrics, and artifacts to each trial, and track the lineage of model artifacts to experiments for governance and reproducibility.
2. **Compare multiple ML frameworks**. Evaluate frameworks like TensorFlow, PyTorch, Apache MXNet, and Scikit-learn for your specific use cases. Use AWS Deep Learning AMIss and AWS Deep Learning Containers to experiment with different frameworks using consistent infrastructure. These AMIs come with popular frameworks preinstalled, making it simple to switch between them for comparison.
3. **Benchmark framework performance**. Design standardized benchmarking tests for your specific workloads across different frameworks. Track metrics such as training time, inference latency, memory usage, and accuracy to determine which framework performs best for your use case.
4. **Implement right-sizing strategies for ML instances**. Use SageMaker AI's managed instances to automatically select the most appropriate and cost-effective instance type for your workloads. Experiment with different instance types to find the optimal balance between performance and cost.
5. **Use SageMaker AI's bring-your-own-container capability**. If you need to use specialized ML frameworks or versions not available in standard containers, use SageMaker AI's flexibility to bring your own containers so that you can use a framework while maintaining the benefits of SageMaker AI's managed infrastructure.
6. **Implement automatic resource scaling**. Configure automatic scaling for inference endpoints based on traffic patterns to optimize costs during varying load conditions. Use SageMaker AI Inference Recommender to identify the best configuration for deployment.
7. **Use enhanced experiment tracking with MLflow**. Use managed MLflow on SageMaker AI to create, manage, analyze, and compare your machine learning experiments across different frameworks with better organization and tracking capabilities.
8. **Monitor and optimize costs continuously**. Implement cost monitoring using AWS Cost Explorer and SageMaker AI's built-in monitoring capabilities. Set up alerts for unusual spending patterns and regularly review resource utilization to identify optimization opportunities.

Resources

Related documents:

- AWS Deep Learning AMIss
- Machine Learning Frameworks and Languages
- Accelerate generative AI development with Amazon SageMaker AI and MLflow
- Amazon SageMaker AI Studio Classic
- Amazon SageMaker AI Inference Recommender

MLCOST04-BP05 Use automated machine learning

Automate your model development process by using systems that experiment with and select the best algorithms from high-performing options. These automated systems test various solutions and parameter settings to achieve optimal models, significantly speeding up development while reducing the need for manual experimentation and comparisons.

Desired outcome: You gain the ability to develop high-quality machine learning models in a fraction of the time traditionally required. By using automated machine learning tools like Amazon SageMaker AI Autopilot, you can focus on business problems rather than algorithm selection and parameter tuning. Your team can produce optimized models with better performance, reduce development costs, and accelerate time-to-market for ML-powered solutions.

Common anti-patterns:

- Manually testing multiple algorithms and configurations one by one.
- Spending excessive time on hyperparameter tuning without systematic approach.
- Using the same algorithm for each problem without considering alternatives.
- Neglecting cross-validation during model selection.

Benefits of establishing this best practice:

- Dramatically reduced time to develop production-ready models.
- Access to a broader range of algorithms and optimization techniques.
- Improved model performance through systematic evaluation.
- Lower costs through optimized resource utilization.
- Ability for domain experts to build models without deep ML expertise.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Automated machine learning (AutoML) systems democratize the process of building machine learning models. By automating key steps in model development—from data preparation to algorithm selection and hyperparameter tuning—these systems enable even those without extensive machine learning expertise to develop high-quality models.

When using AutoML solutions like Amazon SageMaker AI Autopilot, you provide your dataset and define your objective, and the system handles the complex work of exploring potential algorithms, optimizing parameters, and evaluating model performance. The system applies cross-validation procedures automatically to check that models can generalize well to new data. By ranking optimized models by their performance, AutoML can identify the best solution for your specific problem.

Beyond simply producing models, modern AutoML systems provide visibility into the development process, allowing you to understand what choices were made and why. This transparency builds trust in the models and provides learning opportunities for your team to understand what approaches work best for different problem types.

Implementation steps

1. **Evaluate your use case compatibility**. Determine if your ML problem is suitable for automated machine learning solutions. AutoML works particularly well for standard machine learning tasks like classification, regression, and some time series forecasting scenarios.
2. **Prepare your data for AutoML**. Clean your dataset, handle missing values, and convert categorical features appropriately. While AutoML handles feature engineering, providing high-quality data improves results. Use Amazon SageMaker AI Data Wrangler to simplify this preparation process.
3. **Set up Amazon SageMaker AI Autopilot with Canvas**. Open Amazon SageMaker AI Canvas, import your dataset into Amazon S3, and configure to access this data. Define your target variable and specify your problem type (classification or regression) if known.
4. **Launch the automated ML job**. Start Canvas training and let it analyze your data, select algorithms, and optimize models. Specify resources like maximum runtime and instance types to control costs. Canvas will automatically handle data preprocessing, feature engineering, algorithm selection, and hyperparameter optimization.

5. **Review candidate models**. Examine the generated models along with their performance metrics. Autopilot provides detailed reports on the data exploration, feature engineering decisions, and model optimization steps it performed.
6. **Deploy the best model**. Select the best-performing model from the Canvas recommendations and deploy it using Amazon SageMaker AI's deployment capabilities. You can deploy as a real-time endpoint or for batch inference depending on your needs.
7. **Monitor and evaluate performance**. Set up model monitoring to track your model's performance in production and detect concept drift. Use Amazon SageMaker AI Model Monitor to automate this process.
8. **Customize and refine models**. If needed, extract and customize the models generated by Autopilot. The solution provides full visibility into the notebooks and artifacts it creates, allowing you to further refine specific aspects of the model.
9. **Enhance model development with foundation models**. Use Amazon Bedrock to incorporate foundation model capabilities into your AutoML workflow for tasks like text processing, content generation, and multimodal applications. Foundation models can complement traditional ML approaches handled by Autopilot.
10. **Use enhanced Canvas capabilities with Q integration**. Use SageMaker AI Canvas with improved natural language support and Q integration for conversational data analysis, enabling business users to build models through natural language interactions.
11. **Implement intelligent preprocessing with generative AI**. Use generative AI tools to enhance data preprocessing, augment training datasets, generate synthetic data for edge cases, and improve feature engineering through intelligent text and image processing.

Resources

Related documents:

- Getting started with using Amazon SageMaker AI Canvas
- SageMaker AI Canvas
- Amazon SageMaker AI Data Wrangler

Related examples:

- SageMaker AI Autopilot
- Amazon SageMaker AI Autopilot Sample Notebooks

MLCOST04-BP06 Use managed training capabilities

Machine learning model training can be an iterative, compute-intensive, and time-consuming process. Instead of using the notebook itself, which might be running on a small instance, offloading the training to a managed cluster of compute resources including both CPUs and GPUs enables more efficient and cost-effective model training.

Desired outcome: By using managed training capabilities, you optimize your machine learning training workflows and infrastructure management. You gain access to scalable computing resources that automatically adjust based on your workload needs, from single GPUs to thousands, without managing the underlying infrastructure. You can significantly reduce training costs through specialized hardware options, compiler optimizations, and spot instance utilization while maintaining visibility into metrics and logs for proper monitoring and governance.

Common anti-patterns:

- Running complex model training jobs on notebook instances, leading to resource constraints and inefficiency.
- Managing your own GPU clusters for training, requiring significant operational overhead.
- Using exclusively on-demand instances for training jobs, resulting in higher costs.
- Not using specialized training optimizations like distributed training or compiler acceleration.

Benefits of establishing this best practice:

- Lower training costs by up to 90% using managed spot instances.
- Accelerate training time by up to 50% with training compiler optimizations.
- Scale resources automatically based on training job requirements.
- Track and monitor training experiments and resource utilization effectively.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Machine learning training is computationally intensive and can become prohibitively expensive when not optimized properly. Using managed training capabilities allows you to focus on model development while the infrastructure scales and optimizes automatically to your needs. Managed training services provide a range of optimization options from distributed training across multiple

GPUs to cost-saving options through spot instances. Additionally, these services integrate with monitoring tools to track resource utilization, model metrics, and training progress to continually refine your training approach.

For example, when training large language models, you can use SageMaker AI's distributed training libraries to split the model across multiple GPUs and instances, reducing training time from weeks to days while maintaining control over your training costs through automatic scaling and spot instance usage.

Implementation steps

1. **Use Amazon SageMaker AI managed training capabilities**. Amazon SageMaker AI reduces the time and cost to train and tune ML models without the need to manage infrastructure. With SageMaker AI, you can train and tune ML models using built-in tools to manage and track training experiments, automatically choose optimal hyperparameters, debug training jobs, and monitor the utilization of system resources such as GPUs, CPUs, and network bandwidth. SageMaker AI can automatically scale infrastructure up or down based on your training job requirements, from one GPU to thousands, or from terabytes to petabytes of storage. SageMaker AI also offers the highest-performing ML compute infrastructure currently available-including Amazon EC2 P4d instances, which can reduce ML training costs by up to 60% compared with previous generations. And, since you pay only for what you use, you can manage your training costs more effectively.
2. **Use Spot Instances for cost optimization**. Amazon SageMaker AI makes it simple to train machine learning models using managed Amazon EC2 Spot Instances. Managed Spot training can optimize the cost of training models up to 90% over On-demand Instances. SageMaker AI manages the Spot interruptions on your behalf. You can specify which training jobs use Spot Instances and a stopping condition that specifies how long SageMaker AI waits for a job to run using Spot Instances. Metrics and logs generated during training runs are available in Amazon CloudWatch.
3. **Configure optimal data sources**. Select the appropriate data source for your training job to optimize performance and cost. Consider using Amazon S3 for persistent storage, Amazon FSx for Lustre for high-performance file systems, or Amazon EFS based on your specific training requirements and dataset characteristics.
4. **Implement experiment tracking and management**. Use Amazon SageMaker AI Experiments to track training jobs, compare results, and manage different versions of your models. This provides visibility into model performance, resource utilization, and training metrics to optimize future iterations.

5. **Use SageMaker AI HyperPod for large-scale training**. Use SageMaker AI HyperPod to scale and accelerate generative AI model development across thousands of AI accelerators with purpose-built infrastructure, automatic checkpoint storage and recovery, and support for both Slurm and Amazon EKS for cluster orchestration.
6. **For generative AI, optimize large language model training**. Use SageMaker AI Model Parallelism to efficiently distribute model parameters across multiple devices and instances. Consider using Amazon Bedrock for foundation model access and fine-tuning capabilities to further reduce the computational cost of training generative AI models from scratch.

Resources

Related documents:

- SageMaker AI HyperPod
- Managed Spot Training in Amazon SageMaker AI
- Train a Model with Amazon SageMaker AI
- Model training
- Distributed training in Amazon SageMaker AI
- Accelerate generative AI development with Amazon SageMaker AI and MLflow

Related examples:

- SageMaker AI Examples GitHub Repository
- SageMaker AI Training Workshop

MLCOST04-BP07 Use distributed training

Accelerate your machine learning model training process by utilizing distributed computing resources, which can significantly reduce training time and optimize costs. Amazon SageMaker AI distributed training capabilities enable efficient processing of large models and datasets across multiple compute instances.

Desired outcome: You achieve faster training times for your machine learning models by distributing the workload across multiple instances. You optimize resource utilization and reduce overall training costs by using SageMaker AI's managed distributed training capabilities, which automatically handle infrastructure provisioning and termination when training completes. This

approach allows you to train complex models that may be too large for a single machine or train standard models much faster through parallel processing.

Common anti-patterns:

- Training large models on a single instance even when they could benefit from distribution.
- Manually managing distributed training infrastructure rather than using managed services.
- Keeping training instances running after training is complete.
- Implementing custom distributed training code when built-in libraries would suffice.

Benefits of establishing this best practice:

- Significantly reduced training time for large models and datasets.
- Cost optimization through efficient resource utilization.
- Ability to train models that are too large to fit on a single GPU.
- Automatic infrastructure management with no need to maintain distributed training clusters.
- Enhanced team productivity by reducing waiting time for model results.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Distributed training allows you to split your machine learning workloads across multiple compute instances to accelerate the training process. This approach is particularly valuable when working with large models or datasets that would otherwise take too long to train on a single instance. Amazon SageMaker AI provides built-in support for distributed training through its specialized libraries that handle the complexity of distributing workloads efficiently.

When implementing distributed training, you need to consider the most appropriate approach based on your model architecture and data size. Data parallelism works by dividing your dataset across multiple GPUs, with each GPU having a complete copy of the model. This approach is ideal for scenarios where your model fits on a single GPU but training on the full dataset is time-consuming. Alternatively, model parallelism is designed for situations where your model is too large to fit on a single GPU. In this case, the model itself is partitioned across multiple GPUs.

SageMaker AI's distributed training libraries automatically handle the communication between nodes and optimize the distribution strategy, making it straightforward to scale your training workloads without managing the underlying infrastructure.

Implementation steps

1. **Evaluate your workload for distributed training suitability**. Assess if your training job would benefit from distribution by considering factors like model size, dataset size, and current training times. Ideal candidates are models that take hours or days to train on a single instance or models too large to fit in a single GPU's memory.
2. **Choose the appropriate distributed training approach**. Select between data parallelism and model parallelism based on your specific needs. Use data parallelism when your model fits on a single GPU but you want to process data faster. Use model parallelism when your model is too large to fit on a single GPU.
3. **Utilize Amazon SageMaker AI distributed training libraries**. Implement distributed training using SageMaker AI's distributed training libraries, which automatically handle the complexities of distributing workloads across multiple instances. These libraries provide optimized implementations for both data parallelism and model parallelism strategies.
4. **Configure your training cluster**. Define the number and type of instances for your training cluster in your SageMaker AI training job configuration. Consider using GPU-optimized instance types like P3, P4d, or G4dn based on your model requirements and budget constraints.
5. **Adapt your training script for distributed processing**. Modify your training code to work with SageMaker AI's distributed training libraries. For data parallelism, you'll need to use the SageMaker AI data parallelism library to distribute data across workers. For model parallelism, you'll integrate the SageMaker AI model parallelism library to partition your model across devices.
6. **Monitor and optimize training performance**. Use Amazon SageMaker AI Debugger to monitor your distributed training jobs, identify bottlenecks, and optimize resource utilization. Analyze metrics like GPU utilization, communication overhead, and training throughput to fine-tune your distributed training configuration.
7. **Consider Amazon SageMaker AI HyperPod for persistent training clusters of foundation models**. For workloads requiring long-running or repeated distributed training jobs, use Amazon SageMaker AI HyperPod to create persistent, managed clusters that can handle multiple training jobs efficiently while maintaining cost optimization through automatic scaling and resource management.
8. **Use SageMaker AI HyperPod for persistent training clusters**. Use SageMaker AI HyperPod for workloads requiring long-running or repeated distributed training jobs, providing persistent, managed clusters with automatic scaling, checkpoint storage and recovery, and support for various instance types including P5e, G6, and Trn2.

9. **Use AI-powered code generation for distributed training implementation**. Use AI-powered development tools like Amazon Q Developer and Kiro to generate complex distributed training code, automate infrastructure setup scripts, and accelerate the implementation of distributed training workflows.
10. **Consider Amazon Bedrock for fine-tuning foundation models**. For generative AI applications, consider using Amazon Bedrock for fine-tuning foundation models, model distillation, or continued pretraining, which provides optimized distributed training capabilities specifically designed for large language models.

Resources

Related documents:

- Run distributed training with the SageMaker AI distributed data parallelism library
- SageMaker AI model parallelism library v2
- Amazon SageMaker AI HyperPod
- Distributed Training
- Amazon SageMaker AI XGBoost now offers fully distributed GPU training

Related examples:

- Distributed Training
- Distributed training using Amazon SageMaker AI Distributed Data Parallel library and debugging using Amazon SageMaker AI Debugger
- SageMaker AI developer guide on distributed training
- Distributed Training Examples

MLCOST04-BP08 Stop resources when not in use

Stop resources that are not in use to reduce cost. For example, hosted Jupyter environments used to explore small samples of data can be stopped when not actively in use. Where practical, commit the work, stop them, and restart when needed. The same approach can be used to stop the computing and the data storage services.

Desired outcome: You significantly reduce your ML infrastructure costs by only paying for resources when they are actively being used. You have automated systems in place to monitor

and shut down idle resources, along with proper alerts to track spending patterns and avoid unexpected charges. You maintain the ability to quickly restart resources when needed while minimizing wasteful spending on idle compute and storage.

Common anti-patterns:

- Leaving development environments running regardless of actual usage.
- Neglecting to set up automatic shutdown mechanisms for idle resources.
- Ignoring cost monitoring tools and billing alerts.
- Using persistent storage for temporary data that could be deleted.

Benefits of establishing this best practice:

- Significant cost savings (up to 75% by running resources only during business hours compared to running continually).
- Better alignment of spending with actual usage patterns.
- Reduced environmental impact through more efficient resource consumption.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Optimizing costs is a crucial aspect of running machine learning workloads in the cloud. ML workloads often require significant computational resources, but those resources aren't needed continuously. By implementing automatic shutdown mechanisms for idle resources, you can achieve substantial cost savings while maintaining the ability to rapidly resume work when needed.

For ML development environments like SageMaker AI notebooks, the cost-optimization opportunity is particularly significant since these environments are typically used intermittently during the exploration and development phases. By committing code to repositories regularly and shutting down environments when not in use, you improve both cost efficiency and version control of your work.

Additionally, proper monitoring of spending patterns assists you in identifying optimization opportunities and avoiding unexpected costs. With AWS tools, you can set up alerts, track resource utilization, and implement automated responses to idle resources.

Implementation steps

1. **Set up Amazon CloudWatch billing alarms**. Use Amazon CloudWatch to monitor your estimated AWS charges. When you enable monitoring of estimated charges, these calculations are sent several times daily to CloudWatch as metric data. Configure alerts to be notified when your resource charges exceed predefined thresholds to stay within budget and quickly identify unexpected spending patterns.
2. **Configure Amazon SageMaker AI notebook lifecycle configurations**. Create lifecycle configurations that include shell scripts to run when you create or start notebook instances. These scripts can check for notebook instance activity and automatically shut down idle instances. This way, you're not paying for compute resources when they aren't actively processing workloads.
3. **Implement Amazon SageMaker AI Studio idle shutdown**. For Amazon SageMaker AI Studio environments, install the auto-shutdown JupyterLab extension either manually or automatically. This extension detects idle Studio resources and can shut down individual components, including notebooks, terminals, kernels, applications, and instances when they're not being used.
4. **Use AWS Cost Explorer to identify optimization opportunities**. Regularly analyze your ML infrastructure spending patterns using AWS Cost Explorer to identify resources that might be consistently underutilized. Look for patterns that indicate resources could benefit from scheduled shutdowns during off-hours.
5. **Implement instance scheduling**. Use the AWS Instance Scheduler to create automated schedules for starting and stopping resources based on your team's working hours. This is particularly useful for development environments that are only needed during business hours.
6. **Train teams on cost-aware practices**. Educate your ML teams on the importance of shutting down resources when not in use and committing their work regularly. Create a cost-aware culture where resource efficiency is valued alongside development productivity.
7. **Implement enhanced auto-shutdown capabilities**. Use improved SageMaker AI Studio auto-shutdown features with better idle detection and more granular control over resource shutdown policies to minimize costs from unused resources.
8. **Use Spot Instances for interruptible workloads**. For ML training jobs that can handle interruptions, use Amazon EC2 Spot Instances to achieve significant cost savings compared to on-demand pricing. Make sure your workloads are designed to checkpoint progress and can resume from interruptions.

Resources

Related documents:

- Idle shutdown
- Customization of a SageMaker AI notebook instance using an LCC script
- Instance Scheduler on AWS
- Create a billing alarm to monitor your estimated AWS charges
- AWS Cost Explorer
- Amazon SageMaker AI Pricing
- Cloud Financial Management with AWS

Related videos:

- Saving cost on your machine learning training and inference on AWS
- Deploy an ML model for best performance, cost, and prediction quality

Related examples:

- AWS CloudFormation templates for automated instance scheduling

MLCOST04-BP09 Start training with small datasets

Start experimentation with smaller datasets on a small compute instance or local system. This approach allows you to iterate quickly at low cost. After the experimentation period, scale up to train with the full dataset available on a separate compute cluster. Choose the appropriate storage layer for training data based on the performance requirements.

Desired outcome: You can develop your machine learning models cost-effectively by starting with small datasets for rapid iteration and experimentation. When you're confident in your approach, you scale up to the full dataset on appropriate compute resources. This progressive scaling methodology optimizes both development time and infrastructure costs while maintaining the flexibility to refine your models before committing to full-scale training.

Common anti-patterns:

- Immediately training with the full dataset on large instances, leading to excessive costs during experimentation.

- Using the same compute resources for both experimentation and full-scale training.
- Not planning for the transition from small-scale to large-scale training.

Benefits of establishing this best practice:

- Reduced costs during the experimentation phase.
- Faster iteration cycles for model development.
- More efficient use of compute resources.
- Ability to identify and fix issues early in the development process.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Machine learning development often requires multiple iterations to achieve optimal results. Using smaller, representative samples of your dataset during initial experimentation can significantly reduce costs and increase productivity. This approach lets you rapidly test various model architectures, hyperparameters, and preprocessing techniques without the expense and time required to process the full dataset.

When implementing this approach, check that your smaller dataset properly represents the characteristics of your full dataset to avoid developing models that don't generalize well. Once you've established effective approaches using the smaller dataset, you can scale up your training to use the complete dataset on appropriately sized compute resources.

The cloud makes this approach particularly powerful, as you can scale your compute resources to match your current phase of development. For example, you might use a notebook instance with modest resources during experimentation, then transition to distributed training on a cluster of more powerful instances when you're ready for full-scale training.

Implementation steps

1. **Create a representative subset of your data**. Extract a small but representative sample of your full dataset that maintains the same distribution of features and classes as your original data. Aim for 10-20% of your data or a size that can be processed on your local machine or small instance.
2. **Set up SageMaker AI notebook instances for experimentation**. Amazon SageMaker AI notebook instances provide a hosted Jupyter environment ideal for exploring and

experimenting with your sample dataset. Choose a smaller instance type to keep costs low during experimentation.

3. **Configure notebook lifecycle management**. Use lifecycle configuration scripts to automate the setup of your development environment, including installing necessary libraries and dependencies when your notebook instance starts.
4. **Develop and iterate on your model**. Use the notebook environment to build, train and evaluate your models on the sample data. Take advantage of this faster iteration cycle to explore different approaches, hyperparameters, and preprocessing techniques.
5. **Test scaling considerations**. Before moving to full-scale training, test your code with slightly larger data samples to identify scaling issues that might arise when processing the full dataset.
6. **Prepare for distributed training**. Once your approach is validated with the sample data, refactor your code as needed to support distributed training using SageMaker AI's distributed training capabilities.
7. **Scale up compute resources for full training**. Launch appropriately sized training instances or clusters for your full-scale training job. SageMaker AI training jobs allow you to select the instance type and count that matches your workload requirements.
8. **Monitor training metrics and costs**. Use Amazon CloudWatch to track the performance and resource utilization of your training jobs to check that they're running efficiently.

Resources

Related documents:

- Amazon SageMaker AI notebook instances
- Amazon SageMaker AI Studio Lab
- Customization of a SageMaker AI notebook instance using an LCC script
- Distributed training in Amazon SageMaker AI

Related examples:

- SageMaker AI Notebook Instance Lifecycle Config Samples
- SageMaker AI Local Mode

MLCOST04-BP10 Use warm start and checkpointing hyperparameter tuning

When training machine learning models, you can significantly reduce time and costs by using previous training efforts. This practice shows how to use warm start and checkpointing techniques in hyperparameter tuning to accelerate your model development process and optimize resource utilization.

Desired outcome: You can create more efficient hyperparameter tuning jobs by using knowledge from previous tuning efforts and saved model states. By implementing warm start capabilities, you can initialize new tuning jobs with information from previous runs, avoiding unnecessary repetition. With checkpointing, you can save intermediate model states during training, allowing you to resume jobs from the last checkpoint rather than starting from scratch. These techniques enable you to accelerate your model development process, reduce computational costs, and find optimal hyperparameter configurations more efficiently.

Common anti-patterns:

- Starting every hyperparameter tuning job from scratch without using previous knowledge.
- Not saving model checkpoints during lengthy training jobs, risking complete loss of progress if interrupted.
- Using unnecessarily wide hyperparameter search ranges when previous jobs have already identified promising areas.

Benefits of establishing this best practice:

- Lower computational costs through more efficient resource utilization.
- Accelerated convergence to optimal model configurations.
- Improved resilience to training interruptions through checkpoint recovery.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Hyperparameter tuning is an essential but computationally intensive part of machine learning model development. Without warm start capabilities, each tuning job begins with no prior knowledge, potentially wasting resources by exploring already-evaluated hyperparameter

combinations. Without checkpointing, an interrupted training job must restart from the beginning, losing progress.

You can overcome these inefficiencies by implementing warm start and checkpointing strategies in your ML workflow. Warm start allows you to use knowledge from previous hyperparameter tuning jobs, focusing the search on promising areas of the hyperparameter space. Checkpointing enables you to save model states periodically during training, providing a recovery point if training is interrupted.

Amazon SageMaker AI offers built-in support for both warm start and checkpointing capabilities. For warm start, you can specify one or more parent tuning jobs whose results inform the new job's hyperparameter search. SageMaker AI offers two warm start types: TRANSFER_LEARNING for adapting knowledge to new datasets and IDENTICAL_DATA_AND_ALGORITHM for continuing tuning with the same dataset. For checkpointing, you can configure your training jobs to periodically save model states to Amazon S3, which can be used to resume training if needed.

Implementation steps

1. **Configure warm start for hyperparameter tuning jobs**. Set up a new hyperparameter tuning job that builds upon the knowledge gained from previous tuning jobs. In Amazon SageMaker AI, you can configure this by specifying one or more parent tuning jobs and selecting an appropriate warm start type. This approach is particularly effective when you want to refine hyperparameter search after initial exploration or adapt a model to a similar dataset.
2. **Select appropriate parent jobs for warm start**. Choose parent jobs that are relevant to your current tuning objective. The best parent jobs are those that used similar datasets, algorithms, or optimization objectives. In SageMaker AI, you can specify up to five parent jobs when configuring a warm start tuning job.
3. **Choose the right warm start type**. Select IDENTICAL_DATA_AND_ALGORITHM when continuing tuning with the same dataset and algorithm, or TRANSFER_LEARNING when adapting knowledge to a new but related dataset or problem. The warm start type determines how SageMaker AI will use information from the parent jobs.
4. **Configure checkpointing for training jobs**. Enable checkpointing in your training script by saving model states at regular intervals. In SageMaker AI, specify a checkpoint S3 location where these model states will be stored. This allows you to resume training from the last saved checkpoint if a job is interrupted or if you want to extend training later.
5. **Implement checkpoint saving in your training code**. Add callback functions in your ML framework (such as TensorFlow, PyTorch, or MXNet) to periodically save model states during

training. These frameworks typically provide built-in checkpoint functionality that you can configure with minimal code changes.

6. **Set up checkpoint recovery mechanisms**. Configure your training jobs to check for existing checkpoints at startup and resume from the latest checkpoint if available. In SageMaker AI, you can specify the checkpoint configuration when creating a training job, including the S3 location where checkpoints are stored.
7. **Optimize hyperparameter search ranges based on previous results**. When using warm start, refine your hyperparameter search ranges based on promising values identified in parent jobs. Narrowing search ranges around previously successful values can significantly improve tuning efficiency.
8. **Run parallel hyperparameter tuning jobs strategically**. Use warm start to distribute the hyperparameter tuning workload across multiple jobs that can share knowledge. This approach is particularly effective for exploring large hyperparameter spaces efficiently.
9. **Monitor and evaluate warm start efficiency**. Track the performance and efficiency gains from warm start by comparing with cold-start approaches. This analysis refines your warm start strategy for future jobs.
10. **Use enhanced hyperparameter tuning capabilities**. Use improved SageMaker AI hyperparameter tuning with better algorithms and support for multi-objective optimization to find optimal configurations more efficiently.
11. **Use generative AI for hyperparameter selection**. Use large language models to suggest promising hyperparameter ranges based on model architecture and dataset characteristics. Generative AI can identify sensible starting points for hyperparameter tuning jobs, especially for new model architectures.

Resources

Related documents:

- Run a Warm Start Hyperparameter Tuning Job
- Checkpoints in Amazon SageMaker AI
- Automatic model tuning in Amazon SageMaker AI
- HyperparameterTuner

Related examples:

- Automatic Model Tuning: Warm Starting Tuning Jobs

- Hyperparameter Optimization with Checkpointing Example

MLCOST04-BP11 Use hyperparameter optimization technologies

Optimize your machine learning models through automatic hyperparameter tuning to find the optimal model configuration with minimal manual effort, reducing the time and resources needed to achieve peak model performance.

Desired outcome: You achieve better performing machine learning models by using automatic hyperparameter optimization technologies that run multiple training jobs in parallel. You can efficiently explore a wide range of hyperparameter combinations to find the optimal configuration that maximizes model performance according to your specified metrics, ultimately delivering better business results while reducing the time and resources spent on manual tuning.

Common anti-patterns:

- Manually tuning hyperparameters through trial and error.
- Using a narrow range of hyperparameter values that don't adequately explore the solution space.
- Selecting arbitrary hyperparameter values without considering the specific requirements of your business problem.
- Running one training job at a time instead of using parallel capabilities.

Benefits of establishing this best practice:

- Reduced time to develop high-performing machine learning models.
- Lower computational costs by efficiently exploring the hyperparameter space.
- Consistent and repeatable approach to model optimization.
- Ability to scale hyperparameter tuning efforts across multiple algorithms.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Hyperparameter optimization (HPO) is a critical aspect of developing effective machine learning models. Unlike model parameters that are learned during training, hyperparameters are configuration variables that govern the training process itself and significantly impact model performance. Finding the optimal combination of hyperparameters manually is time-consuming and inefficient.

By implementing automatic hyperparameter tuning, you can systematically explore the hyperparameter space and identify the configuration that maximizes model performance. SageMaker AI's automatic model tuning service employs techniques like Bayesian optimization to intelligently search through the hyperparameter space, focusing computational resources on the most promising regions and accelerating the discovery of the optimal configuration.

When implementing hyperparameter optimization, you should define appropriate search spaces for your hyperparameters based on domain knowledge and previous experiments. You also need to select relevant evaluation metrics that align with your business objectives. For classification problems, this might include accuracy, F1 score, or AUC-ROC, while for regression problems, it could be mean squared error or mean absolute error.

Implementation steps

1. **Identify key hyperparameters for your model**. Begin by determining which hyperparameters have the greatest impact on your model's performance. For neural networks, this might include learning rate, batch size, and network architecture parameters. For tree-based models, this could include tree depth, number of trees, and minimum samples per leaf.
2. **Define appropriate hyperparameter ranges**. Establish meaningful ranges for each hyperparameter based on domain knowledge and best practices for your chosen algorithm. Use logarithmic scales for parameters that span multiple orders of magnitude (like learning rate) for efficient exploration.
3. **Select relevant evaluation metrics**. Choose metrics that align with your business requirements and the problem you're solving. Check that these metrics provide a meaningful assessment of model performance in the context of your specific application.
4. **Configure SageMaker AI automatic model tuning**. Create a hyperparameter tuning job using the SageMaker AI Python SDK or the SageMaker AI console. Specify the algorithm or framework you're using, the hyperparameter ranges, and the evaluation metric to optimize.
5. **Implement early stopping for efficiency**. Enable early stopping features to automatically terminate poorly performing training jobs, saving computational resources. SageMaker AI can monitor the evaluation metric during training and stop jobs that are unlikely to produce competitive models.
6. **Use warm start for incremental tuning**. Use the warm start feature to accelerate new hyperparameter tuning jobs by using information from previous tuning jobs, reducing the time and resources needed to find optimal configurations.

7. **Implement parallel training jobs**. Configure SageMaker AI to run multiple training jobs concurrently to explore different hyperparameter combinations simultaneously, dramatically reducing the time required to find optimal values.
8. **Analyze tuning job results**. Review the performance of different hyperparameter combinations to understand how each parameter affects model performance. Use this information to refine your hyperparameter ranges for future tuning jobs.
9. **Select the best model for deployment**. After the tuning job completes, identify the best-performing model based on your evaluation metric and deploy it using SageMaker AI's deployment capabilities.
10. **Use no-code hyperparameter optimization**. Use SageMaker AI Canvas with enhanced capabilities for business users to perform hyperparameter optimization through natural language interfaces without requiring deep technical expertise.
11. **Document hyperparameter configurations**. Maintain comprehensive documentation of hyperparameter configurations, tuning strategies, and results to facilitate knowledge sharing and reproducibility.

Resources

Related documents:

- Automatic model tuning with SageMaker AI
- Stop Training Jobs Early
- Best Practices for Hyperparameter Tuning
- Create a Hyperparameter Optimization Tuning Job for One or More Algorithms (Console)
- Run a Warm Start Hyperparameter Tuning Job

Related examples:

- SageMaker AI examples - hyperparameter tuning

MLCOST04-BP12 Set up a budget and use resource tagging to track costs

Setting up budgets and implementing resource tagging for machine learning workloads provides clear visibility into your ML-related expenses and optimizes costs across your organization. By

tracking costs effectively, you can make data-driven decisions about resource allocation and identify opportunities for cost optimization.

Desired outcome: You gain complete visibility into your machine learning costs across development, training, and production environments. You can track expenses by project, business unit, or environment, allowing for accurate cost allocation and forecasting. Through tagging and budgeting tools, you can proactively manage your ML spending, receive alerts before exceeding budgeted amounts, and make informed decisions about resource provisioning and termination.

Common anti-patterns:

- Running ML workloads without cost monitoring mechanisms in place.
- Using generic cost tracking that doesn't differentiate between ML projects or environments.
- Failing to tag ML resources consistently, making cost allocation difficult.
- Ignoring budget alerts or failing to take action when exceeding thresholds.

Benefits of establishing this best practice:

- Clear visibility into where ML spending occurs across your organization.
- Ability to accurately allocate costs to specific projects or business units.
- Early warning through alerts when costs exceed or are forecasted to exceed budgeted amounts.
- Improved governance and financial accountability for ML initiatives.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Cost management is a critical aspect of running machine learning workloads in the cloud. Without proper cost tracking and budget controls, ML expenses can quickly escalate due to compute-intensive training jobs, large storage requirements for datasets, and continuous inference endpoints. By implementing comprehensive budgeting and tagging strategies, you gain visibility and control over these costs.

AWS provides several tools that work together to track, analyze, and optimize your ML costs. AWS Budgets allows you to set custom budgets for your SageMaker AI resources, while AWS Cost Explorer provides visualization and analysis capabilities to understand spending patterns. Resource tagging serves as the foundation for detailed cost tracking, enabling you to categorize expenses by project, team, environment, or other dimension important to your organization.

For example, you might tag resources related to a fraud detection model with a Project tag value of FraudDetection and an Environment tag value of Production. This allows you to track the total cost of this specific ML use case across its components, from development notebooks to training jobs to deployment endpoints.

Implementation steps

1. **Set up AWS Budgets for ML cost tracking**. Create customized budgets in AWS Budgets to monitor your Amazon SageMaker AI costs across development, training, and hosting. Configure the budget to track specific services (such as SageMaker AI) or specific tagged resources. Set thresholds for actual costs and forecasted costs to receive notifications before you exceed your budget. This gives you time to make adjustments to your resource usage if needed. Access your budgets through the AWS Budgets console to track progress and make adjustments as necessary.
2. **Implement a tagging strategy for ML resources**. Develop a consistent tagging strategy for all your ML resources. Define mandatory tags such as Project, BusinessUnit, Environment (dev/test/prod), and Owner. Document your tagging standards and verify that team members understand and follow these standards. Apply these tags to relevant resources, including Amazon SageMaker AI notebook instances, training jobs, models, endpoints, and related resources like Amazon S3 buckets for dataset storage.
3. **Activate cost allocation tags**. After implementing your tagging strategy, activate your tags as cost allocation tags in the AWS Billing and Cost Management console. Note that it may take up to 24 hours for newly activated tags to appear in your cost management tools. Once activated, you can use your tags to filter and group costs in AWS Cost Explorer and other cost reporting tools.
4. **Configure detailed cost analysis using AWS Cost Explorer**. Use AWS Cost Explorer to visualize and analyze your ML costs over time. Create custom reports that filter costs by specific tags (like Project or Environment) or by specific services like SageMaker AI. Set up regular reports to track spending trends, identify cost spikes, and understand usage patterns. Use the insights gained to optimize your resource allocation and scheduling for ML workloads.
5. **Create cost anomaly detection**. Set up AWS Cost Anomaly Detection to automatically identify unusual spending patterns in your ML workloads. Configure alerts to notify relevant stakeholders when anomalies are detected. This assists you in quickly identifying and addressing unexpected cost increases, which can happen with ML workloads due to extended training times or inefficient resource usage.

6. **Establish cost governance processes**. Create clear processes for reviewing costs, responding to budget alerts, and making cost optimization decisions. Assign responsibility for cost monitoring to specific individuals or teams. Conduct regular cost reviews with stakeholders to discuss spending trends, identify optimization opportunities, and align ML resource usage with business priorities. Document cost-saving actions taken and their impact on the overall budget.
7. **Optimize ML resources based on cost data**. Use the cost insights gained from your tagging and budgeting tools to optimize ML resource usage. Identify underutilized notebook instances that can be stopped when not in use. Select appropriate instance types based on workload requirements. Consider using Amazon SageMaker AI Managed Spot Training to reduce training costs by up to 90%. Implement auto-scaling for inference endpoints to match capacity with demand.

Resources

Related documents:

- Managing your costs with AWS Budgets
- Organizing and tracking costs using AWS cost allocation tags
- Getting started with AWS Cost Anomaly Detection
- Best Practices for Tagging AWS Resources
- Cost Optimization Pillar - AWS Well-Architected Framework
- Amazon SageMaker AI Pricing
- AWS Cloud Financial Management
- Analyze Amazon SageMaker AI spend and determine cost optimization opportunities based on usage, Part 4: Training jobs

MLCOST04-BP13 Enable data and compute proximity

Positioning data and compute resources in the same AWS Region reduces data transfer costs and improves processing speeds for machine learning workloads. By minimizing the physical distance between data storage and compute resources, you can significantly decrease latency and avoid cross-region data transfer fees.

Desired outcome: You achieve cost-efficient and high-performance machine learning operations by placing your data and compute resources in the same AWS Region. You experience faster training times, reduced latency, and avoid unnecessary data transfer costs that can significantly impact your ML project budgets.

Common anti-patterns:

- Storing data in one Region and running compute resources in another Region.
- Repeatedly transferring large datasets across Regions for training or inference.
- Failing to consider the impact of data transfer costs on overall ML project budgets.

Benefits of establishing this best practice:

- Decreased latency for data access during model training and inference.
- Improved overall machine learning workflow performance.
- Simplified management of data compliance and sovereignty requirements.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Data transfer costs between AWS Regions can significantly impact your machine learning project's budget, especially when working with large datasets that are repeatedly accessed during model training. By keeping your compute resources in the same Region as your data storage, you minimize these costs and improve performance.

When planning your machine learning infrastructure on AWS, consider data locality as a primary design principle. For example, if your organization stores datasets in Amazon S3 buckets in the US West (Oregon) Region, you should provision EC2 instances, SageMaker AI notebooks, or other ML compute resources in that same Region.

This principle applies to various machine learning scenarios, including model training, data preprocessing, and inference. Even though AWS provides high-speed network connections between Regions, the laws of physics still impose latency limitations, and cross-Region data transfers incur additional costs that can be avoided.

Implementation steps

1. **Identify data storage locations**. Determine where your primary data is stored on AWS. Check which Regions contain your Amazon S3 buckets, Amazon EFS file systems, or other storage services holding your training data. Use the AWS Management Console, AWS CLI, or infrastructure as code tools to inventory your data storage resources across Regions.
2. **Audit compute resource placement**. Review your current machine learning compute resources, including Amazon EC2 instances, Amazon SageMaker AI notebooks, and training jobs. Verify

if they are in the same Regions as your data sources. Use AWS Cost Explorer and AWS Trusted Advisor to identify cross-Region data transfer costs that may indicate misaligned resources.

3. **Consolidate resources by Region**. When creating new compute resources for machine learning workloads, consistently provision them in the same Region as your data. For example, if using Amazon SageMaker AI, create your notebook instances, training jobs, and endpoints in the Region where your training data is stored in Amazon S3.
4. **Use Regional data transfer analysis**. Review your AWS billing information to identify and quantify cross-Region data transfer costs. The AWS Cost Explorer service can assist you in analyzing data transfer costs between AWS services and across Regions. Set up cost allocation tags to track expenses related to machine learning projects specifically.
5. **Consider data replication for specific use cases**. In scenarios requiring multi-Region deployments for high availability or disaster recovery, implement a data replication strategy to maintain copies of datasets in each Region where compute resources exist. Services like Amazon S3 Cross-Region Replication can automate this process while managing costs.
6. **Leverage edge computing for distributed ML workloads**. When working with data that exists at the edge of the network, consider using AWS Outposts, AWS Wavelength, or AWS Local Zones to bring compute resources closer to your data sources, especially for applications requiring low-latency inference.
7. **Implement data caching strategies**. For frequently accessed data, implement caching solutions like Amazon ElastiCache or Amazon DynamoDB Accelerator (DAX) in the same Region as your compute resources to further reduce latency and data transfer costs.

Resources

Related documents:

- AWS Cost Explorer
- Replicating objects within and across Regions
- AWS Data Transfer Pricing
- AWS Local Zones
- AWS Outposts
- Regions and Zones
- AWS Global Infrastructure

MLCOST04-BP14 Select optimal algorithms

Selecting optimal algorithms for machine learning (ML) workloads is crucial for balancing cost efficiency and performance. By identifying appropriate ML paradigms and carefully evaluating algorithmic choices, you can optimize both technical performance and business outcomes while managing costs.

Desired outcome: You are able to identify the most suitable ML algorithm for your specific use case that balances accuracy, explainability, computational requirements, and cost efficiency. You can conduct effective trade-off analyses between different approaches and use AWS services to optimize algorithm selection, training, and deployment.

Common anti-patterns:

- Using complex deep learning solutions without first exploring simpler algorithms.
- Ignoring the explainability requirements of the business use case.
- Failing to consider data constraints when selecting algorithms.
- Not evaluating computational and maintenance costs alongside accuracy metrics.

Benefits of establishing this best practice:

- Reduced computational costs by using algorithms appropriate for the specific problem.
- Improved model performance through systematic comparison of algorithm options.
- Enhanced model explainability when required by business stakeholders.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Selecting the optimal algorithm requires understanding your specific ML problem type and the business constraints around it. Begin by categorizing your problem into basic ML paradigms: supervised learning (for labeled data), unsupervised learning (for unlabeled data), or reinforcement learning (for sequential decision problems). Consider what matters most for your use case, whether it's prediction accuracy, model explainability, inference speed, or a balance of these factors.

Algorithm selection significantly impacts both the performance and cost efficiency of your ML solutions. A computationally expensive algorithm might deliver marginally better accuracy but at substantially higher operational costs. Similarly, a complex but highly accurate algorithm might sacrifice the explainability needed for regulatory adherence or business transparency. Finding

the right balance requires systematic experimentation and evaluation against your business requirements.

AWS provides various services test, compare, and optimize algorithms, allowing you to make data-driven decisions about which approach delivers the best value for your specific use case.

Implementation steps

1. **Define your machine learning problem type**. Categorize your problem as supervised learning (classification, regression), unsupervised learning (clustering, dimensionality reduction), or reinforcement learning. This initial classification narrows down the appropriate algorithms to consider.
2. **Determine business requirements and constraints**. Document specific accuracy targets, explainability needs, inference time requirements, and budget constraints. These requirements will serve as criteria for evaluating algorithm options.
3. **Start with simple algorithms first**. Begin experimentation with simpler algorithms like linear or logistic regression, decision trees, or k-means clustering before moving to more complex approaches. These algorithms are computationally efficient, simpler to interpret, and establish important baselines for performance comparison.
4. **Conduct structured experimentation**. Use Amazon SageMaker AI Experiments to track different algorithm trials, hyperparameter configurations, and their results. This creates reproducibility and facilitates comparison between approaches.
5. **Perform comprehensive trade-off analysis**. When comparing algorithms, consider multiple dimensions beyond accuracy:
   - Data requirements (amount needed for training)
   - Computational resources required for training and inference
   - Model explainability and interpretability
   - Deployment complexity and operational overhead
   - Long-term maintenance costs
6. **Use AWS optimized algorithms and frameworks**. Use Amazon SageMaker AI built-in algorithms that are optimized for performance and cost-efficiency on AWS infrastructure. AWS also provides optimized versions of popular frameworks like TensorFlow, PyTorch, and MXNet that include performance enhancements for training across Amazon EC2 instance families.
7. **Consider automated ML approaches**. For exploratory projects or when seeking optimal performance with minimal manual tuning, use SageMaker AI Canvas for rapid algorithm prototyping with the ability to export generated code to notebooks for further customization.

8. **Explore pre-trained models**. Search AWS Marketplace for pre-trained models that can accelerate development through transfer learning or direct deployment. Pre-trained models can significantly reduce computational costs and development time.
9. **Implement continuous evaluation**. As new algorithms and model versions emerge, periodically reassess whether your chosen approach remains optimal. Business requirements and available technologies evolve over time.
10. **Document algorithm selection rationale**. Create clear documentation explaining why specific algorithms were selected, what trade-offs were accepted, and how these decisions align with business requirements.
11. **For generative AI projects, consider foundation models from Amazon Bedrock for natural language processing, image generation, and other tasks where these models can provide state-of-the-art performance with lower development costs.** Use techniques like prompt engineering and fine-tuning to adapt foundation models to your specific business needs while avoiding the computational expense of training from scratch.

Resources

Related documents:

- Built-in algorithms and pretrained models in Amazon SageMaker AI
- Accelerate generative AI development using managed MLflow on Amazon SageMaker AI
- How custom models work
- Types of Algorithms
- SageMaker AI JumpStart pretrained Models

Deployment

Best practices

- MLCOST05-BP01 Use an appropriate deployment option
- MLCOST05-BP02 Explore cost effective hardware options
- MLCOST05-BP03 Right-size the model hosting instance fleet

MLCOST05-BP01 Use an appropriate deployment option

Use the right deployment option for your machine learning models to optimize cost and performance based on your specific use case requirements. Select real-time inference for low latency applications, batch transform for large datasets, or edge deployment for applications that require local processing.

Desired outcome: You have an optimized model deployment strategy that balances performance and cost efficiency. You can choose the appropriate deployment option based on your specific use case requirements, whether that's real-time inference for low-latency applications, batch processing for large datasets, or edge deployment for scenarios requiring local processing.

Common anti-patterns:

- Using real-time endpoints for deployment scenarios regardless of traffic patterns.
- Overlooking serverless or asynchronous options when they would be more cost-effective.
- Deploying separate endpoints for each model when multiple models could be hosted more efficiently together.
- Running inference in the cloud when edge deployment would be more efficient for local data processing.
- Overprovisioning compute resources for inference endpoints.

Benefits of establishing this best practice:

- Cost optimization through selection of the most efficient deployment option for each use case.
- Improved performance by matching deployment options to specific latency requirements.
- Increased operational efficiency through managed inference services.
- Flexibility to handle varying inference workloads and traffic patterns.
- Simplified ML model management across cloud and edge environments.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

When deploying machine learning models, selecting the right deployment option is crucial for achieving optimal performance and cost efficiency. Amazon SageMaker AI provides multiple deployment options that can be tailored to your specific use case requirements.

Real-time inference is ideal for applications requiring low latency responses, such as real-time recommendations or fraud detection. Batch transform is better suited for processing large datasets in offline mode, such as document processing or periodic scoring jobs. Edge deployment brings inference capabilities directly to edge devices, reducing latency and bandwidth requirements while enabling offline processing.

Consider the pattern of requests your application needs to handle. If you need consistent, low-latency responses for interactive applications with steady traffic, real-time inference is appropriate. If you process data in batches without immediate response requirements, batch transform offers cost efficiency. For applications with unpredictable or bursty traffic patterns, serverless inference can automatically scale to match demand while minimizing costs during idle periods. For workloads with large payloads or long processing times, asynchronous inference provides a queuing mechanism that improves efficiency.

Also consider resource utilization. Multi-model endpoints and multi-container endpoints enable you to optimize costs by sharing resources across multiple models or containers. This approach is particularly valuable when you have many models with variable usage patterns or complementary resource requirements.

Implementation steps

1. **Evaluate your inference requirements**. Determine your application's needs for latency, throughput, payload size, and traffic patterns. Consider whether your application requires real-time responses or can process data in batches. Assess if your models should run in the cloud or at the edge based on connectivity, latency requirements, and data privacy considerations.
2. **Use Amazon SageMaker AI for model deployment**. Amazon SageMaker AI offers a comprehensive set of deployment options to optimize price-performance for most use cases. It's a fully managed service that integrates with MLOps tools for effective model management in production with reduced operational burden.
3. **Select the appropriate inference option based on your use case**. Choose from several SageMaker AI inference options:
   - Amazon SageMaker AI Real-time Inference for low-latency, interactive applications requiring immediate responses
   - Amazon SageMaker AI Serverless Inference for workloads with intermittent or unpredictable traffic patterns
   - Amazon SageMaker AI Asynchronous Inference for large payload sizes, long processing times, or when immediate responses aren't required
   - Amazon SageMaker AI Batch Transform for offline processing of large datasets

4. **Implement multi-model endpoints for cost optimization**. Use Amazon SageMaker AI Multi-Model Endpoints to deploy multiple models on a single endpoint with shared container resources. This approach improves endpoint utilization and reduces hosting costs compared to single-model endpoints. SageMaker AI manages the loading of models into memory and scales them based on traffic patterns.
5. **Deploy multiple containers on a single endpoint**. Implement SageMaker AI multi-container endpoints to deploy multiple containers using different models or frameworks on a single endpoint. Run containers in sequence as an inference pipeline or access each container individually through direct invocation to improve endpoint utilization and optimize costs.
6. **Automate endpoint changes through a pipeline**. Use Amazon SageMaker AI Pipelines to automate the model deployment process. Create CI/CD pipelines that handle model training, evaluation, and deployment, enabling consistent and repeatable deployment processes.
7. **Monitor and optimize your deployment**. Implement continuous monitoring of your inference endpoints to track performance metrics, cost, and resource utilization. Use this data to fine-tune your deployment strategy and make adjustments as needed to optimize for cost efficiency and performance.
8. **Use AI-powered code generation for deployment automation**. Use AI-powered development tools like Amazon Q Developer and Kiro to generate deployment scripts, automate infrastructure configuration, and accelerate the implementation of optimal deployment strategies.
9. **For generative AI workloads, consider deployment options for foundation models**. Evaluate specialized deployment options like Amazon Bedrock for fully managed foundation models or SageMaker AI JumpStart for pre-trained models with optimized deployment configurations.

Resources

Related documents:

- Deploy models for inference
- Deploy models with Amazon SageMaker AI Serverless Inference
- Multi-model endpoints
- Batch transform for inference with Amazon SageMaker AI
- Hosting options
- Asynchronous inference
- Model deployment at the edge with SageMaker AI Edge Manager
- Inference pipelines in Amazon SageMaker AI

Related examples:

- SageMaker AI Serverless Inference Walkthrough
- SageMaker AI Edge Manager Workshop
- SageMaker AI Asynchronous Inference Examples

MLCOST05-BP02 Explore cost effective hardware options

Machine learning models that power AI applications are becoming increasingly complex resulting in rising underlying compute infrastructure costs. Up to 90% of the infrastructure spend for developing and running ML applications is often on inference. Look for cost-effective infrastructure solutions for deploying their ML applications in production.

Desired outcome: You achieve significant cost savings while maintaining or improving the performance of your machine learning inference workloads. By implementing cost-effective hardware options, you optimize your infrastructure spend, reduce operational costs, and can allocate resources more efficiently across your ML applications. Your ML models run on purpose-built hardware that provides the right balance of performance and cost for your specific use case.

Common anti-patterns:

- Using general-purpose compute instances for ML workloads without considering specialized hardware options.
- Over-provisioning inference resources to handle peak loads without implementing scaling strategies.
- Ignoring model optimization opportunities before deploying to production.
- Selecting hardware based solely on performance metrics without considering cost-efficiency.

Benefits of establishing this best practice:

- Reduced infrastructure costs for ML model inference.
- Improved inference throughput and latency.
- More efficient use of computational resources.
- Lower total cost of ownership for AI applications.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Machine learning inference costs represent a significant portion of the total expenses associated with running ML workloads in production. As models become more complex, their computational requirements increase, which can lead to higher infrastructure costs. Selecting the right hardware for your ML workloads is crucial for maintaining cost efficiency without sacrificing performance.

AWS offers multiple options to optimize the cost and performance of your ML inference workloads. These include services that optimize models for specific hardware targets, instances that provide cost-effective acceleration for inference workloads, and deployment options that match your specific latency and throughput requirements.

Evaluating your specific workload requirements is essential before selecting hardware options. Consider factors such as latency requirements, throughput needs, model complexity, batch size capabilities, and budget constraints. This evaluation will assist you to determine the most appropriate hardware solution for your use case.

Implementation steps

1. **Use Amazon SageMaker AI Neo for model optimization**. Amazon SageMaker AI Neo automatically optimizes machine learning models for inference on cloud instances and edge devices. For inference in the cloud, SageMaker AI Neo speeds up inference and saves cost by creating an inference optimized container in SageMaker AI hosting. For inference at the edge, SageMaker AI Neo saves developers months of manual tuning by automatically tuning the model for the selected operating system and processor hardware. Neo optimizes models trained in TensorFlow, PyTorch, MXNet, and other frameworks for deployment on ARM, Intel, and NVIDIA processors.
2. **Deploy on Amazon EC2 Inf2 Instances**. Amazon EC2 Inf1 instances deliver high-performance ML inference at the lowest cost in the cloud. They deliver up to 2.3-times higher throughput and up to 70% lower cost per inference than comparable current generation GPU-based Amazon EC2 instances. Inf1 instances are built from the ground up to support machine learning inference applications. They feature up to 16 AWS Inferentia chips, high-performance machine learning inference chips designed and built by AWS. Additionally, Inf1 instances include second generation Intel Xeon Scalable processors and up to 100 Gbps networking to deliver high throughput inference.
3. **Explore Amazon EC2 Inf2 Instances**. The second generation of AWS Inferentia-based instances, EC2 Inf2 instances, offer even greater performance improvements over previous generations. These instances are powered by AWS Inferentia2 chips and provide up to 4x higher throughput

and up to 10x lower latency than Inf1 instances. They're ideal for more complex generative AI models and large language models (LLMs) that require high performance and cost-effective inference solutions.

4. **Consider Amazon SageMaker AI serverless inference**. SageMaker AI serverless inference is a purpose-built inference option that automatically provisions, scales, and shuts down compute capacity based on your workload needs. This pay-per-use model can reduce costs by avoiding the need to continuously run instances when there are no inference requests, making it ideal for workloads with intermittent traffic patterns.
5. **Evaluate batch and asynchronous inference options**. For non-real-time inference requirements, consider using SageMaker AI batch transform for offline inference processing or SageMaker AI asynchronous inference for workloads that can tolerate higher latency. These options often allow for more efficient resource utilization and lower costs compared to real-time inference endpoints.
6. **Implement automated scaling policies**. Configure auto-scaling for your SageMaker AI endpoints to dynamically adjust the number of instances based on workload demands. This way, you can pay for the resources you need while maintaining performance requirements during peak usage periods.
7. **Use enhanced SageMaker AI Inference Recommender**. Use SageMaker AI Inference Recommender with enhanced algorithms and support for multi-model endpoints to get sophisticated cost optimization recommendations for your specific workloads.
8. **Regularly monitor and analyze inference costs**. Use AWS Cost Explorer and Amazon CloudWatch metrics to track your inference costs and performance metrics. Regularly review this data to identify optimization opportunities and adjust your hardware strategy accordingly.

Resources

Related documents:

- Model performance optimization with SageMaker AI Neo
- Amazon EC2 Inf2 Instances
- AWS Inferentia
- Amazon SageMaker AI Inference Recommender
- Deploy models for inference
- Deploy models with Amazon SageMaker AI Serverless Inference
- Asynchronous inference

Related examples:

- AWS Neuron SDK Examples for Inferentia and Trainium instances

MLCOST05-BP03 Right-size the model hosting instance fleet

Use efficient compute resources to run models in production. In many cases, up to 90% of the infrastructure spend for developing and running an ML application is on inference, making it critical to use high-performance, cost-effective ML inference infrastructure. Selecting the right way to host and the right type of instance can have a large impact on the total cost of ML projects. Use automatic scaling for your hosted models. Auto scaling dynamically adjusts the number of instances provisioned for a model in response to changes in your workload.

Desired outcome: You optimize your ML infrastructure costs while maintaining performance by using the right instance types and quantities for your model deployments. You use automated tools to recommend the most cost-effective configurations and implement dynamic scaling that adjusts capacity based on actual demand patterns, resulting in significant cost savings and consistent performance.

Common anti-patterns:

- Using the same instance types for each model regardless of their specific requirements.
- Maintaining static instance counts rather than scaling with workload demands.
- Selecting instance types based solely on performance without considering cost implications.
- Not distributing model instances across multiple availability zones for resilience.

Benefits of establishing this best practice:

- Reduced ML infrastructure costs by up to 90% through optimal instance selection.
- Improved model performance through use of appropriately sized resources.
- Enhanced reliability through automatic scaling and multi-AZ deployment.
- Better handling of variable workloads without performance degradation.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Optimizing ML inference costs requires a careful balance between performance and cost. When selecting compute resources for model hosting, consider both the model's specific requirements and the expected workload patterns. CPU instances may be sufficient for many traditional ML models, while GPU instances deliver better performance for deep learning models but at a higher cost. The key is using the right resource for the specific workload.

Amazon SageMaker AI provides tools that can automatically select the optimal instance type and size for your models. By testing different configurations, you can find the sweet spot that delivers the required performance at the lowest possible cost. Additionally, implementing auto scaling assists in verifying that your deployment can handle varying loads efficiently, scaling up during peak demand and down during quiet periods to avoid unnecessary costs.

Implementation steps

1. **Use Amazon SageMaker AI Inference Recommender for instance selection**. Amazon SageMaker AI Inference Recommender automatically selects the right compute instance type, instance count, container parameters, and model optimizations for inference to maximize performance and minimize cost. You can use SageMaker AI Inference Recommender from SageMaker AI Studio, the AWS Command Line Interface (AWS CLI), or the AWS SDK, and within minutes, get recommendations to deploy your ML model. You can then deploy your model to one of the recommended instances or run a fully managed load test on a set of instance types you choose without worrying about testing infrastructure. You can review the results of the load test in SageMaker AI Studio and evaluate the tradeoffs between latency, throughput, and cost to select the most optimal deployment configuration.
2. **Configure auto scaling for SageMaker AI endpoints**. Amazon SageMaker AI supports an auto scaling feature that monitors your workloads and dynamically adjusts the capacity to maintain steady and predictable performance at the lowest possible cost. When the workload increases, auto scaling brings more instances online. When the workload decreases, auto scaling removes unnecessary instances, which can reduce your compute cost. SageMaker AI automatically attempts to distribute your instances across Availability Zones. So, we strongly recommend that you deploy multiple instances for each production endpoint for high availability. If you're using a VPC, configure at least two subnets in different Availability Zones so Amazon SageMaker AI can distribute your instances across those Availability Zones.
3. **Implement proper scaling policies**. Define appropriate scaling policies based on your model's performance characteristics and usage patterns. Set scaling metrics such as CPU utilization,

GPU utilization, model latency, or custom metrics that reflect your workload's needs. Define appropriate target values and cooldown periods to avoid rapid scaling oscillations.

4. **Consider serverless inference options**. For workloads with unpredictable or intermittent traffic patterns, evaluate Amazon SageMaker AI Serverless Inference, which automatically provisions and scales compute capacity based on traffic. This option reduces the need to select instance types or manage scaling policies while providing pay-per-use pricing.
5. **Regularly review and optimize deployments**. Set up a process to periodically review your model deployments' performance and cost metrics. As your models evolve and usage patterns change, rerun Inference Recommender tests to keep your infrastructure optimized. Look for opportunities to consolidate models or use multi-model endpoints where appropriate.
6. **Use SageMaker AI Training Plans for predictable access**. Use SageMaker AI Training Plans as a compute reservation system for predictable access to high-demand GPU resources, managing large-scale AI training workloads more efficiently with better resource planning and scheduling capabilities.
7. **Use model optimization techniques**. For large language models and other generative AI workloads, consider techniques like quantization, distillation, or pruning to reduce model size and computational requirements. Amazon SageMaker AI supports optimization techniques through SageMaker AI Neo and integration with AWS Neuron for optimized inference on AWS Inferentia and Trainium chips.

Resources

Related documents:

- Amazon SageMaker AI Inference Recommender
- Automatic scaling of Amazon SageMaker AI models
- Deploy models with Amazon SageMaker AI Serverless Inference
- Multi-model endpoints
- Model performance optimization with SageMaker AI Neo

Related examples:

- SageMaker AI Inference Recommender
- Right-sizing your Amazon SageMaker AI Endpoints
- SageMaker AI Serverless Inference Examples
- Automatically Scale Amazon SageMaker AI Models

Monitoring

Best practices

- MLCOST06-BP01 Monitor usage and cost by ML activity
- MLCOST06-BP02 Monitor return on investment for ML models
- MLCOST06-BP03 Monitor endpoint usage and right-size the instance fleet
- MLCOST06-BP04 Enable debugging and logging

MLCOST06-BP01 Monitor usage and cost by ML activity

Use cloud resource tagging to manage, identify, organize, search for, and filter resources. Tags categorize resources by purpose, owner, environment, or other criteria. Associate costs with resources using ML activity categories, such as re-training and hosting, by using tagging to manage and optimize cost in deployment phases. Tagging can be useful for generating billing reports with breakdown of cost by associated resources.

Desired outcome: You gain visibility into your machine learning costs by activity type, allowing for better allocation, forecasting, and optimization of ML resources. You can track expenses across different phases of the ML lifecycle including development, training, and deployment. This enables data-driven decisions about resource allocation and identifies cost-saving opportunities while maintaining performance.

Common anti-patterns:

- Using default AWS account structure without proper tagging strategy for ML resources.
- Not separating costs between development, training, and production environments.
- Failing to automate tagging as part of resource provisioning.
- Overlooking unused or idle resources that continue to incur costs.

Benefits of establishing this best practice:

- Clear visibility into costs associated with different ML activities.
- Ability to allocate costs to appropriate business units or projects.
- Improved forecasting and budgeting for ML initiatives.
- Identification of cost-saving opportunities across the ML lifecycle.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Monitoring and optimizing costs for machine learning workloads requires a systematic approach to resource tagging and usage tracking. ML workloads typically have distinct phases—development, training, inference, and experimentation—each with different resource requirements and cost profiles. By implementing a comprehensive tagging strategy, you can track and attribute costs to specific ML activities, making it more straightforward to understand where your cloud spend is going and identify opportunities for optimization.

AWS provides various tools and services to implement cost monitoring for ML workloads. With proper tagging, you can generate detailed cost reports, set up budgets with alerts, and make data-driven decisions about resource allocation. This practice is particularly important for ML workloads, which can be compute-intensive and potentially costly if not properly managed.

Implementation steps

1. **Establish a tagging strategy for ML resources**. Create a consistent tagging schema that captures relevant dimensions for ML activities. Include tags for project name, environment (development, testing, and production), ML phase (training, inference, and experiment), owner, and cost center. Document this strategy and verify that your team members understand and follow it when creating resources.
2. **Implement AWS tagging**. A tag is a label that you or AWS assigns to an AWS resource. Each tag consists of a key and a value. For each resource, each tag key must be unique, and each tag key can have only one value. You can use tags to organize your resources, and cost allocation tags to track your AWS costs on a detailed level. AWS uses the cost allocation tags to organize your resource costs on your cost allocation report. This streamlines categorizing and tracking your AWS costs. AWS provides two types of cost allocation tags, an AWS-generated tag and user-defined tags.
3. **Activate cost allocation tags**. After creating your tags, you need to activate them for cost tracking in the AWS Billing and Cost Management and Cost Management console. Note that it can take up to 24 hours for new tags to appear in your billing reports.
4. **Automate resource tagging**. Use AWS CloudFormation templates, AWS CDK, or Terraform to automate the application of tags when provisioning resources. For SageMaker AI resources, implement tagging in your deployment pipelines and notebook initialization scripts. Consider using AWS Tag Editor for bulk tagging operations on existing resources.

5. **Use AWS Budgets to keep track of cost**. AWS Budgets can track your Amazon SageMaker AI cost, including development, training, and hosting. You can also set alerts and get a notification when your cost or usage exceeds (or is forecasted to exceed) your budgeted amount. After you create your budget, you can track the progress on the AWS Budgets console.
6. **Implement cost monitoring and reporting**. Use AWS Cost Explorer to visualize and analyze your ML costs across different dimensions. Create custom reports filtered by your ML activity tags to understand spending patterns. Schedule regular exports of cost reports for stakeholders review.
7. **Establish cost optimization processes**. Regularly review resource utilization and costs to identify optimization opportunities. Implement automated shutdown of idle resources such as SageMaker AI notebooks when not in use. Consider using SageMaker AI Managed Spot Training to reduce training costs by up to 90%.
8. **Create governance for tagging**. Use AWS Config Rules or AWS CloudFormation Hooks to enforce tagging policies. Implement processes to review and correct untagged or incorrectly tagged resources. Consider using AWS Organizations Tag Policies to standardize tags across multiple accounts.
9. **Implement enhanced cost tracking with improved tagging**. Use enhanced AWS tagging capabilities with better automation and governance features to make your cost allocation more consistent across ML workloads and improve your visibility into spending patterns.
10. **Use cost optimization services**. Use AWS Cost Anomaly Detection to identify unusual spending patterns in your ML workloads. Consider AWS Compute Optimizer for recommendations on right-sizing your ML compute resources based on historical utilization data.

Resources

Related documents:

- Organizing and tracking costs using AWS cost allocation tags
- Managing your costs with AWS Budgets
- AWS Cost Explorer
- Getting started with AWS Cost Anomaly Detection
- What is Tag Editor?
- Best Practices for Tagging AWS Resources
- Analyze Amazon SageMaker AI spend and determine cost optimization opportunities based on usage, Part 1: Overview
- Analyze Amazon SageMaker AI spend and determine cost optimization opportunities based on usage, Part 4: Training jobs

MLCOST06-BP02 Monitor return on investment for ML models

Once a model is deployed into production, establish a reporting capability to track the value which is being delivered. For example:

- **If a model is used to support customer acquisition:** How many new customers are acquired and what is their spend when the model's advice is used compared with a baseline?
- **If a model is used to predict when maintenance is needed:** What savings are being made by optimizing the maintenance cycle?

Effective reporting compares the value delivered by an ML model against the ongoing runtime cost and to take appropriate action. If the ROI is substantially positive, are there ways in which this might be scaled to similar challenges, for example. If the ROI is negative, could this be addressed by remedial action, such as reducing the model latency by using serverless inference, or reducing the run time cost by changing the compromise between model accuracy and model complexity, or layering in an additional simpler model to triage or filter the cases that are submitted to the full model.

Desired outcome: By implementing this practice, you establish a clear line of sight between your ML investments and business outcomes. You can continuously track the value delivered by your ML models in terms of measurable business KPIs, enabling data-driven decisions about scaling successful models, optimizing underperforming ones, or sunsetting those with negative ROI. Your organization has transparency into the cost-effectiveness of ML initiatives and can strategically allocate resources based on proven business value.

Common anti-patterns:

- Deploying ML models without defining success metrics or business KPIs.
- Focusing only on technical metrics like accuracy without linking to business outcomes.
- Measuring ROI only once after initial deployment rather than continuously.
- Failing to account for the full costs of ML model operation in ROI calculations.
- Ignoring opportunities to scale successful models to similar business challenges.

Benefits of establishing this best practice:

- Clear visibility into the business value generated by ML investments.
- Ability to make data-driven decisions about model optimization or retirement.

- Improved accountability for ML investments across the organization.
- Better allocation of ML resources to high-impact use cases.
- Enhanced stakeholder confidence in ML initiatives through transparent reporting.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Monitoring the return on investment for your ML models requires an intentional approach that connects technical model performance with tangible business outcomes. You need to establish a continuous feedback loop between model operations and business metrics to understand the true value being generated. This means going beyond traditional ML metrics like accuracy or precision and focusing on how the model's predictions translate into business results.

Start by defining clear business KPIs that your model is expected to influence before deployment. These KPIs should be measurable and directly tied to business objectives, such as increased revenue, reduced costs, or improved customer satisfaction. For customer acquisition models, track metrics like conversion rates, customer lifetime value, and acquisition costs. For predictive maintenance models, measure metrics like maintenance cost savings, reduced downtime, and extended equipment lifespan.

Once deployed, collect data on both the model's performance and these business metrics to establish correlation between the two. Use A/B testing where possible to compare outcomes with and without the model's influence. This can isolate the specific impact of your ML investment against other factors that might affect business outcomes.

Regularly review the ROI of your models and be prepared to take action based on the findings. For models with strong positive ROI, explore opportunities to scale the approach to similar business problems or increase the scope of the current implementation. For models with marginal or negative ROI, consider optimization strategies like reducing inference costs through more efficient infrastructure, simplifying model complexity while maintaining acceptable accuracy, or implementing a multi-tiered approach where simpler models handle routine cases and complex models only process edge cases.

Implementation steps

1. **Define business-oriented success metrics.** Before deploying your ML model, clearly define the business KPIs that will be used to measure its impact. Work with business stakeholders to

connect these metrics directly to business outcomes and measure them practically. For example, for a customer churn prediction model, success metrics might include reduction in churn rate, increase in retention-driven revenue, and decreased cost of retention campaigns.

2. **Establish baseline performance.** Measure and document the current performance on your defined KPIs before implementing the ML model. This baseline is essential for determining the incremental value the model delivers. Consider using A/B testing approaches where feasible, sending some cases through the ML-driven process and others through the traditional approach.
3. **Implement data collection pipelines.** Set up automated data collection for both model metrics and business outcomes. Use AWS services like Amazon CloudWatch to monitor technical aspects of your model and Amazon Kinesis to capture business event data. Store this data in Amazon S3 or Amazon Redshift for further analysis.
4. **Create ROI dashboards using Quick.** Develop business-focused dashboards in Quick that visualize the relationship between model performance and business outcomes. Include metrics that show both the value generated (increased revenue, cost savings) and costs incurred (infrastructure, maintenance, human review). Use QuickSight's ML Insights to automatically identify trends and anomalies in your ROI data.
5. **Schedule regular ROI reviews.** Establish a cadence for reviewing model ROI with both technical and business stakeholders. These reviews should assess whether the model continues to deliver positive business impact and identify opportunities for optimization. Use these sessions to make data-driven decisions about continuing investment, scaling successful approaches, or adjusting underperforming models.
6. **Optimize underperforming models.** For models not meeting ROI targets, implement strategic improvements. Consider Amazon SageMaker AI Serverless Inference to reduce costs for infrequent or variable workloads. Explore model compression techniques like SageMaker AI Neo to improve inference efficiency without sacrificing accuracy. Implement tiered prediction approaches where simple, low-cost models filter cases before routing to more complex models.
7. **Scale successful models.** When models demonstrate strong positive ROI, look for opportunities to expand their impact. Apply similar modeling approaches to related business problems, increase the scope of existing models, or integrate the model with additional business processes to maximize value creation.
8. **Use enhanced QuickSight capabilities for ROI analysis**. Use improved Quick with generative AI insights and natural language query capabilities to automatically identify trends, anomalies, and optimization opportunities in your ROI data.
9. **Use generative AI for enhanced insights.** Use generative AI capabilities through Amazon Bedrock to analyze patterns in your ROI data and suggest optimization strategies. Generative

AI can identify non-obvious correlations between model configurations and business outcomes, leading to better ROI optimization decisions.

Resources

Related documents:

- What is Quick?
- Publish custom metrics
- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- What are AWS Cost and Usage Reports?
- Quick
- Cost Optimization Pillar - AWS Well-Architected Framework
- AWS Cost Explorer

MLCOST06-BP03 Monitor endpoint usage and right-size the instance fleet

Use efficient compute resources to run models in production. Monitor your endpoint usage and right-size the instance fleet. Use automatic scaling (auto scaling) for your hosted models. *Auto scaling* dynamically adjusts the number of instances provisioned for a model in response to changes in your workload.

Desired outcome: You have optimized SageMaker AI endpoints that automatically adjust to workload demands while maintaining performance and minimizing costs. Your model deployment uses appropriately sized instances that are neither over-provisioned nor under-provisioned, and you have continuous monitoring in place to inform scaling decisions.

Common anti-patterns:

- Provisioning static endpoint configurations that remain unchanged regardless of workload fluctuations.
- Over-provisioning instances "just to be safe" without analyzing actual resource utilization.
- Ignoring endpoint metrics and failing to adjust resource allocation based on usage patterns.
- Deploying resources across different Availability Zones without consideration for data transfer costs.

- Using default instance types without evaluating performance requirements.

Benefits of establishing this best practice:

- Reduced compute costs by reducing over-provisioned resources.
- Improved performance during peak usage periods through automatic scaling.
- Higher resource utilization through right-sizing.
- Increased availability by distributing instances across Availability Zones.
- Better understanding of model usage patterns to inform future optimizations.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Monitoring and optimizing your SageMaker AI endpoints is essential for maintaining cost-efficiency while providing high availability and performance. By implementing CloudWatch monitoring and auto scaling, your deployments use only the resources they needs when they need them. Start by establishing baseline metrics for your endpoints to understand typical usage patterns and resource requirements. Then implement auto scaling policies based on these metrics to automatically adjust capacity in response to changing workloads.

For production environments, distribute your endpoint deployment across multiple Availability Zones to maintain high availability. Consider the placement of related resources, such as data storage solutions like FSx for Lustre, to minimize cross-AZ data transfer costs and optimize performance. Regular review of your metrics and scaling configurations assists you to continuously refine your deployment for optimal cost and performance.

Implementation steps

1. **Monitor Amazon SageMaker AI endpoints with Amazon CloudWatch**. You can monitor Amazon SageMaker AI using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. Use metrics such as CPUUtilization, GPUUtilization, MemoryUtilization, and DiskUtilization to view your endpoint's resource utilization and make informed decisions about right-sizing your endpoint instances. Set up CloudWatch dashboards to visualize these metrics over time and identify patterns in resource usage.
2. **Implement CloudWatch alarms for proactive monitoring**. Configure alarms for key metrics that can indicate when an endpoint is under-provisioned or over-provisioned. For example, set up alarms that go off when CPU utilization consistently exceeds 80% (indicating potential

under-provisioning) or remains below 20% (indicating over-provisioning). These alarms can notify your team to take action or run automated responses through AWS Lambda functions.

3. **Configure auto scaling for SageMaker AI endpoints**. Amazon SageMaker AI supports auto scaling that monitors your workloads and dynamically adjusts capacity to maintain steady performance at the lowest possible cost. When workload increases, auto scaling brings more instances online. When workload decreases, auto scaling removes unnecessary instances, which can reduce compute costs. Define appropriate scaling policies based on your application's requirements, including minimum and maximum instance counts, target metrics, and scale-in and scale-out cooldown periods.
4. **Distribute instances across Availability Zones**. SageMaker AI automatically attempts to distribute your instances across Availability Zones, so deploy multiple instances for each production endpoint to provide high availability. If you're using a VPC, configure at least two subnets in different Availability Zones to allow SageMaker AI to distribute your instances across those zones, providing resilience against zone failures.
5. **Optimize resource placement for data access**. When using Amazon FSx for Lustre as an input data source for SageMaker AI, deploy FSx for Lustre and SageMaker AI in the same Availability Zone to avoid cross-AZ data transfer costs. This configuration removes the initial Amazon S3 download step, accelerating ML training jobs while minimizing costs. Consider similar placement strategies for other related resources to optimize performance and cost.
6. **Regularly review and adjust instance types**. Periodically evaluate whether your selected instance types are appropriate for your workload. SageMaker AI offers a variety of instance types optimized for different workload characteristics. Analyze your CloudWatch metrics to determine if you could achieve better price-performance by switching to a different instance family, such as compute-optimized, memory-optimized, or GPU instances.
7. **Use inference optimization techniques**. Implement model optimization techniques such as Amazon SageMaker AI Neo to automatically optimize models for your target hardware, improving performance and potentially allowing you to use smaller instance types. Consider techniques like model compression, quantization, and batching to improve inference efficiency and throughput.
8. **Use enhanced SageMaker AI Inference Recommender**. Use SageMaker AI Inference Recommender with enhanced algorithms and support for multi-model endpoints to get sophisticated instance selection and cost optimization recommendations.
9. **Implement specialized instance types for generative AI models**. For large language models and other generative AI workloads, use specialized instances like AWS Inferentia or AWS Trainium, which are designed specifically for machine learning inference and training. These

instances can provide significant cost savings compared to general-purpose GPU instances when running transformer-based models. Consider Amazon Bedrock for fully managed generative AI capabilities with built-in scaling.

Resources

Related documents:

- Amazon SageMaker AI metrics in Amazon CloudWatch
- Automatic scaling of Amazon SageMaker AI models
- Amazon SageMaker AI Inference Recommender
- AWS Inferentia
- Best practices for deploying models on SageMaker AI Hosting Services
- Data and model quality monitoring with Amazon SageMaker AI Model Monitor

MLCOST06-BP04 Enable debugging and logging

Implementing comprehensive logging and debugging capabilities for your machine learning workflows assists you to understand resource consumption patterns and identify optimization opportunities. By collecting and analyzing runtime metrics, you can reduce costs and enhance the efficiency of your ML training operations.

Desired outcome: You gain visibility into training jobs through metrics and logs that reveal resource consumption patterns. This practice identifies optimization opportunities, reduces costs, and improves ML model training performance. You implement monitoring systems to track compute and storage utilization, and instrument code to record key metrics.

Common anti-patterns:

- Training ML models without performance visibility.
- Ignoring resource consumption data until costs become problematic.
- Deploying ML solutions without adequate logging infrastructure.
- Using manual methods to track performance metrics.
- Waiting for issues to arise before implementing monitoring.

Benefits of establishing this best practice:

- Early identification of model training inefficiencies.

- Reduced compute and storage costs through resource optimization.
- Faster troubleshooting of training job issues.
- Enhanced visibility into ML pipelines.
- Data-driven decisions for infrastructure provisioning.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Proper debugging and logging are crucial for cost management in machine learning workflows. As ML models grow in complexity, the computational resources required for training also increase. By implementing comprehensive monitoring, you can identify inefficiencies, optimize resource allocation, and reduce overall costs.

Effective logging and debugging require instrumentation at multiple levels, from the ML code itself to the underlying infrastructure. This visibility provides an understanding of how resources are being utilized during training jobs and identifies bottlenecks so that you can make data-driven decisions about when and how to scale resources. The metrics and logs collected can reveal patterns of inefficient resource utilization that might otherwise go unnoticed.

Additionally, monitoring storage consumption is important as data preparation and feature engineering can generate large intermediate datasets. By tracking both compute and storage metrics, you can identify opportunities for optimization across your entire ML pipeline.

Implementation steps

1. **Set up Amazon SageMaker AI Debugger**. Amazon SageMaker AI Debugger captures training job states at regular intervals, providing visibility into the ML training process. It monitors, records, and analyzes data during training, enabling you to:
   - Track model parameters, gradients, and tensor values
   - Identify training issues like vanishing gradients or tensor explosions
   - Receive automated alerts for common training problems
   - Visualize and analyze captured data interactively
2. **Implement CloudWatch logging**. Integrate Amazon CloudWatch Logs with your SageMaker AI training jobs to centralize and analyze log data. Configure CloudWatch to:
   - Collect standard output and error logs from training jobs
   - Encrypt log data using an AWS KMS key for security
   - Set up custom log groups and streams for different ML workflows

- Create log retention policies to manage storage costs

3. **Instrument ML code for metrics collection**. Add instrumentation code to your ML training scripts to capture performance metrics and resource utilization data:
   - Track timing information for different training phases
   - Monitor memory usage during training operations
   - Record batch processing statistics and convergence metrics
   - Log hyperparameter values and their impact on training performance
4. **Configure resource monitoring**. Set up monitoring for compute and storage resources used by your ML workflows:
   - Use CloudWatch metrics to track instance utilization
   - Monitor data transfer volumes between storage and compute resources
   - Set up alerts for abnormal resource consumption patterns
   - Create dashboards to visualize resource utilization trends
5. **Implement automated alerting**. Configure notification systems to alert you when resource consumption exceeds expected thresholds:
   - Set up CloudWatch alarms for high CPU, memory, or GPU utilization
   - Create alerts for extended training job durations
   - Configure notifications for storage capacity issues
   - Establish alerting for debugging rule violations in SageMaker AI Debugger
6. **Analyze and optimize training jobs**. Use the collected metrics and logs to identify optimization opportunities:
   - Review resource utilization patterns to identify right-sizing opportunities
   - Analyze training job logs for inefficient code paths
   - Examine data loading and preprocessing bottlenecks
   - Optimize hyperparameters based on performance metrics
7. **Use enhanced debugging capabilities**. Use improved SageMaker AI Studio debugging and monitoring capabilities with better integration to popular ML frameworks and enhanced visualization tools for more efficient troubleshooting.
8. **Use generative AI for log analysis**. Use generative AI capabilities to analyze and extract insights from ML training logs. Utilize Q Diagnostics integrated into the console or your preferred IDE for log analysis.
   - Implement natural language processing to summarize log patterns
   - Use Amazon Bedrock to build intelligent log analysis assistants
   - Deploy ML models that can predict resource needs based on historical data
   - Create automated reports of cost optimization opportunities from log data

Resources

Related documents:

- Amazon SageMaker AI Debugger
- What is Amazon CloudWatch Logs?
- Analyzing log data with CloudWatch Logs Insights
- Logging and Monitoring
- Logging Amazon ML API Calls with AWS CloudTrail
- Amazon SageMaker AI Debugger API

Related examples:

- Debugger example notebooks)
- SageMaker AI Debugger GitHub Repository

Sustainability

The sustainability pillar focuses on environmental impacts, especially energy consumption and efficiency, since they are important levers for architects to guide direct action on how to reduce resource usage.

Each best practice in this section is presented based on its place in the ML lifecycle as detailed in *Machine learning lifecycle*.

Lifecycle phases

- Business goal identification
- ML problem framing
- Data processing
- Model development
- Deployment
- Monitoring

Business goal identification

Best practices

- MLSUS01-BP01 Define the overall environmental impact or benefit

MLSUS01-BP01 Define the overall environmental impact or benefit

Measure your workload's impact and its contribution to the overall sustainability goals of the organization. Understand the environmental footprint of machine learning systems to make informed decisions about resource allocation and optimization.

Desired outcome: You gain a clear understanding of how your ML workload impacts your organization's sustainability objectives, including energy consumption, carbon emissions, and resource utilization. You have established specific sustainability objectives and success criteria to measure against, which assists you when optimizing your ML workloads and demonstrating their environmental value proposition in relation to their impact.

Common anti-patterns:

- Focusing solely on model accuracy without considering environmental trade-offs.
- Implementing ML systems without measuring their carbon footprint.
- Overprovisioning resources for ML workloads.
- Unnecessarily retraining models when not required.

Benefits of establishing this best practice:

- Improved alignment between ML initiatives and organizational sustainability goals.
- Enhanced ability to meet regulatory and reporting requirements.
- Transparent assessment of the sustainability impact of ML projects.
- Identification of opportunities to optimize ML systems for lower environmental impact.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Understanding the environmental impact of your ML workloads is essential for sustainable AI development. ML operations can be resource-intensive, requiring substantial computing power for training and inference, which translates to energy consumption and carbon emissions. By measuring and tracking this impact, you can make informed decisions about architecture choices, region selection, and optimization strategies.

The cloud provides significant advantages for sustainable ML development through its economies of scale and renewable energy investments. By leveraging AWS's efficiency, you can reduce your carbon footprint compared to on-premises alternatives. However, you still need to make conscious decisions about how you design, deploy, and operate your ML workloads.

Consider the full lifecycle of your ML system, from data collection and processing to model training, deployment, and ongoing operations. Each stage has different resource requirements and environmental impacts. For example, training large models is typically more computationally intensive than inference, but if your model serves millions of predictions daily, the inference stage might have a larger cumulative impact over time.

Implementation steps

1. **Establish sustainability objectives for ML workloads**. Begin by defining specific sustainability goals for your ML projects that align with your organization's broader environmental commitments. Determine what metrics you'll track (for example, carbon emissions per training run or energy efficiency per inference) and set concrete targets for improvement.

2. **Assess the environmental impact of data processing**. Evaluate how much data you're storing and processing for your ML workloads. Consider implementing data lifecycle management using Amazon S3 Lifecycle policies to automatically transition infrequently accessed data to more efficient storage classes or archive data that's no longer needed for active training.
3. **Measure the impact of model training**. Calculate the computational resources and associated carbon emissions of your model training processes using tools like the AWS Customer Carbon Footprint Tool. Consider factors such as instance types, training duration, and region selection. Track these metrics over time to identify trends and opportunities for improvement.
4. **Optimize model training frequency and approach**. Determine how often retraining is truly necessary based on model drift monitoring. Implement incremental training where possible using Amazon SageMaker AI to update existing models with new data rather than retraining from scratch. Use techniques like transfer learning to leverage pre-trained models and reduce computational requirements.
5. **Assess inference efficiency**. Evaluate the environmental impact of your deployed models in production. Consider techniques like model compression, quantization, and distillation to reduce the computational requirements of inference while maintaining acceptable accuracy. Use Amazon SageMaker AI Neo to automatically optimize models for specific deployment targets.
6. **Calculate the net sustainability value**. Compare the environmental impact of your ML workload with its benefits, including the sustainability improvements it enables. For example, if your ML system optimizes energy usage in manufacturing processes, calculate the net reduction in emissions to demonstrate its overall positive impact.
7. **Implement ongoing monitoring and reporting**. Establish a regular cadence for reviewing sustainability metrics and reporting on progress toward your goals. Use AWS CloudWatch to monitor resource utilization and create dashboards to track your sustainability KPIs over time.
8. **For GenAI workloads, consider your model customization strategy**. When implementing generative AI solutions, evaluate whether fine-tuning existing foundation models like Amazon Bedrock models is more environmentally efficient than training custom models from scratch. Consider prompt engineering and retrieval-augmented generation (RAG) approaches that can achieve business goals with lower computational intensity than full model training.

Resources

Related documents:

- AWS Customer Carbon Footprint Tool
- AWS Sustainability Data Initiative

- AWS Well-Architected Framework: Sustainability Pillar
- Corporate Climate Action - Science Based Targets initiative (SBTi)
- Greenhouse Gas Protocol
- Optimize AI/ML workloads for sustainability: Part 1, identify business goals, validate ML use, and process data

ML problem framing

Best practices

- MLSUS02-BP01 Consider AI services and pre-trained models
- MLSUS02-BP02 Select sustainable Regions

MLSUS02-BP01 Consider AI services and pre-trained models

Using AI services and pre-trained models can significantly reduce the resources needed for machine learning workloads, enabling you to quickly implement AI capabilities without developing custom models from scratch.

Desired outcome: You identify opportunities to use managed AI services or pre-trained models instead of building custom models, reducing the environmental impact of training and deploying ML solutions while accelerating time to market. By using existing capabilities through APIs or fine-tuning pre-trained models, you minimize computational resources required for data preparation, model training, and deployment.

Common anti-patterns:

- Building custom models from scratch when suitable managed services already exist.
- Collecting and processing large datasets unnecessarily when pre-trained models could be utilized.
- Ignoring transfer learning opportunities that could reduce training time and computational resources.
- Overlooking model optimization techniques that could reduce inference resource requirements.

Benefits of establishing this best practice:

- Reduced environmental impact through decreased computational resource usage.
- Lower operational costs for ML development and deployment.

- Faster time-to-market for ML-powered solutions.
- Access to state-of-the-art models without specialized ML expertise.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

When developing machine learning solutions, evaluate whether you truly need to build a custom model or if existing services and pre-trained models can meet your requirements. Many common use cases like image recognition, natural language processing, or recommendation systems can use pre-built capabilities available through APIs. These managed services handle the underlying infrastructure, data processing, and model maintenance, reducing both environmental impact and operational overhead.

If fully managed services don't meet your specific requirements, consider starting with pre-trained models that you can fine-tune with your data. This approach, known as transfer learning, allows you to benefit from models that have already undergone resource-intensive training on large datasets. By fine-tuning, you can achieve similar or better performance than training from scratch while using significantly fewer computational resources.

For example, instead of training a computer vision model from scratch, you could use a pre-trained image recognition model from Amazon Rekognition or fine-tune a model available through SageMaker AI JumpStart with your specific images. This approach reduces the carbon footprint associated with extensive model training while delivering high-quality results.

Implementation steps

1. **Assess your ML use case requirements**. Before developing an ML solution, clearly define your requirements and success criteria. Understand the specific problem you're trying to solve, the data you have available, and the performance metrics that matter for your application.
2. **Explore AWS AI services**. AWS AI services provide ready-to-use capabilities through APIs for common ML tasks. These services include Amazon Rekognition for image and video analysis, Amazon Comprehend for natural language processing, Amazon Forecast for time-series forecasting, and Amazon Personalize for personalization and recommendations.
3. **Evaluate foundation models in Amazon Bedrock**. Amazon Bedrock provides serverless access to leading foundation models from AI companies like Anthropic, Cohere, AI21 Labs, and Amazon's own Nova models through a single API. These models can handle tasks like text generation, summarization, chatbots, and content creation without requiring model training.

4. **Explore pre-trained models from AWS Marketplace**. AWS Marketplace offers over 1,400 ML-related assets that you can subscribe to, including pre-trained models for various industry-specific use cases that can be deployed with minimal configuration.
5. **Leverage SageMaker AI JumpStart**. SageMaker AI JumpStart provides pre-trained, open-source models for a wide range of problem types to get started with machine learning. You can incrementally train and fine-tune these models before deployment, reducing the computational resources needed compared to training from scratch.
6. **Consider Hugging Face models**. Hugging Face with Amazon SageMaker AI enables you to use thousands of pre-trained transformer models for NLP, computer vision, and audio tasks. These models can be fine-tuned with your specific data to achieve high-quality results with minimal training.
7. **Implement efficient fine-tuning techniques**. When customizing pre-trained models, use efficient fine-tuning methods like parameter-efficient fine-tuning (PEFT), low-rank adaptation (LoRA), or quantization to minimize computational resources while maintaining performance.
8. **Monitor resource usage and optimize**. After deploying your solution, continuously monitor its resource consumption and performance. Look for opportunities to optimize through model compression, quantization, or pruning to further reduce computational requirements.
9. **Leverage expanded pre-trained model libraries**. Use the expanded SageMaker AI JumpStart catalog which now includes broader selection of pre-trained models and industry-specific solutions, reducing the need for custom model development and associated computational resources.
10. **For generative AI workloads, implement retrieval-augmented generation (RAG)**. For generative AI applications requiring domain-specific knowledge, consider using retrieval-augmented generation with SageMaker AI JumpStart foundation models instead of fine-tuning large models, which can significantly reduce computational resources while improving accuracy.

Resources

Related documents:

- AWS AI Services
- SageMaker AI JumpStart pretrained models
- Choose the best data source for your Amazon SageMaker AI training job
- Pre-trained machine learning models available in AWS Marketplace
- Resources for using Hugging Face with Amazon SageMaker AI

- Optimize AI/ML workloads for sustainability: Part 2, model development

MLSUS02-BP02 Select sustainable Regions

Choose the Regions where you implement your workloads based on both your business requirements and sustainability goals.

Desired outcome: You select AWS Regions that align with your organizational sustainability objectives while meeting your business requirements. By choosing Regions with renewable energy sources and lower carbon intensity, you reduce the environmental impact of your machine learning workloads while maintaining optimal performance for your business needs.

Common anti-patterns:

- Selecting Regions based solely on proximity without considering environmental impact.
- Ignoring renewable energy availability when deploying machine learning workloads.
- Deploying workloads across multiple Regions without considering their carbon footprints.

Benefits of establishing this best practice:

- Alignment with organizational sustainability goals and ESG initiatives.
- Enhanced reputation as an environmentally responsible organization.
- Potential cost savings through efficient Region selection.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

When deploying your machine learning workloads, Region selection plays a crucial role in meeting both your operational requirements and sustainability goals. While factors such as latency, data residency, and service availability remain important, incorporating sustainability considerations into your Region selection process can minimize your environmental impact. AWS is continuously expanding its renewable energy projects globally, making it increasingly possible to host your workloads in Regions powered by sustainable energy sources.

The cloud offers significant sustainability advantages compared to on-premises deployments due to higher utilization rates, more energy-efficient infrastructure, and AWS' commitment to

renewable energy. By selecting Regions thoughtfully, you can further enhance these sustainability benefits while still meeting your business needs.

Implementation steps

1. **Understand your business requirements first**. Identify the non-negotiable requirements for your workload, including data sovereignty regulations, compliance-aligned needs, latency requirements, and service availability in specific Regions. Create a shortlist of Regions that meet these baseline requirements.
2. **Research AWS renewable energy projects**. Use the Amazon Around the Globe resource to identify Regions that are near Amazon renewable energy projects. AWS achieved powering its operations with 100% renewable energy in 2023, seven years ahead of their original 2030 commitment.
3. **Consider the grid's carbon intensity**. Look for Regions where the electrical grid has lower published carbon intensity. This information may be available through regional utility reports or sustainability documentation. Lower carbon intensity means reduced emissions even for non-renewable energy sources.
4. **Evaluate the trade-offs**. When selecting Regions, consider potential trade-offs between sustainability goals and business requirements such as latency or availability. In some cases, minor performance trade-offs may be acceptable to achieve significant sustainability improvements.
5. **Monitor sustainability metrics**. After deployment, track relevant sustainability metrics to verify that your Region selection is delivering the expected environmental benefits. Consider implementing dashboards with key performance indicators (KPIs) for sustainability tracking.
6. **Review and adjust periodically**. As AWS adds more renewable energy projects and as your business requirements evolve, periodically reassess your Region selections to continually align with your sustainability goals.

Resources

Related documents:

- AWS Global Infrastructure
- Delivering on net-zero carbon by 2040
- Climate solutions
- AWS Well-Architected Framework - Sustainability Pillar

- How to select a Region for your workload based on sustainability goals

Data processing

Best practices

- MLSUS03-BP01 Minimize idle resources
- MLSUS03-BP02 Implement data lifecycle policies aligned with your sustainability goals
- MLSUS03-BP03 Adopt sustainable storage options

MLSUS03-BP01 Minimize idle resources

Minimize your environmental impact by efficiently managing compute resources in your ML data pipeline. Use serverless architectures that provision resources only when needed, reducing energy consumption and carbon footprint.

Desired outcome: You implement a serverless, event-driven architecture for your ML data pipelines that only provisions resources when work needs to be done. This approach reduces idle resources, optimizes utilization of computing infrastructure, and reduces your organization's environmental impact while maintaining performance and scalability.

Common anti-patterns:

- Provisioning compute instances that run 24/7 regardless of workload requirements.
- Overprovisioning resources rather than scaling dynamically.
- Using traditional batch processing with fixed schedules instead of event-driven approaches.
- Failing to monitor and optimize resource utilization metrics.

Benefits of establishing this best practice:

- Lower carbon footprint and energy consumption.
- Improved resource efficiency across your ML pipeline.
- Automatic scaling to match workload demands.
- Simplified maintenance with managed services.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Adopting a serverless architecture for your ML data pipelines significantly reduces idle resources by allocating compute power only when needed. This approach uses AWS managed services that automatically scale based on workload, avoiding the need to maintain an always-on infrastructure. When you design your data pipeline using serverless technologies like AWS Glue and AWS Step Functions, you not only optimize resource utilization but also distribute the sustainability impact across the tenants of those services, reducing your individual environmental contribution.

The key principle is to transition from a static infrastructure model to an event-driven approach where resources are provisioned in response to triggers. This verifies that compute resources are only active during actual processing tasks rather than sitting idle waiting for work. AWS managed services handle the underlying infrastructure optimization, allowing you to focus on your ML workloads while maintaining efficiency.

Implementation steps

1. **Evaluate your current infrastructure**. Assess your existing data pipeline architecture to identify components that run continuously but have low utilization. Look for workloads with predictable patterns or batch processes that could benefit from an event-driven approach using AWS CloudWatch metrics to identify utilization patterns.
2. **Adopt managed services**. Replace self-managed infrastructure with AWS managed services to distribute sustainability impact across service tenants. Services like AWS Glue, AWS Lambda, and Amazon EMR Serverless provision resources on-demand and automatically scale with your workloads.
3. **Create serverless, event-driven data pipelines**. Use AWS Glue for data processing and AWS Step Functions for orchestration to build ETL and ELT pipelines that only consume resources when triggered. Step Functions can coordinate AWS Glue jobs efficiently, provisioning compute resources only when needed and releasing them immediately after completion.
4. **Implement efficient data storage**. Choose appropriate storage solutions like Amazon S3 for your data lake and Amazon S3 Intelligent-Tiering to automatically move data between access tiers based on usage patterns, reducing storage costs and resource waste.
5. **Configure event-based triggers**. Set up event notifications through Amazon EventBridge to automatically launch processing jobs when new data arrives. This avoids the need for scheduled jobs that might run when no new data is available, reducing unnecessary compute usage.

6. **Optimize compute resources**. For AWS Glue jobs, configure appropriate worker types and dynamically allocate resources based on workload requirements. Use features like AWS Glue Auto Scaling to automatically adjust capacity as needed during jobs.
7. **Implement monitoring and metrics**. Set up comprehensive monitoring of your serverless infrastructure using Amazon CloudWatch to track resource utilization, job execution time, and idle periods. Use these metrics to identify further optimization opportunities.
8. **Establish automatic cleanup processes**. Implement automated processes to remove temporary resources, intermediate data, and other artifacts after job completion to avoid unnecessary storage costs and reduce digital waste.
9. **Optimize data transfer**. Minimize data movement between services by processing data close to where it's stored when possible. Use AWS Glue DataBrew for data preparation tasks within the same environment as your data storage.
10. **Use AI-powered optimization**. Use Amazon Q data integration in AWS Glue to automatically generate optimized ETL jobs for common data sources. This reduces development time while implementing efficient resource utilization patterns from the start.

Resources

Related documents:

- What is AWS Lambda?
- What is Amazon EMR Serverless?
- Using auto scaling for AWS Glue
- What is Amazon EventBridge?
- Start an AWS Glue job with Step Functions
- Serverless Applications Lens - AWS Well-Architected Framework
- Optimize AI/ML workloads for sustainability: Part 3, deployment and monitoring
- Fuel your data with Generative AI

MLSUS03-BP02 Implement data lifecycle policies aligned with your sustainability goals

Classify your data to identify its business relevance and implement efficient storage strategies that support your sustainability goals. By understanding your data's importance, you can appropriately

tier storage, implement retention policies, and manage the entire lifecycle to reduce your environmental footprint while meeting business requirements.

Desired outcome: You have a comprehensive data management strategy that classifies data based on business importance and implements automatic storage optimization. Your workloads store data in the most energy-efficient storage tiers possible, and data that no longer serves a business purpose is automatically purged, resulting in minimal storage footprint and reduced environmental impact.

Common anti-patterns:

- Storing data indefinitely without classification or lifecycle policies.
- Using a single storage tier for data regardless of access patterns.
- Manually managing data retention and deletion.
- Keeping redundant or obsolete data that no longer serves business purposes.

Benefits of establishing this best practice:

- Reduced storage infrastructure and energy consumption.
- Improved data governance and adherence.
- Enhanced system performance with streamlined data management.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Implementing data lifecycle policies begins with understanding the relationship between your data and business outcomes. Each category of data has different requirements for retention, access patterns, and eventual disposal. By aligning these requirements with sustainability goals, you can optimize storage utilization while improving business continuity.

Start by creating a data classification framework that categorizes data based on its criticality, frequency of access, and business value over time. This classification will guide your decisions about which storage tiers to use and when data should be moved or deleted. For instance, frequently accessed operational data might remain in high-performance storage, while rarely accessed archival data can be moved to more energy-efficient cold storage options.

Once you've classified your data, use AWS storage features like S3 Lifecycle policies to automate data transitions between storage tiers and eventual deletion. For example, you can configure

policies that automatically transition data from S3 Standard to S3 Intelligent-Tiering, S3 Standard-IA, S3 One Zone-IA, and eventually to Amazon Glacier or S3 Glacier Deep Archive based on access patterns and age.

Implementation steps

1. **Define your data classification framework.** Develop a comprehensive data classification system that categorizes data based on business importance, access frequency, and regulatory requirements. Include clear definitions for each data category and establish ownership for classification decisions.
2. **Map retention requirements to data classes.** For each data classification, determine appropriate retention periods that satisfy business needs, regulatory requirements, and sustainability goals. Document these requirements to guide policy implementation.
3. **Analyze current storage usage patterns.** Use Amazon S3 Storage Lens to gain visibility into your current storage usage patterns, identifying opportunities for optimization and tracking progress on storage efficiency metrics.
4. **Implement S3 Lifecycle policies.** Configure Amazon S3 Lifecycle policies to automatically transition data between storage classes and enforce deletion timelines based on your defined retention requirements.
5. **Deploy intelligent storage tiering.** Implement Amazon S3 Intelligent-Tiering storage class to automatically move data between access tiers based on changing access patterns, optimizing for both cost and sustainability.
6. **Establish monitoring and reporting.** Create dashboards to track storage optimization metrics, including total storage used, storage class distribution, and lifecycle transition metrics. Regularly review these metrics to identify further optimization opportunities.
7. **Continuously refine data classification.** Review and update your data classification framework regularly to align it with evolving business needs and sustainability goals.

Resources

Related documents:

- Managing the lifecycle of objects
- Managing storage costs with Amazon S3 Intelligent-Tiering
- Comparing the Amazon S3 storage classes
- Assessing your storage activity and usage with Amazon S3 Storage Lens

- Setting an S3 Lifecycle configuration on a bucket
- Data discovery and cataloging in AWS Glue

MLSUS03-BP03 Adopt sustainable storage options

Reduce the volume of data to be stored and adopt sustainable storage options to limit the carbon impact of your workload. For artifacts like models and log files that must be kept for long-term regulatory and audit requirements, use efficient compression algorithms and use energy efficient cold storage.

Desired outcome: You optimize your ML workload storage to minimize environmental impact while improving adherence to regulatory requirements. By implementing efficient storage practices, you reduce data redundancy, properly size resources, use energy-efficient storage options, and implement effective compression techniques. This results in reduced carbon footprint, lower storage costs, and improved overall sustainability of your ML systems.

Common anti-patterns:

- Storing redundant copies of datasets across multiple locations.
- Over-provisioning storage resources for notebooks and instances.
- Using inefficient file formats like CSV instead of columnar formats such as parquet.
- Keeping data in high-performance storage regardless of access frequency.

Benefits of establishing this best practice:

- Lower operational costs through efficient resource utilization.
- Improved data access performance through appropriate format selection.
- Improved ability to adhere to regulatory requirements through proper long-term storage planning.
- Reduced waste through optimization of storage resources.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Storage is a critical component of ML workloads, with large datasets and model artifacts requiring significant resources. By optimizing how you store, compress, and manage this data, you can

substantially reduce the environmental impact of your ML systems. Consider the entire lifecycle of your data, from initial collection through processing to long-term retention. For frequently accessed data, choose efficient formats and compression algorithms. For infrequently accessed data, use Amazon S3 storage tiers that minimize energy consumption while improving your adherence to regulatory requirements. By right-sizing your storage resources and removing redundant data, you can achieve both sustainability goals and cost optimization.

Implementation steps

1. **Reduce redundancy of processed data**. If you can re-create an infrequently accessed dataset, use the Amazon S3 One Zone-IA class to minimize the total data stored. Implement S3 Lifecycle policies to automatically transition objects to more energy-efficient storage tiers based on access patterns.
2. **Right size block storage for notebooks**. Don't over-provision block storage of your notebooks and use centralized object storage services like Amazon S3 for common datasets to avoid data duplication. Monitor usage patterns and adjust storage allocations accordingly.
3. **Use efficient file formats**. Use Parquet to train your models. Compared to CSV, they can reduce your storage by up to 87%. These columnar formats not only reduce storage requirements but also improve query performance.
4. **Migrate to more efficient compression algorithms**. Evaluate different compression algorithms and select the most efficient for your data. For example, Zstandard produces 10–15% smaller files than Gzip at the same compression speed.
5. **Implement storage lifecycle management**. Configure S3 Intelligent-Tiering to automatically move data between access tiers based on changing usage patterns. For long-term archival needs, use S3 Glacier Deep Archive to minimize energy consumption for rarely accessed data.
6. **Monitor and optimize storage utilization**. Regularly review and clean up unnecessary data and snapshots. Use Amazon S3 Storage Lens to gain visibility into usage patterns and identify optimization opportunities across your organization.
7. **Centralize and share datasets**. Implement a centralized data catalog using AWSAWS Glue Data Catalog to make datasets discoverable and reusable, reducing the need for multiple copies of the same data.
8. **For generative AI workloads, use AI for intelligent data management**. Implement embeddings storage with efficient vector databases like Amazon OpenSearch Service to optimize storage of large language model context.

Resources

Related documents:

- Understanding and managing Amazon S3 storage classes
- Managing storage costs with Amazon S3 Intelligent-Tiering
- Amazon S3 Storage Analytics and Insights
- AWS Well-Architected Framework: Performance Efficiency Pillar - Data Management

Model development

Best practices

- MLSUS04-BP01 Define sustainable performance criteria
- MLSUS04-BP02 Select energy-efficient algorithms
- MLSUS04-BP03 Archive or delete unnecessary training artifacts
- MLSUS04-BP04 Use efficient model tuning methods

MLSUS04-BP01 Define sustainable performance criteria

Creating machine learning models that balance accuracy with environmental impact is essential for sustainable AI development. When we focus exclusively on model accuracy, we overlook the economic, environmental, and social costs of achieving marginal performance improvements. Since the relationship between model accuracy and complexity is logarithmic at best, continuing to train a model longer or searching extensively for better hyperparameters often yields only minimal gains while significantly increasing resource consumption.

Desired outcome: You establish balanced performance criteria for your ML models that satisfy your business requirements without excessive resource usage. You implement mechanisms to optimize training efficiency, reducing carbon footprint and costs while maintaining appropriate model performance. Your machine learning lifecycle incorporates sustainability as a core consideration alongside traditional metrics like accuracy.

Common anti-patterns:

- Pursuing maximum model accuracy without considering environmental impact.
- Using oversized models when smaller models would be sufficient.
- Allowing training jobs to run indefinitely with minimal performance improvements.

- Ignoring resource utilization metrics during model development.

Benefits of establishing this best practice:

- Reduced energy consumption and carbon footprint from ML operations.
- Lower computational costs for model training and deployment.
- Faster development cycles with earlier training completion.
- Better alignment between business requirements and model capabilities.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

When developing machine learning models, balancing accuracy with sustainability requires deliberate consideration of how much performance is actually needed. The incremental gains in accuracy often diminish significantly beyond certain thresholds, while computational costs continue to rise. By defining sustainable performance criteria upfront, you create guardrails that reduce unnecessary environmental impact.

These criteria should reflect your specific business needs rather than abstract notions of best possible performance. For many applications, a model that is 95% accurate may provide the same business value as one that is 96% accurate but requires twice the computational resources to train. Understand this relationship to make informed trade-offs.

Early stopping mechanisms represent one practical implementation of sustainable criteria. These techniques automatically terminate training when improvement plateaus, avoiding wasted computation. Tools like SageMaker AI Debugger and Automatic Model Tuning provide built-in capabilities to implement early stopping without compromising model quality.

Regularly measuring and monitoring resource utilization can identify opportunities for optimization. Tracking metrics like CPU and GPU utilization, memory consumption, and training duration provides visibility into your model development's environmental impact and can identify inefficiencies.

Implementation steps

1. **Establish sustainable performance criteria**. Define concrete performance thresholds that meet your business requirements without excessive resource consumption. Consider both the absolute performance levels needed and the point at which additional gains become negligible. Include

both accuracy and efficiency metrics in your criteria, such as inference latency, model size, and training resource requirements.

2. **Analyze the accuracy-resource tradeoff**. Conduct experiments to understand the relationship between model performance and resource consumption for your specific use case. Plot accuracy against training time, model size, or computational resources to identify the point of diminishing returns. Use this data to set reasonable stopping criteria for your training jobs.
3. **Configure early stopping in training jobs**. Set up SageMaker AI Automatic Model Tuning with early stopping to terminate training jobs that are not showing significant improvement. Navigate to the SageMaker AI console, create a hyperparameter tuning job, and enable the early stopping option. Alternatively, configure early stopping programmatically using the SageMaker AI Python SDK by setting the early_stopping_type parameter to **Auto**.
4. **Implement debugging rules**. Use SageMaker AI Debugger to automatically stop training when specific conditions are met. Add rules like LossNotDecreasing to your training script to detect when your model stops improving. For example, configure the rule to stop training if the loss doesn't decrease by at least 0.01% over the last ten epochs.
5. **Monitor resource utilization**. Track the efficiency of your training jobs using CloudWatch metrics or SageMaker AI Debugger's Profiling Report. Monitor metrics such as CPUUtilization, GPUUtilization, GPUMemoryUtilization, MemoryUtilization, and DiskUtilization. Identify patterns of resource underutilization that might indicate opportunities for right-sizing your infrastructure.
6. **Right-size your training infrastructure**. Based on utilization metrics, adjust the instance types and counts for your training jobs. Select the most efficient instance type that meets your performance requirements rather than defaulting to the most powerful option. For distributed training jobs, verify that you're using an appropriate number of instances to maximize utilization.
7. **Validate against business requirements**. Before finalizing your model, verify that it meets the business requirements while adhering to your sustainable performance criteria. Document the tradeoffs made and the rationale behind them to provide transparency to stakeholders.
8. **Use no-code ML for rapid prototyping**. Use SageMaker AI Canvas with natural language support for data exploration and model development to quickly validate ML approaches before investing in resource-intensive custom development. Canvas can generate models with minimal computational overhead for initial feasibility testing. Export Canvas-generated models and code to notebooks for further optimization and sustainable development practices.
9. **Use AI-powered code generation for sustainable development**. Use AI-powered development tools like Amazon Q Developer and Kiro to generate efficient ML code, automate performance

optimization scripts, and accelerate the implementation of sustainable ML practices while reducing development resource consumption.

10. **Consider smaller specialized models**. For generative AI applications, evaluate whether smaller, domain-specific models can meet your needs instead of using large general-purpose models. Techniques like retrieval-augmented generation (RAG) can enhance smaller models' capabilities while maintaining lower resource requirements. Fine-tuning a smaller base model on your specific data often provides better sustainability outcomes than using larger generic models.

Resources

Related documents:

- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- Model Registration Deployment with Model Registry
- Stop Training Jobs Early
- Model Explainability
- Amazon SageMaker AI Debugger - Built-in Rules: LossNotDecreasing
- SageMaker AI job metrics

MLSUS04-BP02 Select energy-efficient algorithms

Choosing energy-efficient algorithms minimizes resource usage while maintaining performance, reducing your machine learning workloads' environmental impact and operational costs.

Desired outcome: You establish a systematic approach for selecting and optimizing algorithms that deliver the necessary performance while minimizing computational resources. Your ML workloads run more efficiently, reducing energy consumption, carbon footprint, and infrastructure costs without significant performance degradation.

Common anti-patterns:

- Defaulting to the most complex algorithm without evaluating simpler alternatives.
- Ignoring model compression techniques that could reduce resource requirements.
- Overlooking the environmental impact of computational resources.
- Focusing solely on model accuracy without considering resource efficiency.

Benefits of establishing this best practice:

- Reduced energy consumption and carbon footprint.
- Faster inference times and improved user experience.
- Ability to deploy ML models on resource-constrained devices.
- Extended battery life for edge devices running ML workloads.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Energy-efficient algorithm selection requires balancing model performance with resource consumption. When developing machine learning models, computational efficiency directly impacts sustainability and cost. Starting with simpler algorithms provides a baseline for comparison and often delivers sufficient results without excessive resource demands. Modern approaches like model distillation, pruning, and quantization enable you to achieve near-equivalent results using significantly fewer resources.

The environmental impact of ML workloads increases with model complexity, making optimization techniques essential for sustainable AI development. By systematically evaluating algorithm efficiency alongside performance metrics, you can make informed decisions that reduce your carbon footprint while maintaining service quality.

Implementation steps

1. **Begin with a simple algorithm to establish a baseline:** Start your development process with straightforward algorithms that provide a reference point for performance and resource usage. Then test different algorithms with increasing complexity to observe whether performance improvements justify additional resource consumption. Measure both model accuracy and resource utilization metrics to make informed decisions about complexity trade-offs.
2. **Explore simplified versions of popular algorithms:** Research and implement distilled or optimized versions of standard algorithms that deliver similar performance with reduced computational requirements. For example, DistilBERT, a distilled version of BERT, has 40% fewer parameters, runs 60% faster, and preserves 97% of its performance. Similar approaches exist for many common model architectures.
3. **Implement model compression techniques:** Apply pruning to remove model weights that contribute minimally to predictions, reducing model size and computational requirements. Use quantization to represent numerical values with lower precision, significantly decreasing memory usage and processing demands while maintaining acceptable accuracy levels.

4. **Leverage AWS optimization services:** Deploy Amazon SageMaker AI Neo to automatically optimize your ML models for inference on cloud resources and edge devices. SageMaker AI Neo analyzes your model and generates optimized code that maximizes performance while minimizing resource consumption, allowing you to deploy more efficient models across diverse deployment targets.
5. **Monitor and optimize resource utilization:** Track the resources provisioned for your training and inference jobs (InstanceCount, InstanceType, and VolumeSizeInGB) and their efficient use (CPUUtilization, GPUUtilization, GPUMemoryUtilization, MemoryUtilization, and DiskUtilization) through the SageMaker AI Console, CloudWatch Console or your SageMaker AI Debugger Profiling Report. Use these insights to right-size your resources and identify optimization opportunities.
6. **Consider hardware-specific optimizations:** Choose appropriate instance types for training and inference based on your model's characteristics. Some algorithms perform better on GPU instances, while others may be more efficient on CPU or specialized accelerators like AWS Inferentia. Matching your algorithm to the optimal hardware can significantly improve energy efficiency.
7. **Use optimized foundation model containers:** Deploy models using SageMaker AI's optimized foundation model containers that include pre-configured environments with built-in quantization and optimization techniques. These containers support frameworks like Hugging Face Transformers and provide automatic performance optimizations.
8. **Use AI-powered code generation for algorithm optimization**. Use AI-powered development tools like Amazon Q Developer and Kiro to generate optimized algorithm implementations, automate model compression code, and accelerate the development of energy-efficient ML solutions.
9. **Apply efficient architectures for foundation models:** When working with generative AI models, consider parameter-efficient fine-tuning approaches like LoRA (Low-Rank Adaptation) or P-tuning instead of full fine-tuning. These techniques can reduce the computational resources required while achieving comparable performance. Leverage pre-trained foundation models available through SageMaker AI JumpStart to avoid the energy-intensive process of training from scratch.

Resources

Related documents:

- SageMaker AI JumpStart pretrained models

- Multi-model endpoints
- Model performance optimization with SageMaker AI Neo
- The AWS Inferentia Chip With DLAMI
- Prepare Model for Compilation
- Optimize AI/ML workloads for sustainability: Part 2, model development

Related videos:

- Deploy an ML model for best performance, cost, and prediction quality
- SageMaker AI HyperPod: Revolutionizing Foundation Model Training with Resilience and Performance

MLSUS04-BP03 Archive or delete unnecessary training artifacts

Remove training artifacts that are unused and no longer required to limit wasted resources. Determine when you can archive training artifacts to more energy-efficient storage or safely delete them.

Desired outcome: You reduce your environmental footprint by removing unnecessary storage of ML training artifacts. Your organization maintains only essential training data and models, efficiently archives what might be needed later, and systematically removes what is no longer required. This approach not only conserves resources but also simplifies management of your machine learning assets.

Common anti-patterns:

- Keeping training artifacts indefinitely.
- Ignoring the accumulation of unused logs, models, and experiment data.
- Not implementing systematic cleanup processes for completed experiments.

Benefits of establishing this best practice:

- Reduced storage costs and resource consumption.
- Improved organization and discoverability of important ML artifacts.
- Enhanced security through reduced data surface area.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Machine learning workflows generate substantial volumes of artifacts during the development process. These include experiment data, logs, model checkpoints, and various intermediary outputs. While some of these artifacts are essential for long-term use, many become unnecessary after model deployment or project completion.

Consider the lifecycle of your ML artifacts. Some might need preservation for compliance-aligned or reproducibility purposes, while others can be safely removed once they've served their immediate purpose. For artifacts that must be retained but are rarely accessed, consider using tiered storage options that balance accessibility with resource efficiency.

Implementation steps

1. **Organize your ML experiments with management tools**. Use SageMaker AI Experiments to track, compare, and organize your machine learning experiments in a structured manner. This organization makes it more straightforward to identify which artifacts are essential and which can be archived or deleted.
2. **Implement regular cleanup procedures**. Follow the clean up training resources guidance from AWS to systematically remove SageMaker AI resources you no longer need. Create automated cleanup workflows that run on a schedule to avoid the accumulation of unused artifacts.
3. **Set appropriate log retention policies**. By default, Amazon CloudWatch retains logs indefinitely, which consumes unnecessary resources. Implement limited retention time for your notebooks and training logs to automatically expire and delete logs after they're no longer needed.
4. **Establish storage lifecycle policies**. Configure Amazon S3 lifecycle policies to automatically transition training artifacts to more cost-effective and energy-efficient storage classes like Amazon Glacier or S3 Glacier Deep Archive based on access patterns.
5. **Monitor storage utilization**. Use Amazon S3 Storage Lens to gain visibility into your storage usage patterns and identify opportunities for optimization. Track metrics regularly to verify that your cleanup procedures are effective.
6. **Implement container image cleanup**. Use Amazon ECR lifecycle policies to automatically clean up unused container images that may have been created during training jobs. This avoids the accumulation of outdated or unused container images.
7. **Establish artifact tagging standards**. Create a consistent tagging strategy for ML artifacts to identify their purpose, associated projects, and expiration dates. This makes it simpler to determine what can be archived or deleted during cleanup processes.

8. **Leverage managed MLflow for experiment tracking**. Use managed MLflow on SageMaker AI to create, manage, analyze, and compare your machine learning experiments with better organization and tracking capabilities, making it more straightforward to identify which experiments and associated artifacts can be safely archived or deleted.
9. **For GenAI foundation models, implement token usage monitoring and cleanup**. For generative AI projects, monitor token usage and intermediate outputs, which can quickly accumulate. Implement automated cleanup of temporary prompts, completions, and generated content that isn't required for the final model.

Resources

Related documents:

- Clean up Amazon SageMaker AI notebook instance resources
- Cataloging and analyzing your data with S3 Inventory
- What Is Amazon EventBridge?
- Working with log groups and log streams
- Automate the cleanup of images by using lifecycle policies in Amazon ECR
- AWS Systems Manager for Automation
- What Is AWS Config?
- Optimizing MLOps for Sustainability

MLSUS04-BP04 Use efficient model tuning methods

Optimize your machine learning model's hyperparameters with resource-efficient strategies that minimize computational needs while improving performance. Efficient tuning methods can significantly reduce costs, energy consumption, and time-to-market compared to resource-intensive brute force approaches.

Desired outcome: You can systematically find optimal hyperparameters for your machine learning models while minimizing computational resource consumption. By implementing intelligent search strategies and following best practices for optimization, you achieve better model performance with fewer resources, reduce your environmental footprint, and accelerate time-to-market for your ML solutions.

Common anti-patterns:

- Using grid search, which tests the most possible combinations of hyperparameters.

- Running many concurrent training jobs without considering how results from previous jobs can inform later ones.
- Specifying excessively broad hyperparameter ranges without proper understanding of their impact.
- Using random search when more efficient options like Bayesian or Hyperband are available.

Benefits of establishing this best practice:

- Reduce computational resources required for hyperparameter tuning by up to 10x compared to random search.
- Decrease energy consumption and associated carbon emissions from ML training.
- Find optimal hyperparameters faster, accelerating time-to-market.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Hyperparameter tuning is a crucial step in machine learning model development that directly impacts both model performance and resource utilization. Efficient tuning strategies can dramatically reduce the computational resources required while finding optimal or near-optimal parameter settings.

When approaching hyperparameter tuning, consider the relationship between tuning strategy and resource consumption. Grid search, which exhaustively tests the most possible combinations, is the most resource-intensive approach and should generally be avoided. Random search provides better resource efficiency but lacks intelligence in selecting which configurations to test. More sophisticated approaches like Bayesian optimization and Hyperband can find optimal hyperparameters with significantly fewer training jobs, reducing both time and resource usage.

The efficiency of your hyperparameter tuning is also affected by how you configure concurrent jobs and specify parameter ranges. Running too many concurrent jobs can waste resources when using methods that benefit from sequential exploration. Similarly, specifying unnecessarily wide parameter ranges increases the search space complexity without adding value.

Implementation steps

1. **Choose an efficient search strategy**. Implement Bayesian optimization or Hyperband search strategies instead of random or grid search. Bayesian search uses information from previous trials to make intelligent guesses about promising hyperparameter configurations, typically

requiring 10 times fewer jobs than random search to find optimal parameters. For large models like deep neural networks addressing computer vision problems, Amazon SageMaker AI Hyperband can find optimal hyperparameters up to three times faster than Bayesian search.

2. **Optimize job concurrency settings**. Configure your hyperparameter tuning jobs with appropriate concurrency settings. With Bayesian optimization, running fewer concurrent jobs often yields better results since the algorithm benefits from information gathered in previous iterations. Balance the tradeoff between parallelism (which speeds up overall completion time) and sequential learning (which improves optimization efficiency) based on your specific requirements for time-to-completion versus resource efficiency.
3. **Define thoughtful parameter ranges**. Carefully select which hyperparameters to tune and their corresponding value ranges. Focus on parameters that significantly impact model performance and limit ranges to reasonable values based on domain knowledge or prior experiments. For parameters known to be log-scaled (learning rates, regularization strengths), convert them to log space to improve optimization efficiency. This focused approach reduces the search space complexity, discovering optimal configurations with fewer resources.
4. **Leverage early stopping capabilities**. Implement mechanisms to terminate underperforming training jobs early to avoid wasting resources on unpromising hyperparameter configurations. Use Amazon SageMaker AI's built-in early stopping functionality that automatically terminates training jobs that are unlikely to produce better models than previously completed jobs.
5. **Monitor resource utilization**. Track metrics related to resource provisioning and utilization for your hyperparameter tuning jobs. Use Amazon SageMaker AI's integration with Amazon CloudWatch to monitor CPU utilization, GPU utilization, memory usage, and other resource metrics. Analyze these metrics to identify optimization opportunities in your tuning strategy.
6. **Integrate with your MLOps pipeline**. Incorporate efficient hyperparameter tuning as a standard component of your MLOps workflow. Automate the process of selecting optimal hyperparameters and retraining models when necessary. This verifies the consistent application of efficient tuning practices across your machine learning projects.
7. **Leverage pre-trained models to reduce tuning scope**. Start with pre-trained models from the expanded SageMaker AI JumpStart catalog or Amazon Bedrock foundation models, which often require minimal hyperparameter tuning for your use case, significantly reducing computational resources compared to training from scratch.
8. **Leverage AI-powered code generation for tuning automation**. Use AI-powered development tools like Amazon Q Developer and Kiro to generate efficient hyperparameter tuning scripts, automate optimization workflows, and accelerate the implementation of resource-efficient tuning strategies.

9. **Consider warm-starting tuning jobs**. Use the results from previous hyperparameter tuning jobs to initialize new jobs when incrementally improving models or adapting to changing data patterns. This approach reduces the resources required to find good hyperparameters compared to starting from scratch.

Resources

Related documents:

- Best Practices for Hyperparameter Tuning
- Automatic model tuning with SageMaker AI
- Managed Spot Training in Amazon SageMaker AI
- Amazon SageMaker AI Training Compiler
- Choosing the Best Number of Concurrent Training Jobs
- Choosing Hyperparameter Ranges
- Amazon SageMaker AI Hyperband Search Strategy

Deployment

Best practices

- MLSUS05-BP01 Align SLAs with sustainability goals
- MLSUS05-BP02 Use efficient silicon
- MLSUS05-BP03 Optimize models for inference
- MLSUS05-BP04 Deploy multiple models behind a single endpoint

MLSUS05-BP01 Align SLAs with sustainability goals

Define service level agreements (SLAs) that support your sustainability goals while meeting your business requirements. Define SLAs to meet your business requirements, not exceed them. Make trade-offs that significantly reduce environmental impacts in exchange for acceptable decreases in service levels.

Desired outcome: You establish SLAs that balance business requirements with sustainability objectives, optimizing resource utilization while maintaining acceptable service levels. By implementing appropriate inference methods based on latency tolerance, availability needs, and

response time requirements, you can reduce idle resources, minimize energy consumption, and lower your machine learning workload's environmental impact.

Common anti-patterns:

- Maintaining always-on inference endpoints for workloads with sporadic or batch processing needs.
- Setting unnecessarily stringent response time requirements when users can tolerate some latency.
- Configuring excessive redundancy beyond what's needed for business continuity.

Benefits of establishing this best practice:

- Reduced infrastructure costs through optimized resource utilization.
- Lower carbon footprint from minimized idle computing resources.
- Alignment of technical operations with organizational sustainability goals.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

When designing machine learning systems, your SLA choices directly impact resource consumption and environmental sustainability. By carefully analyzing your actual business requirements rather than automatically opting for maximum performance, you can identify opportunities to make sustainable trade-offs without compromising essential functionality.

Consider your application's true latency requirements, availability needs, and processing patterns. For example, if your users can tolerate a response time of seconds rather than milliseconds, asynchronous or batch processing approaches can dramatically reduce resource usage compared to always-on real-time endpoints. Similarly, if your application can gracefully handle occasional unavailability during instance failures, you can avoid overprovisioning redundant capacity.

The goal is to make conscious trade-offs that balance sustainability with business needs, focusing on what's truly required rather than defaulting to full time maximum performance.

Implementation steps

1. **Queue incoming requests and process them asynchronously**. If your users can tolerate some latency, deploy your model on serverless or asynchronous endpoints to reduce resources that

are idle between tasks and minimize the impact of load spikes. These options will automatically scale the instance or endpoint count to zero when there are no requests to process, so you only maintain an inference infrastructure when your endpoint is processing requests.

2. **Adjust availability**. If your users can tolerate some latency in the rare case of a failover, don't provision extra capacity. If an outage occurs or an instance fails, Amazon SageMaker AI automatically attempts to distribute your instances across Availability Zones. Adjusting availability is an example of a conscious trade off you can make to meet your sustainability targets.
3. **Adjust response time**. When you don't need real-time inference, use SageMaker AI Batch Transform. Unlike a persistent endpoint, clusters are decommissioned when batch transform jobs finish so you don't continuously maintain an inference infrastructure.
4. **Conduct workload analysis**. Assess your machine learning workload's usage patterns and latency requirements to determine the most sustainable deployment option. Identify periods of peak activity versus low or no usage to determine if on-demand scaling is appropriate for your needs.
5. **Define sustainability metrics**. Establish key metrics to track your sustainability improvements, such as compute hours saved, idle time reduced, or overall carbon footprint reduction. Include these metrics alongside traditional performance indicators in your operational dashboards.
6. **Leverage enhanced serverless inference capabilities**. Use improved SageMaker AI Serverless Inference with increased memory configurations and better cold-start performance for variable workloads that don't require always-on infrastructure.
7. **Optimize large language model deployments with serverless deployment or batch processing**. For generative AI workloads using large language models (LLMs), consider serverless model inference through SageMaker AI or implement Bedrock batch processing for non-interactive generation tasks like content summarization or document analysis to reduce resource consumption.

Resources

Related documents:

- Asynchronous inference
- Batch transform for inference with Amazon SageMaker AI
- Deploy models with Amazon SageMaker AI Serverless Inference
- Multi-model endpoints

- Best practices for deploying models on SageMaker AI Hosting Services
- Amazon SageMaker AI Inference Recommender
- Sustainability as a non-functional requirement
- **Related services:**
- Amazon Bedrock

MLSUS05-BP02 Use efficient silicon

Choosing the right compute architecture for your machine learning workloads can significantly reduce energy consumption and carbon footprint while maintaining high performance.

Desired outcome: You select and deploy the most energy-efficient instance types for your machine learning workloads, resulting in reduced power consumption, lower costs, and a more sustainable ML infrastructure without compromising performance or functionality.

Common anti-patterns:

- Using general-purpose instances for specialized ML workloads.
- Selecting hardware based primarily on performance without considering power efficiency.
- Not optimizing ML models to work efficiently on specialized hardware.

Benefits of establishing this best practice:

- Reduced energy consumption by up to 60% with purpose-built ML accelerators.
- Decreased carbon footprint of your ML operations.
- Improved performance-per-watt metrics for your ML infrastructure.
- Better alignment with organizational sustainability goals.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

The energy efficiency of your ML infrastructure directly impacts both your operating costs and environmental footprint. By selecting purpose-built hardware accelerators designed specifically for ML workloads, you can achieve significant sustainability improvements while maintaining or even improving performance.

AWS has developed several specialized compute architectures optimized for different ML workload types, from training to inference. Each is designed to deliver maximum performance per watt to assist in meeting sustainability goals while effectively running your ML applications. These purpose-built solutions are particularly important for large-scale ML deployments where small efficiency improvements can result in substantial energy savings when scaled across your infrastructure.

When choosing compute resources for your ML workloads, consider not only the raw performance but also the energy efficiency of the hardware. The most powerful instance sometimes isn't the most sustainable choice, as matching the hardware capabilities to your specific workload requirements can often lead to better sustainability outcomes.

Implementation steps

1. **Assess your ML workload requirements**. Before selecting compute resources, analyze your ML workload characteristics including model size, batch processing capabilities, latency requirements, and throughput needs. This assessment can determine which specialized hardware will provide the optimal balance between performance and sustainability.
2. **Use AWS Graviton3 for CPU-based ML inference**. AWS Graviton3 processors offer the best performance per watt in Amazon EC2, using up to 60% less energy than comparable instances. They deliver up to three times better performance compared to Graviton2 processors for ML workloads and support bfloat16, making them ideal for efficient CPU-based inference.
3. **Deploy AWS Inferentia for deep learning inference**. Amazon EC2 Inf2 instances offer up to 50% better performance per watt over comparable Amazon EC2 instances. These instances are purpose-built to run deep learning models at scale and assist in meeting sustainability goals when deploying ultra-large models.
4. **Leverage AWS Trainium for ML training**. Amazon EC2 Trn2 instances based on custom-designed AWS Trainium chips offer up to 50% cost-to-train savings over comparable instances. When using a Trainium-based instance cluster, total energy consumption for training BERT Large from scratch is approximately 25% lower compared to same-sized clusters of comparable accelerated EC2 instances.
5. **Optimize your models for the target hardware**. Use the AWS Neuron SDK to compile and optimize your ML models specifically for AWS Inferentia and Trainium chips. This verifies that your models can take full advantage of the hardware's power-efficient design and specialized ML acceleration features.
6. **Monitor and measure power efficiency**. Use Amazon CloudWatch metrics to track the resource utilization of your ML workloads. Compare performance-per-watt metrics across

different instance types to validate your efficiency improvements and identify areas for further optimization.

7. **Leverage purpose-built training infrastructure**. For large-scale model training, use SageMaker AI HyperPod which provides purpose-built infrastructure for distributed training with automatic checkpoint storage and recovery, optimizing resource utilization for long-running training jobs.
8. **Evaluate serverless options for intermittent workloads**. For ML inference workloads with variable traffic patterns, consider Amazon SageMaker AI Serverless Inference to automatically scale compute resources based on traffic, reducing idle resource waste.

Resources

Related documents:

- Amazon EC2 instance types
- Specifications for Amazon EC2 accelerated computing instances
- AWS Neuron SDK Documentation
- What is Amazon SageMaker AI?

Related examples:

- AWS Graviton Technical Guide

MLSUS05-BP03 Optimize models for inference

Optimize machine learning models for inference to achieve higher performance with lower computational resources, reducing both costs and environmental impact.

Desired outcome: You achieve more efficient machine learning inference with optimized models that use less computational resources, consume less energy, and deliver faster predictions. This optimization can reduce operational costs and carbon footprint while improving the user experience through faster response times.

Common anti-patterns:

- Deploying models directly from training without optimization.
- Using generic frameworks for inference when optimized alternatives exist.
- Selecting oversized models when smaller ones would suffice for the task.

- Ignoring hardware-specific optimizations for deployment targets.

Benefits of establishing this best practice:

- Reduced inference costs through more efficient resource utilization.
- Improved response times for better user experience.
- Extended battery life for edge device deployments.
- Ability to deploy complex models on resource-constrained devices.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Model optimization for inference represents a critical step in the machine learning lifecycle that is often overlooked. While data scientists typically focus on model accuracy during development, the computational efficiency of these models during deployment significantly impacts costs, energy consumption, and user experience.

Model compilation transforms your trained models into optimized forms that can run more efficiently on specific hardware. This process analyzes your model's computational graph, applies various optimizations like operator fusion and memory layout transformations, and generates optimized code that takes advantage of hardware-specific capabilities. The result is a model that delivers the same predictions but requires less computational resources and energy.

The optimization approach varies based on your model type and deployment target. For tree-based models like XGBoost, specialized compilers can significantly reduce inference latency. For deep learning models, frameworks can avoid training-specific operations and optimize the execution path. For edge deployments, additional optimizations like quantization can reduce model size while maintaining acceptable accuracy.

Implementation steps

1. **Select appropriate model architectures**. Choose model architectures that naturally lend themselves to efficient inference. Consider simpler architectures or distilled versions of larger models when possible. Balance accuracy requirements against efficiency needs for your specific use case.
2. **Use open-source model compilers**. Use specialized tools like Treelite for decision tree ensembles such as XGBoost, LightGBM, and RandomForest. These compilers transform models

into optimized C code that improves prediction throughput through more efficient memory access patterns and computational optimizations.

3. **Leverage Amazon SageMaker AI Neo**. Use Amazon SageMaker AI Neo to optimize models for inference on Amazon SageMaker AI in the cloud and supported edge devices. Neo automatically optimizes models trained in TensorFlow, PyTorch, MXNet, and other frameworks, delivering up to 25 times the performance improvement while maintaining accuracy. The Neo runtime consumes only a fraction of the resources required by full deep learning frameworks.
4. **Consider quantization techniques**. Apply post-training quantization to reduce model precision from 32-bit floating point to 16-bit or 8-bit integers where appropriate. This reduces model size and improves computational efficiency, particularly on hardware with specialized integer arithmetic capabilities.
5. **Optimize for specific hardware targets**. Configure your model compilation process to target the specific hardware where inference will run. Different optimizations apply to CPUs, GPUs, and specialized accelerators like AWS Inferentia or AWS Trainium.
6. **Use efficient model serving architectures**. Implement SageMaker AI multi-model endpoints and inference pipelines to build modular inference architectures that efficiently share resources across models and processing stages, allowing for improved resource utilization and optimization.
7. **Leverage AI-powered code generation for optimization automation**. Use AI-powered development tools like Amazon Q Developer and Kiro to generate model optimization code, automate inference pipeline creation, and accelerate the implementation of efficient deployment strategies.
8. **Test performance under realistic conditions**. Measure inference latency, throughput, and resource utilization under realistic workloads before deploying to production. Compare optimized models against baselines to quantify improvements and verify that optimization doesn't impact accuracy.

Resources

Related documents:

- Model performance optimization with SageMaker AI Neo
- SageMaker AI JumpStart pretrained models
- Multi-model endpoints
- Edge Devices

MLSUS05-BP04 Deploy multiple models behind a single endpoint

Host multiple models behind a single endpoint to improve endpoint utilization. Sharing endpoint resources is more sustainable and less expensive than deploying a single model behind one endpoint.

Desired outcome: Your organization achieves greater efficiency in your model deployments by consolidating multiple models on shared infrastructure. This can reduce costs by increasing utilization of your endpoint resources, minimize environmental impact through reduced carbon emissions, and simplify your model deployment architecture.

Common anti-patterns:

- Deploying each model on its own dedicated endpoint regardless of utilization patterns.
- Over-provisioning resources for endpoints that serve infrequently accessed models.
- Creating separate infrastructure for similar models that could share resources.

Benefits of establishing this best practice:

- Reduced costs through better utilization of compute resources.
- Decreased carbon footprint by up to 90% compared to single-model deployments.
- Improved scalability for serving multiple models.
- Enhanced operational efficiency for inference workloads.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Model deployment architecture significantly impacts both cost and sustainability. By hosting multiple models behind a single endpoint, you can substantially improve resource utilization, reducing both expenses and environmental impact. Amazon SageMaker AI provides several approaches to implement this practice, each suitable for different scenarios depending on your model types, access patterns, and processing requirements.

Consider your workload characteristics when selecting a deployment approach. For a large collection of similar models that aren't accessed simultaneously, multi-model endpoints (MME) offer the most efficient solution. When you need to deploy different model types with varying framework requirements, multi-container endpoints (MCE) provide flexibility. For sequential

processing workflows, inference pipelines allow you to chain preprocessing, prediction, and postprocessing steps.

Implementation steps

1. **Assess your model deployment needs**. Evaluate your current deployment architecture, focusing on model similarity, access patterns, and resource requirements. Identify opportunities to consolidate models based on these characteristics.
2. **Select the appropriate deployment method**. Choose from one of SageMaker AI's three approaches based on your workload requirements:
   - Multi-model endpoints for similar models with varied access patterns
   - Multi-container endpoints for heterogeneous models requiring different frameworks
   - Inference pipelines for sequential processing workflows
3. **Implement multi-model endpoints**. Use SageMaker AI's multi-model endpoint capability to host multiple models within a single container. This approach is ideal when you have many similar models that use the same framework and don't need to be accessed simultaneously. Configure the endpoint to dynamically load and unload models based on usage patterns to optimize memory utilization.
4. **Deploy multi-container endpoints**. When your models require different containers or frameworks, use SageMaker AI multi-container endpoints to host up to 15 containers on a single endpoint. Configure each container with its specific model and framework requirements while sharing the underlying infrastructure resources.
5. **Create inference pipelines**. For workflows that require sequential processing, implement a SageMaker AI inference pipeline to chain multiple containers. Define the sequence to handle preprocessing, model inference, and postprocessing steps as a unified flow, passing outputs from one container as inputs to the next.
6. **Monitor and optimize resource utilization**. Use Amazon CloudWatch to track endpoint metrics including CPU utilization, memory usage, and invocation patterns. Analyze this data to further optimize your deployment by adjusting instance types or scaling configurations based on actual usage.
7. **Implement cost tracking**. Set up cost allocation tags to monitor the efficiency gains from your consolidated endpoint deployment. Compare the costs before and after implementation to quantify savings and justify the architectural approach.
8. **Leverage modular deployment architectures**. Use SageMaker AI multi-container endpoints and inference pipelines to create modular inference architectures that can efficiently share resources across different model components and processing stages.

9. **Consider sustainability metrics**. Track carbon emission reductions resulting from your optimized deployment architecture. Using AWS Customer Carbon Footprint Tool, you can measure the environmental impact of your workloads and report on sustainability improvements.

Resources

Related documents:

- Multi-model endpoints
- Multi-container endpoints
- Inference pipelines in Amazon SageMaker AI
- Deploy models with Amazon SageMaker AI Serverless Inference
- Automatic scaling of Amazon SageMaker AI models
- Asynchronous inference

Monitoring

Best practices

- MLSUS06-BP01 Measure material efficiency
- MLSUS06-BP02 Retrain only when necessary

MLSUS06-BP01 Measure material efficiency

Measure efficiency of your workload in provisioned resources per unit of work to determine not only the business success of the workload, but also its material efficiency. Use this measure as a baseline for your sustainability improvement process.

Desired outcome: You can quantify and track the resources required by your machine learning workload to deliver its business outcomes. By measuring resources per unit of work, you create a sustainable baseline that allows you to track improvements over time, make data-driven decisions about resource optimization, and demonstrate the environmental impact of your sustainability efforts.

Common anti-patterns:

- Focusing exclusively on business metrics without considering resource consumption.

- Measuring total resource usage without normalizing by business outcomes.
- Making optimization decisions without quantitative data on resource efficiency.

Benefits of establishing this best practice:

- Creates a clear way to measure sustainability progress over time.
- Enables comparison of different implementations based on material efficiency.
- Provides data to demonstrate ROI on sustainability improvements.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Material efficiency is a critical aspect of sustainable machine learning workloads. By measuring the resources provisioned per unit of work, you gain visibility into how efficiently your workload uses cloud resources to deliver business value. This approach normalizes your sustainability metrics across different workload sizes, usage patterns, and business outcomes.

When implementing material efficiency measurements, you should first determine what constitutes a *unit of work* for your specific ML workload. This could be a model training run, a prediction request, a processed transaction, or another relevant business outcome. Then, establish which resources to track. Typically, this is compute (vCPU minutes), storage (GB), and network (GB transferred). By dividing your provisioned resources by these units of work, you create a normalized efficiency metric that can be tracked over time.

For example, a recommendation engine might track vCPU minutes per recommendation delivered, or a fraud detection system could measure GB of storage per fraudulent transaction identified. Tracking these metrics can determine if changes to your architecture, algorithms, or deployment strategies are improving efficiency or creating waste.

Implementation steps

1. **Define your unit of work**. Identify what business outcomes your ML workload produces, such as model training completions, predictions made, insights generated, or transactions processed. Verify that this metric directly relates to business value delivered.
2. **Establish resource metrics**. Track key resource consumption metrics using Amazon CloudWatch. For ML workloads, important metrics include compute utilization (vCPU minutes), memory

usage, storage consumption (GB), and network transfer (GB). AWS Cost Explorer can identify key cost drivers in your ML workload.

3. **Calculate baseline efficiency**. Divide your resource consumption metrics by your units of work to create efficiency ratios (for example, vCPU minutes per prediction, GB storage per model training run, or network transfer per transaction). Document these values as your baseline for future comparisons.
4. **Set improvement targets**. Based on your baseline measurements, set realistic targets for reducing resource consumption per unit of work. Consider both absolute reductions (total resources) and percentage improvements over the baseline.
5. **Implement monitoring and reporting**. Use Amazon CloudWatch dashboards to visualize your efficiency metrics over time. Set up alerts for significant deviations from expected efficiency. Amazon SageMaker AI provides built-in monitoring capabilities for ML workloads to track resource utilization.
6. **Quantify improvement benefits**. When implementing changes, calculate both the immediate resource savings and the projected long-term benefits. Include the return on investment from your improvement activities to demonstrate value to stakeholders.
7. **Review and optimize regularly**. Schedule regular reviews of your efficiency metrics to identify new optimization opportunities. As your workload evolves, your baseline and targets may need adjustment.

Resources

Related documents:

- Analyzing your costs and usage with AWS Cost Explorer
- Data and model quality monitoring with Amazon SageMaker AI Model Monitor
- AWS Trusted Advisor
- Measure results and replicate successes
- AWS Well-Architected Framework - Sustainability Pillar
- Cloud Financial Management with AWS

MLSUS06-BP02 Retrain only when necessary

Because of model drift, robustness requirements, or new ground truth data being available, models usually need to be retrained. Instead of retraining arbitrarily, monitor your ML model in production,

automate your model drift detection and only retrain when your model's predictive performance has fallen below defined KPIs.

Desired outcome: You will establish a data-driven approach to model retraining that optimizes computational resources while maintaining model performance. By implementing automated monitoring and drift detection systems, you can identify when your model's performance degrades below acceptable thresholds and retrain only when necessary. This reduces unnecessary computational overhead while verifying that your models remain accurate and relevant.

Common anti-patterns:

- Retraining models on a fixed schedule regardless of performance.
- Manual monitoring of model performance leading to delayed detection of drift.
- Retraining without clear performance thresholds or KPIs.

Benefits of establishing this best practice:

- Reduced computational resources and carbon footprint.
- Lower operational costs for model maintenance.
- More efficient use of data science team resources.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Machine learning models deployed in production environments naturally experience degradation in performance over time due to changes in data patterns, user behaviors, or business environments. This phenomenon, known as model drift, requires retraining to maintain optimal performance. However, retraining a model consumes significant computational resources, which impacts both operational costs and environmental sustainability.

By implementing automated monitoring systems and establishing clear performance thresholds, you can make data-driven decisions about when to retrain your models. This approach verifies that you're using computational resources efficiently while maintaining the effectiveness of your ML systems. Through continuous monitoring, you can detect both concept drift (changes in the relationship between input and output variables) and data drift (changes in the distribution of input data).

Your monitoring strategy should incorporate both technical metrics (such as accuracy, precision, recall) and business-relevant KPIs that directly tie to organizational outcomes. By correlating these metrics with specific thresholds for retraining, you create a sustainable approach to model maintenance that optimizes both performance and resource utilization.

Implementation steps

1. **Determine key performance indicators**. Work with business stakeholders to identify minimum acceptable accuracy levels and maximum acceptable error rates for your models. These KPIs should directly connect to business outcomes and provide clear thresholds for when retraining becomes necessary. Consider both technical metrics (precision, recall, F1 score) and business metrics (conversion rates, user engagement, revenue impact) when establishing these thresholds.
2. **Monitor your model deployed in Production**. Implement Amazon SageMaker AI Model Monitor to continuously evaluate your deployed models. SageMaker AI Model Monitor provides capabilities for data quality monitoring, model quality monitoring, bias drift monitoring, and feature attribution drift monitoring. Configure alerts based on your established KPIs to automatically notify your team when performance begins to degrade.
3. **Set up baseline metrics**. Create a baseline of your model's performance metrics immediately after deployment. This serves as a reference point against which future performance can be measured. SageMaker AI Model Monitor can automatically generate these baselines from your training data or initial inference data.
4. **Configure drift detection thresholds**. Define specific threshold values that indicate when drift has occurred to a degree that warrants retraining. These thresholds should be based on your KPIs and statistical measures of data or concept drift. Configure Amazon CloudWatch alarms to go off when these thresholds are exceeded.
5. **Automate your retraining pipelines**. Use Amazon SageMaker AI Pipelines, AWS Step Functions for Amazon SageMaker AI, or third-party tools to create automated workflows that can be initiated when drift is detected. These pipelines should handle data preparation, model training, evaluation, and deployment with minimal manual intervention.
6. **Optimize retraining frequency**. Based on historical drift patterns, adjust monitoring sensitivity and retraining thresholds to optimize the frequency of retraining. Finding the right balance keeps models performant while minimizing computational overhead.
7. **Establish canary deployments**. When deploying retrained models, use SageMaker AI deployment options like canary deployments to gradually shift traffic to the new model while monitoring performance, allowing for quick rollback if issues arise.

8. **Leverage enhanced bias and drift detection**. Use improved SageMaker AI Clarify capabilities with enhanced bias detection, new fairness metrics, and better visualization tools to more accurately detect when retraining is necessary.
9. **Implement feedback loops for generative models**. Establish mechanisms to collect user feedback and engagement metrics to detect when outputs become less relevant or helpful. Use Amazon SageMaker AI JumpStart for fine-tuning foundation models when drift is detected in generative applications.

Resources

Related documents:

- Amazon SageMaker AI Model Monitor documentation
- Amazon SageMaker AI Pipelines documentation
- Amazon SageMaker AI Clarify
- Amazon SageMaker AI Feature Store
- AWS Step Functions for Data Science
- SageMaker AI Deployment Guardrails
- SageMaker AI Governance
- Optimizing MLOps for Sustainability

Related videos:

- SageMaker AI HyperPod: Revolutionizing Foundation Model Training with Resilience and Performance

References

- AWS Architecture Center
- Architecture Best Practices for Machine Learning
- AWS Well-Architected Framework
- AWS Operational Excellence pillar
- AWS Security pillar
- AWS Reliability pillar
- AWS Performance Efficiency pillar
- AWS Cost Optimization pillar
- AWS Sustainability pillar
- Amazon AI Fairness and Explainability Whitepaper
- Overview of Deployment Options on AWS
- Crisp-DM 1.0
- Announcing New Tools for Building with Generative AI on AWS $1 Amazon Web Services
- Get started with generative AI on AWS using Amazon SageMaker AI JumpStart $1 Amazon Web Services

Best practices by ML lifecycle

There are six phases in the machine learning lifecycle. The following index lists the best practices by lifecycle phase.

Business goal identification

Operational excellence

- the section called “MLOPS01-BP01 Develop the right skills with accountability and empowerment”
- the section called “MLOPS01-BP02 Discuss and agree on the level of model explainability”
- the section called “MLOPS01-BP03 Monitor model adherence to business requirements”

Security

- the section called “MLSEC01-BP01 Validate ML data permissions, privacy, software, and license terms”

Reliability

There are no reliability pillar best practices for business goal identification.

Performance efficiency

- the section called “MLPERF01-BP01 Determine key performance indicators”

Cost optimization

- the section called “MLCOST01-BP01 Define overall return on investment (ROI) and opportunity cost”
- the section called “MLCOST01-BP02 Use managed services to reduce total cost of ownership (TCO)”

Sustainability

- the section called “MLSUS01-BP01 Define the overall environmental impact or benefit”

ML problem framing

Operational excellence

- the section called “MLOPS02-BP01 Establish ML roles and responsibilities”
- the section called “MLOPS02-BP02 Prepare an ML profile template”
- the section called “MLOPS02-BP03 Establish model improvement strategies”
- the section called “MLOPS02-BP04 Establish a lineage tracker system”
- the section called “MLOPS02-BP05 Establish feedback loops across ML lifecycle phases”
- the section called “MLOPS02-BP06 Review fairness and explainability”

Security

- the section called “MLSEC02-BP01 Design data encryption and obfuscation”

Reliability

- the section called “MLREL01-BP01 Use APIs to abstract change from model consuming applications”
- the section called “MLREL01-BP02 Adopt a machine learning microservice strategy”

Performance efficiency

- the section called “MLPERF02-BP01 Define relevant evaluation metrics”
- the section called “MLPERF02-BP02 Use purpose-built AI and ML services and resources”

Cost optimization

- the section called “MLCOST02-BP01 Identify if machine learning is the right solution”
- the section called “MLCOST02-BP02 Perform a tradeoff analysis between custom and pre-trained models”

Sustainability

- the section called “MLSUS02-BP01 Consider AI services and pre-trained models”
- the section called “MLSUS02-BP02 Select sustainable Regions”

Data processing

Operational excellence

- the section called “MLOPS03-BP01 Profile data to improve quality”
- the section called “MLOPS03-BP02 Create tracking and version control mechanisms”

Security

- the section called “MLSEC03-BP01 Provide least privilege access”
- the section called “MLSEC03-BP02 Secure data and modeling environment”
- the section called “MLSEC03-BP03 Protect sensitive data privacy”
- the section called “MLSEC03-BP04 Enforce data lineage”
- the section called “MLSEC03-BP05 Keep only relevant data”

Reliability

- the section called “MLREL02-BP01 Use a data catalog”
- the section called “MLREL02-BP02 Use a data pipeline”
- the section called “MLREL02-BP03 Automate managing data changes”

Performance efficiency

- the section called “MLPERF03-BP01 Use a modern data architecture”

Cost optimization

- the section called “MLCOST03-BP01 Use managed data labeling”
- the section called “MLCOST03-BP02 Use no-code or low-code and code generation tools for interactive analysis”

- the section called “MLCOST03-BP03 Use managed data processing capabilities”
- the section called “MLCOST03-BP04 Enable feature reusability”

Sustainability

- the section called “MLSUS03-BP01 Minimize idle resources”
- the section called “MLSUS03-BP02 Implement data lifecycle policies aligned with your sustainability goals”
- the section called “MLSUS03-BP03 Adopt sustainable storage options”

Model development

Operational excellence

- the section called “MLOPS04-BP01 Automate operations through MLOps and CI/CD”
- the section called “MLOPS04-BP02 Establish reliable packaging patterns to access approved public libraries”

Security

- the section called “MLSEC04-BP01 Secure governed ML environment”
- the section called “MLSEC04-BP02 Secure inter-node cluster communications”
- the section called “MLSEC04-BP03 Protect against data poisoning threats”

Reliability

- the section called “MLREL03-BP01 Enable CI/CD/CT automation with traceability”
- the section called “MLREL03-BP02 Verify feature consistency across training and inference”
- the section called “MLREL03-BP03 Validate models with relevant data”
- the section called “MLREL03-BP04 Establish data bias detection and mitigation”

Performance efficiency

- the section called “MLPERF04-BP01 Optimize training and inference instance types”
- the section called “MLPERF04-BP02 Explore alternatives for performance improvement”

- the section called “MLPERF04-BP03 Establish a model performance evaluation pipeline”
- the section called “MLPERF04-BP04 Establish feature statistics”
- the section called “MLPERF04-BP05 Perform a performance trade-off analysis”
- the section called “MLPERF04-BP06 Detect performance issues when using transfer learning”

Cost optimization

- the section called “MLCOST04-BP01 Select optimal computing instance size”
- the section called “MLCOST04-BP02 Use managed build environments”
- the section called “MLCOST04-BP03 Select local training for small scale experiments”
- the section called “MLCOST04-BP04 Select an optimal ML framework”
- the section called “MLCOST04-BP05 Use automated machine learning”
- the section called “MLCOST04-BP06 Use managed training capabilities”
- the section called “MLCOST04-BP07 Use distributed training”
- the section called “MLCOST04-BP08 Stop resources when not in use”
- the section called “MLCOST04-BP09 Start training with small datasets”
- the section called “MLCOST04-BP10 Use warm start and checkpointing hyperparameter tuning”
- the section called “MLCOST04-BP11 Use hyperparameter optimization technologies”
- the section called “MLCOST04-BP12 Set up a budget and use resource tagging to track costs”
- the section called “MLCOST04-BP13 Enable data and compute proximity”
- the section called “MLCOST04-BP14 Select optimal algorithms”

Sustainability

- the section called “MLSUS04-BP01 Define sustainable performance criteria”
- the section called “MLSUS04-BP02 Select energy-efficient algorithms”
- the section called “MLSUS04-BP03 Archive or delete unnecessary training artifacts”
- the section called “MLSUS04-BP04 Use efficient model tuning methods”

Model deployment

Operational excellence

- the section called “MLOPS05-BP01 Establish deployment environment metrics”

Security

- the section called “MLSEC05-BP01 Protect against adversarial and malicious activities”

Reliability

- the section called “MLREL04-BP01 Automate endpoint changes through a pipeline”
- the section called “MLREL04-BP02 Use an appropriate deployment and testing strategy”

Performance efficiency

- the section called “MLPERF05-BP01 Evaluate cloud versus edge options for machine learning deployment”
- the section called “MLPERF05-BP02 Choose an optimal deployment option in the cloud”

Cost optimization

- the section called “MLCOST05-BP01 Use an appropriate deployment option”
- the section called “MLCOST05-BP02 Explore cost effective hardware options”
- the section called “MLCOST05-BP03 Right-size the model hosting instance fleet”

Sustainability

- the section called “MLSUS05-BP01 Align SLAs with sustainability goals”
- the section called “MLSUS05-BP02 Use efficient silicon”
- the section called “MLSUS05-BP03 Optimize models for inference”
- the section called “MLSUS05-BP04 Deploy multiple models behind a single endpoint”

Model monitoring

Operational excellence

- the section called “MLOPS06-BP01 Synchronize architecture and configuration, and check for skew across environments”
- the section called “MLOPS06-BP02 Enable model observability and tracking”

Security

- the section called “MLSEC06-BP01 Restrict access to intended legitimate consumers”
- the section called “MLSEC06-BP02 Monitor human interactions with data for anomalous activity”

Reliability

- the section called “MLREL05-BP01 Allow automatic scaling of the model endpoint”
- the section called “MLREL05-BP02 Create a recoverable endpoint with a managed version control strategy”

Performance efficiency

- the section called “MLPERF06-BP01 Include human-in-the-loop monitoring”
- the section called “MLPERF06-BP02 Evaluate model explainability”
- the section called “MLPERF06-BP03 Evaluate data drift”
- the section called “MLPERF06-BP04 Monitor, detect, and handle model performance degradation”
- the section called “MLPERF06-BP05 Establish an automated re-training framework”
- the section called “MLPERF06-BP06 Review for updated data and features for retraining”

Cost optimization

- the section called “MLCOST06-BP01 Monitor usage and cost by ML activity”
- the section called “MLCOST06-BP02 Monitor return on investment for ML models”
- the section called “MLCOST06-BP03 Monitor endpoint usage and right-size the instance fleet”
- the section called “MLCOST06-BP04 Enable debugging and logging”

Sustainability

- the section called “MLSUS06-BP01 Measure material efficiency”
- the section called “MLSUS06-BP02 Retrain only when necessary”

Conclusion

The Well-Architected ML design principles in this paper provide the guidance for the best practices collection. The technology and cloud agnostic best practices across the Well-Architected pillars provide architectural guidance for each phase of the ML lifecycle. Implementation plans provide guidance on implementing these best practices on AWS.

Architecture diagrams demonstrate the lifecycle phases with the supporting technologies, that enable many of the best practices introduced in this paper. The ML lens extends the Well-Architected Framework, and builds specific machine learning best practices upon it. As you work towards building and deploying production ML workloads in AWS, we recommend reviewing the AWS Well-Architected Framework pillar best practices.

Use the lens to build ML workloads with operational excellence, security, reliability, performance efficiency, cost optimization, and sustainability in mind. Plan early and make informed decisions when designing new workloads. Use the best practices to guide you through building and deploying new workloads faster. Using the lens guidance, evaluate existing workloads regularly to identify, mitigate, and address potential issues early.

Contributors

The following individuals and organizations contributed to this document:

- Steven DeVries, Principal Solutions Architect, Amazon Web Services
- Gopi Krishnamurthy, Senior Solutions Architect, Amazon Web Services
- Haleh Najafzadeh, Principal Guidance Strategist, Amazon Web Services
- Venkat Devarajan, Senior Solutions Architect, Amazon Web Services
- Adam Weber, Senior Solutions Architect, Automotive, Amazon Web Services
- Benoit de Chateauvieux, Startup Solutions Architect, Amazon Web Services
- Dan Ferguson, Senior Worldwide Specialist SA, GenAI, Amazon Web Services
- Michael Hsieh, Principal AI/ML Specialist SA, WWSO BDSI, Amazon Web Services
- Ganapathi Krishnamoorthi, Principal Startup Solutions Architect, AI/ML, Amazon Web Services
- Neil Mackin, Principal ML Strategist, AWS Customer Engineering, Amazon Web Services
- Raj Pathak, Senior Solutions Architect, WWCS Geo Solutions Architecture, Amazon Web Services
- Ram Pathangi, Solutions Architect, WWCS Geo Solutions Architecture, Amazon Web Services
- Raju Patil, Data Scientist, AWS WWCO Professional Services, Amazon Web Services
- Eddie Pick, Manager, Startup Solutions Architect, Amazon Web Services
- Deepali Rajale, Specialist Technical Account Manager, AI/ML (Sp), AWS Enterprise Support, Amazon Web Services
- Brendan Sisson, Principal Solutions Architect, Amazon Web Services
- Dhiraj Thakur, Senior Solutions Architect, Amazon Web Services
- Pallavi Nargund, Principal Solutions Architect, Amazon Web Services
- Patrick Roberts, Senior Data Scientist, AWS WWCO Professional Services, Amazon Web Services
- Raju Penmatcha, Senior Solutions Architect, AI Platforms, Amazon Web Services
- Jess Clark, Principal Security Engineer, Identity and Access Management, Amazon Web Services
- Emily Soward, Data Scientist, AWS WWCO Professional Services, Amazon Web Services
- Amit Lulla, Senior Solutions Architect ISV, AWS WWCS Geo Solutions Architect, Amazon Web Services
- Giuseppe Angelo Porcelli, Principal ML Solutions Architect, AI Platforms, Amazon Web Services

- Shelbee Eigenbrode, Principal AI/ML Specialist Solutions Architect, AWS WWSO AI/ML, Amazon Web Services
- Phi Nguyen, Front End Developer, AWS WWCO Professional Services, Amazon Web Services
- Jeremy Sobek, Technical Curriculum Developer, AWS T&C Curriculum, Amazon Web Services
- Bruno Klein, Machine Learning Engineer, AWS WWCO Professional Services, Amazon Web Services
- Chris Boomhower, Machine Learning Engineer, AWS WWCO Professional Services, Amazon Web Services
- Matthew Wygant, Sr. TPM Guidance, Amazon Web Services
- Mohammad Arbabshirani, Sr. Data Scientist Manager, AWS WWCO Professional Services, Amazon Web Services
- Stewart Matzek, Sr. Technical Writer, Amazon Web Services

Document revisions

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
New lens version	Updated all best practices across the entire lens with new structure and guidance. Updated formatting to be consistent with other Well-Architected lenses.	November 19, 2025
Major update	Whitepaper updated to reflect current best practices , implementation guidance, and AWS services. Added the sustainability pillar that describes reducing your impact on the environme nt while taking advantage of AWS advanced AI/ML capabilities for your business. Updated technical reference s and resources including product documentation, blog posts, instructional and video links, and solution briefs.	July 5, 2023
Minor update	Fixed broken links.	April 6, 2023
Minor update	Corrected typo.	July 21, 2022
Major update	Whitepaper updated to reflect current best practices and AWS services.	October 12, 2021

Minor changes	Updated links in the whitepaper.	January 21, 2021
Initial publication	Machine Learning Lens first published.	April 16, 2020

Note

To subscribe to RSS updates, you must have an RSS plug-in enabled for the browser that you are using.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.