



Amazon Nova 2 开发人员指南

# Amazon Nova



# Amazon Nova: Amazon Nova 2 开发人员指南

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

# Table of Contents

|                                   |    |
|-----------------------------------|----|
| 什么是 Amazon Nova 2 ? .....         | 1  |
| 重要概念 .....                        | 1  |
| Amazon Nova 模型 .....              | 1  |
| 您可以构建什么? .....                    | 2  |
| 优势 .....                          | 2  |
| Amazon Nova 工作原理 .....            | 3  |
| 定价 .....                          | 3  |
| 后续步骤 .....                        | 3  |
| Amazon Nova 2 中的新增功能 .....        | 4  |
| 新模型 .....                         | 4  |
| 全新特性与功能 .....                     | 4  |
| Nova Multimodal Embeddings .....  | 4  |
| 扩展思考与推理 .....                     | 4  |
| 内置工具 .....                        | 4  |
| 构建人工智能代理 .....                    | 5  |
| 改进文件理解 .....                      | 5  |
| 增强视频理解 .....                      | 5  |
| 模型自定义 .....                       | 5  |
| Amazon Nova Forge .....           | 5  |
| 后续步骤 .....                        | 5  |
| 开始使用 .....                        | 6  |
| 开始在控制台中进行操作 .....                 | 6  |
| 探索文本演示平台 .....                    | 6  |
| API 入门 .....                      | 7  |
| 获取凭证来授予编程访问权限 .....               | 8  |
| 将 Amazon Bedrock 权限附加到用户或角色 ..... | 9  |
| 使用 Amazon Nova 模型生成对文本提示的回复 ..... | 10 |
| 核心推理 .....                        | 11 |
| 重要提示：超时配置 .....                   | 12 |
| 核心推理主题 .....                      | 12 |
| 使用 Converse API .....             | 13 |
| 请求结构 .....                        | 13 |
| 使用系统提示词 .....                     | 14 |
| 推理参数 .....                        | 14 |

|   |    |
|---|----|
| 使用推理 .....                                    | 15 |
| 使用 Invoke API .....                           | 18 |
| 请求结构 .....                                    | 18 |
| 请求参数 .....                                    | 19 |
| 流式响应 .....                                    | 19 |
| 使用 ConverseStream 进行流式传输 .....                | 19 |
| 使用 InvokeModelWithResponseStream 进行流式响应 ..... | 20 |
| 流事件类型 .....                                   | 20 |
| 使用 Amazon Nova Embeddings .....               | 21 |
| 主要特征 .....                                    | 21 |
| 生成嵌入 .....                                    | 22 |
| 按需型推理 .....                                   | 25 |
| 优势 .....                                      | 25 |
| 使用按需推理 .....                                  | 26 |
| 定价 .....                                      | 26 |
| 限额和限制 .....                                   | 26 |
| 使用 Nova 的功能 .....                             | 27 |
| 推理 .....                                      | 27 |
| 多模态理解 .....                                   | 28 |
| 按模态划分的支持内容类型 .....                            | 28 |
| 图像理解 .....                                    | 29 |
| 视频理解 .....                                    | 31 |
| 文档理解 .....                                    | 33 |
| 语音转语音 .....                                   | 35 |
| 主要特征 .....                                    | 35 |
| 开始使用 .....                                    | 36 |
| 说明导入和配置 .....                                 | 36 |
| 定义 SimpleNovaSonic 类 .....                    | 37 |
| 初始化客户端 .....                                  | 37 |
| 处理事件 .....                                    | 38 |
| 启动会话 .....                                    | 38 |
| 处理音频输入 .....                                  | 41 |
| 结束会话 .....                                    | 42 |
| 处理回复 .....                                    | 43 |
| 播放音频 .....                                    | 44 |
| 捕获音频 .....                                    | 45 |

|                       |     |
|-----------------------|-----|
| 运行主函数 .....           | 46  |
| 代码示例 .....            | 48  |
| 入门示例 .....            | 48  |
| 高级应用场景 .....          | 48  |
| 实践讲习会 .....           | 49  |
| 语音对话提示 .....          | 49  |
| 重要指南 .....            | 49  |
| 推荐的语音基线系统提示 .....     | 50  |
| 语音提示配置 .....          | 50  |
| 系统提示配置 .....          | 50  |
| 性别语言的性一致 .....        | 51  |
| 语音思维链：规范性推理 .....     | 52  |
| 过度使用建议短语 .....        | 53  |
| 核心概念 .....            | 55  |
| 事件生命周期 .....          | 55  |
| 事件流顺序 .....           | 57  |
| 使用双向 API 处理输入事件 ..... | 57  |
| 使用双向 API 处理输出事件 ..... | 66  |
| 插话 .....              | 72  |
| 轮换可控性 .....           | 74  |
| 跨模态输入 .....           | 75  |
| 语言支持和多语言功能 .....      | 78  |
| 管理聊天记录 .....          | 79  |
| 工具配置 .....            | 85  |
| 异步工具调用 .....          | 92  |
| 集成 .....              | 95  |
| Strands Agents .....  | 95  |
| 框架集成 .....            | 98  |
| 使用 Nova 构建高级系统 .....  | 106 |
| 概述 .....              | 106 |
| 扩展思考 .....            | 106 |
| 扩展思考工作原理 .....        | 106 |
| 配置选项 .....            | 109 |
| 支持的模型 .....           | 110 |
| 结合工具使用的扩展思考 .....     | 110 |
| 了解推理词元和定价 .....       | 110 |

|                           |     |
|---------------------------|-----|
| 常见问题 .....                | 110 |
| 工具使用 .....                | 111 |
| 了解工具使用工作流程 .....          | 111 |
| 创建工具 .....                | 112 |
| 调用工具 .....                | 114 |
| 内置系统工具 .....              | 118 |
| 构建人工智能代理 .....            | 120 |
| 创建代理 .....                | 120 |
| 调用代理 .....                | 122 |
| Web Grounding .....       | 125 |
| Web Grounding 的工作原理 ..... | 125 |
| 如何使用 Web Grounding .....  | 125 |
| 区域可用性 .....               | 126 |
| 响应结构 .....                | 126 |
| Grounding 安全 .....        | 127 |
| 错误处理 .....                | 128 |
| 内置工具所需的权限 .....           | 128 |
| 提示工程 .....                | 130 |
| 什么是提示工程 .....             | 130 |
| 开始使用提示工程 .....            | 130 |
| 最佳实践 .....                | 131 |
| 了解角色 .....                | 131 |
| 创建精准提示 .....              | 131 |
| 聚焦提示的各个部分 .....           | 150 |
| 使用系统角色 .....              | 151 |
| 提供示例 ( 少样本提示 ) .....      | 159 |
| 工具调用系统 .....              | 169 |
| 进阶提示技巧 .....              | 170 |
| 多模态输入提示 .....             | 190 |
| 多模态通用准则 .....             | 190 |
| 理解文档与图像 .....             | 191 |
| 视频理解 .....                | 31  |
| 代码和故障排除 .....             | 202 |
| 代码库 .....                 | 202 |
| Converse API 示例 .....     | 202 |
| InvokeModel API 示例 .....  | 222 |

|                             |     |
|-----------------------------|-----|
| 问题排查 .....                  | 227 |
| 身份验证和设置 .....               | 227 |
| 访问模型被拒 .....                | 228 |
| 区域可用性问题 .....               | 228 |
| 超时配置 .....                  | 228 |
| API 响应问题 .....              | 229 |
| 推理模式错误 .....                | 230 |
| 工具使用问题 .....                | 231 |
| API 和 SDK 参考 .....          | 235 |
| 请求和响应架构 .....               | 235 |
| 完整请求结构 .....                | 235 |
| 完整响应结构 .....                | 240 |
| SDK 参考 .....                | 243 |
| Amazon Nova 模型自定义 .....     | 244 |
| 自定义方法 .....                 | 244 |
| 自定义平台 .....                 | 244 |
| 一般先决条 .....                 | 245 |
| Amazon Nova 配方 .....        | 246 |
| 获取基本配方 .....                | 247 |
| 获取 Forge 配方 .....           | 248 |
| 可用的模型和算法 .....              | 248 |
| Amazon Nova 2.0 Lite .....  | 248 |
| 基于 SageMaker 训练作业 .....     | 254 |
| Amazon Nova 自定义最佳实践 .....   | 254 |
| Nova Forge SDK .....        | 257 |
| 训练 .....                    | 264 |
| 监控迭代进度 .....                | 297 |
| 评估 .....                    | 302 |
| 迭代训练 .....                  | 360 |
| 基于 SageMaker HyperPod ..... | 360 |
| Amazon Nova 2 模型的算力需求 ..... | 361 |
| Amazon Nova 自定义最佳实践 .....   | 254 |
| Nova Forge SDK .....        | 365 |
| HP 集群设置 .....               | 366 |
| 核心命令指南 .....                | 366 |
| Nova Forge 访问与设置 .....      | 376 |

|                              |     |
|------------------------------|-----|
| 训练 .....                     | 377 |
| 评估 .....                     | 413 |
| MLflow 监控 .....              | 456 |
| 迭代训练 .....                   | 459 |
| 概述 .....                     | 459 |
| 工作原理 .....                   | 460 |
| 建议使用迭代训练的场景 .....            | 461 |
| 示例工作流：SFT → RFT .....        | 462 |
| 监控迭代进度 .....                 | 464 |
| 最佳实践 .....                   | 466 |
| 成本注意事项 .....                 | 467 |
| 限制 .....                     | 468 |
| 问题排查 .....                   | 469 |
| SageMaker 推理 .....           | 470 |
| 功能 .....                     | 471 |
| 支持的模型与实例 .....               | 472 |
| 支持的 AWS 区域 .....             | 474 |
| 支持的区域与容器映像 .....             | 474 |
| 最佳实践 .....                   | 474 |
| 支持 .....                     | 475 |
| 开始使用 .....                   | 475 |
| API 参考 .....                 | 495 |
| 评估模型 .....                   | 504 |
| Amazon Nova Forge 滥用检测 ..... | 511 |
| Amazon Bedrock 推理 .....      | 512 |
| 部署自定义模型进行按需型推理 .....         | 512 |
| 按需型推理 .....                  | 517 |
| 限制 .....                     | 519 |
| Nova Forge .....             | 520 |
| 先决条件 .....                   | 520 |
| 订阅 Nova Forge .....          | 520 |
| 其他先决条件 .....                 | 521 |
| Nova Forge 访问与设置 .....       | 521 |
| 获取 Nova Forge 文档 .....       | 521 |
| 步骤 1：订阅 Nova Forge .....     | 521 |
| 步骤 2：设置 HyperPod 基础设施 .....  | 527 |

|                              |     |
|------------------------------|-----|
| 内容审核设置 .....                 | 528 |
| 持续预训练与中期训练 .....             | 528 |
| 什么是中间检查点？为什么需要它们？ .....      | 528 |
| 有哪些可用的检查点？ .....             | 529 |
| 应该使用哪个检查点？ .....             | 530 |
| 如何为 Nova 2.0 使用数据混合？ .....   | 530 |
| 剖析数据混合类别 .....               | 531 |
| 监督式微调 .....                  | 535 |
| 介绍 .....                     | 535 |
| 何时使用 SFT？ .....              | 535 |
| 何时使用参数高效微调，何时使用全秩 SFT？ ..... | 536 |
| 适用于 SFT 的数据混合 .....          | 536 |
| YAML 配置示例 .....              | 537 |
| 模型检查点 .....                  | 539 |
| 训练方法 .....                   | 540 |
| 开始之前的先决条件 .....              | 540 |
| 成功实现 SFT 的系统性方法 .....        | 541 |
| 实验排序和数据混合 .....              | 541 |
| 为 SFT 准备数据集 .....            | 542 |
| 评估方法 .....                   | 542 |
| 最佳实践 .....                   | 254 |
| 准备训练作业配置 .....               | 544 |
| 设置数据混合块 .....                | 546 |
| 如何启动作业 .....                 | 550 |
| 如何监控作业 .....                 | 558 |
| SFT 后如何评估模型？ .....           | 561 |
| 经验教训和提示 .....                | 563 |
| 强化学习 .....                   | 564 |
| 使用自带编排工具进行代理多轮评测 .....       | 564 |
| 负责任的人工智能工具包 .....            | 565 |
| 负责任的人工智能工具包 .....            | 565 |
| 负责任地使用 .....                 | 567 |
| 指南要求 .....                   | 567 |
| 建议 .....                     | 568 |
| 配额 .....                     | 569 |
| 请求增加 Amazon Nova 配额 .....    | 569 |

---

|                                 |     |
|---------------------------------|-----|
| 监控 .....                        | 570 |
| 安全性 .....                       | 571 |
| 使用 CloudFormation 创建资源 .....    | 572 |
| 了解有关 CloudFormation 的更多信息 ..... | 572 |
| 文档历史记录 .....                    | 573 |

# 什么是 Amazon Nova 2 ?

Amazon Nova 提供多模态基础模型，可处理文本、图像、视频、文档和语音。Amazon Nova 2 模型支持多达 100 万个词元的上下文及高级推理能力，可助力您构建能够理解复杂输入并生成准确响应的高阶人工智能应用程序。

您可以构建交互式聊天机器人、分析文档和视频、创建具有扩展推理能力的人工智能代理，以及开发支持语音的应用程序。

## 重要概念

在了解 Amazon Nova 模型之前，请先熟悉以下核心概念：

### 基础模型

通过 API 提供的预训练人工智能系统，提供不同规模和功能。

### 推理

向模型发送请求并接收生成响应的过程。

### 推理

一种扩展思考能力，使模型能够拆解复杂问题，并在给出答案前展示分步分析过程。

### 多模态

在单次请求中，可同时处理并理解多种类型输入（文本、图像、视频和文档）的能力。

### RAG（检索增强生成）

将模型响应与自有数据源相结合，以提供更精准、贴合上下文答案的技术方法。

## Amazon Nova 模型

Amazon Nova 2 包含以下模型，每个模型均针对不同使用案例进行了优化：

| 模型          | 输入模式        | 输出模态 | 使用案例               |
|-------------|-------------|------|--------------------|
| Nova 2 Lite | 文本、图像、视频、文档 | 文本   | 注重速度与成本效益的高吞吐量应用程序 |

| 模型                         | 输入模式           | 输出模态  | 使用案例            |
|----------------------------|----------------|-------|-----------------|
| Nova 2 Sonic               | 语音、文本          | 语音、文本 | 需快速响应的语音交互应用程序  |
| Nova Multimodal Embeddings | 文本、图像、文档、视频、音频 | 嵌入    | 语义搜索、推荐系统及相似度匹配 |

所有模型均支持多达 100 万个词元的上下文，并且可在单次响应中生成最高 65,536 个词元。具备推理能力的模型可以进行扩展思考，逐步解决复杂问题。

## 您可以构建什么？

以下是可以使用 Amazon Nova 构建的应用程序示例：

- 智能文档助手：利用多达 100 万个词元的上下文处理大型文档，实现问答与信息提炼（结合 RAG）
- 复杂推理应用程序：通过扩展思考能力展示模型分步分析过程，解决多步骤问题（或结合推理功能）
- 视频分析流程：批量提取视频内容信息、生成摘要并定位关键片段（Nova 2 Lite）
- 语音人工智能代理：构建可理解语音输入并以自然语言回复的对话式代理（Nova 2 Sonic）

## 优势

Amazon Nova 具备以下优势：

### 多模态理解

在单次请求中处理文本、图像、视频、文档和语音。Amazon Nova 模型能够理解不同类型输入之间的关联信息。

### 扩展上下文

支持多达 100 万个词元，可完整处理代码库、长文档或持续对话，且不会丢失上下文。

### 高级推理

具备推理能力的模型可拆解复杂问题并展示分步分析过程，提升多步骤任务的准确性。

## 灵活部署

通过 Amazon Bedrock 调用模型，无需管理基础设施；也可通过微调与强化学习对模型进行自定义。

## 内置工具

借助 Web Grounding 能力获取实时信息，使用代码解释器执行 Python 代码，无需集成外部服务。

# Amazon Nova 工作原理

Amazon Nova 模型为可通过 Amazon Bedrock 调用的基础模型。基本工作流程如下：

1. 应用程序向 Amazon Bedrock 发送包含输入和配置参数的请求。
2. Amazon Nova 模型对输入内容进行处理，若已配置推理功能则会启用该能力。
3. 模型生成响应并将其返回给应用程序。

您可以通过使用 RAG 整合数据、启用内置工具或通过微调自定义模型来增强响应。

## 定价

Amazon Nova 定价基于所处理的输入与输出词元。不同模型对应不同定价套餐：

- Nova 2 Lite：针对高吞吐量、高性价比处理场景优化
- Nova 2 Sonic：为语音应用程序提供均衡定价

有关当前定价信息，请参阅 [Amazon Bedrock 定价](#)。

## 后续步骤

- 要了解 Amazon Nova 2 中的新功能，请参阅 [Amazon Nova 2 中的新增功能](#)。
- 要开始使用 Amazon Nova，请参阅[开始使用 Amazon Nova 2](#)。
- 要了解核心推理能力，请参阅[核心推理](#)。
- 要为使用案例自定义模型，请参阅[在 SageMaker AI 上自定义 Amazon Nova 模型](#)。

# Amazon Nova 2 中的新增功能

Amazon Nova 2 在语义理解、内容创作及语音能力方面实现了大幅升级。以下章节将介绍主要新增功能与改进项。

## 新模型

### Nova 2 Lite

**增强型模型**：具备高级推理能力，支持扩展思考，提供三级强度调控，并针对生产级智能代理工作流程优化了多模态理解能力。

### Nova 2 Sonic

**升级式对话语音模型**：优化了七国语言的语音理解、自然语言处理与语音合成能力。

### Nova Multimodal Embeddings

**多模态嵌入模型**：支持文本、图像、文档、视频及音频输入，并生成对应嵌入向量。

## 全新特性与功能

### Nova Multimodal Embeddings

Nova Multimodal Embeddings 通过单一模型支持文本、文档、图像、视频和音频，可实现跨模态检索应用程序。Nova Multimodal Embeddings 将各类内容映射到统一语义空间，支持执行单模态、跨模态及多模态向量运算，为代理检索增强生成 ( RAG )、多模态语义搜索等应用提供支撑。

### 扩展思考与推理

Nova 2 Lite 支持扩展思考，可让模型在生成回答前投入更多时间对复杂问题进行推理。该能力可提升多步推理任务的准确性，例如涉及多工具调用的代理工作流程、高等数学运算、复杂规划及代码生成等场景。

要了解更多信息，请参阅 [Amazon Nova 2 中的扩展思考](#)。

### 内置工具

Amazon Nova 2 内置多款工具，无需外部集成即可扩展模型能力：

- Web Grounding：获取网络实时信息，提供最新应答并减少幻觉输出。
- 代码解释器：执行 Python 代码完成运算。

## 构建人工智能代理

Amazon Nova 2 模型针对构建人工智能代理进行了优化。这些模型改进了工具的使用，增强了多步骤任务的推理能力，提升了在复杂代理工作流程中保留上下文的能力。

### 改进文件理解

Nova 2 Lite 提供了增强的文档处理功能，可更好地理解复杂的文档布局、表格、图表和多页文档。此类模型可以更准确地从 PDF、电子表格和其他文档格式中提取信息。

### 增强视频理解

Nova 2 Lite 改进了视频分析功能，包括更好的视觉感知、时间理解、动作识别，以及能够以更高的准确性处理更长的视频序列。

### 模型自定义

Nova 2 Lite 支持在 Amazon Bedrock 和 SageMaker AI 上进行监督式微调 (SFT) 与强化微调 (RFT)，可让您根据自身特定业务需求定制 Amazon Nova 2。

## Amazon Nova Forge

Amazon Nova Forge 是一项首创服务，让组织能够以最简单、最具成本效益的方式使用 Amazon Nova 构建自己的前沿模型。

## 后续步骤

- 若要了解 Amazon Nova 模型和功能，请参阅[什么是 Amazon Nova 2？](#)。
- 若要开始使用 Amazon Nova 2.0，请参阅[开始使用 Amazon Nova 2](#)。
- 若要探索核心推理功能，请参阅[核心推理](#)。

# 开始使用 Amazon Nova 2

您可以通过以下方法调用 Amazon Nova 2 模型：

- Amazon Bedrock [InvokeModel](#) 或 [Converse](#) API 操作
- 适用于 Nova 2 Lite、Nova 2 Sonic 的 Amazon Bedrock [聊天和文本演示平台](#)

## Important

如果是 AWS 或 Amazon Bedrock 的新手用户，则必须先按照 [Amazon Bedrock 入门](#) 中的步骤设置 Amazon Bedrock。此外，您还可以通过 [nova.amazon.com](https://nova.amazon.com) 直接试用 Amazon Nova 模型（无需进入 Amazon Bedrock），或访问 [nova.amazon.com/dev](https://nova.amazon.com/dev) 免费开始构建。

选择一个主题，开始通过该方法使用 Amazon Nova。

## 开始在 Amazon Bedrock 控制台中使用 Amazon Nova

本节将介绍如何在 AWS 管理控制台中使用演示平台向 Amazon Nova 2 模型提交文本提示，并生成文本或图像响应。在运行以下示例之前，应检查您是否满足了以下先决条件：

### 先决条件

- 您拥有 AWS 账户并有权使用该账户中具有 Amazon Bedrock 必要权限的角色。否则，请按照 [Amazon Bedrock 入门](#) 中的步骤操作。
- 如果从美国区域访问模型，则必须使用美国 CRIS 端点，这需要在模型 ID 中添加 us 前缀（例如 `us.amazon.nova-2-lite-v1:0`）。
- 如果从美国以外的区域访问模型，则可以使用全局 CRIS 端点（例如 `global.amazon.nova-2-lite-v1:0`），或在模型 ID 中为区域添加前缀（例如 `us/eu/jp`）。

## 探索文本演示平台

以下示例演示了如何使用文本演示平台：

1. 通过以下网址打开 Amazon Bedrock 控制台：<https://console.aws.amazon.com/bedrock/>。

2. 在左侧导航窗格中，从演示平台下选择聊天/文本。
3. 选择选择模型，然后选择提供商和模型。在此示例中，选择 Amazon，然后选择 Amazon Nova 2 模型（例如 Nova 2 Lite 或 Nova 2 Sonic）。然后，选择应用。
4. 从文本面板下方选择默认提示，或者在文本面板中输入提示，例如 **Describe the purpose of a "hello world" program in one line.**
5. 要探索 Amazon Nova 的图像理解能力，您可以从计算机上传小于或等于 25 MB 的 JPEG、PNG、GIF 或 WEBP 格式的图像。上传图像后，您可以向 Amazon Nova 询问有关图像的信息。
6. 要探索 Amazon Nova 的文档理解能力，您可以上传小于或等于 4.5 MB 的 CSV、DOC、DOCX、HTML、MD、PDF、TXT、XLS 或 XLSX 格式的文档。上传文档后，您可以向 Amazon Nova 询问有关文档的信息。
7. 要探索 Amazon Nova 的视频理解能力，您可以从计算机上传小于或等于 25 MB 的 MKV、MOV 或 MP4 格式的视频。您可以使用 Amazon S3 来播放最大 1 GB 的视频。上传视频后，您可以向 Amazon Nova 询问有关视频的信息。
8. 选择运行对模型运行推理。生成的文本显示在文本面板的提示下方。

## API 入门

在开始使用 API 之前，如果不熟悉 AWS 或需要安装 AWS CLI 或 AWS SDK，请按照以下步骤操作。如果这些情况皆不适用，请跳至[获取凭证来授予编程访问权限](#)。

### 我是 AWS 新用户

如果您还没有 AWS 账户，请完成以下步骤创建一个。或者，您也可以通过 Amazon Nova（网址：[nova.amazon.com/dev](https://nova.amazon.com/dev)）免费开始构建。

#### 如何注册 AWS 账户

1. 打开 <https://portal.aws.amazon.com/billing/signup>。
2. 按照屏幕上的说明操作。

在注册时，将接到电话或收到短信，要求使用电话键盘输入一个验证码。

当您注册 AWS 账户时，系统会创建一个 AWS 账户根用户。根用户有权访问该账户中的所有 AWS 服务和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行[需要根用户访问权限的任务](#)。

注册过程完成后，AWS 会向您发送一封确认电子邮件。在任何时候，您都可以通过转至 <https://aws.amazon.com/> 并选择我的账户来查看当前的账户活动并管理您的账户。

## 保护您的 AWS 账户根用户

1. 选择根用户并输入 AWS 账户电子邮件地址，以账户所有者身份登录 [AWS 管理控制台](#)。在下一页上，输入您的密码。

要获取使用根用户登录方面的帮助，请参阅《AWS 登录用户指南》中的 [以根用户身份登录](#)。

2. 为您的根用户启用多重身份验证 (MFA)。

有关说明，请参阅《IAM 用户指南》中的 [为 AWS 账户根用户启用虚拟 MFA 设备 \(控制台\)](#)。

## 我需要安装 AWS CLI 或 AWS SDK

要安装 AWS CLI，请按照 [安装或升级到最新版本的 AWS CLI](#) 中的步骤操作。

要安装 AWS SDK，请在 [AWS 开发工具](#) 页面中选择与所用编程语言对应的选项卡。

AWS 软件开发工具包 (SDK) 适用于许多常用编程语言。每个 SDK 都提供 API、代码示例和文档，帮助开发人员更轻松地以其首选语言构建应用程序。SDK 会自动为您执行有用的任务，例如：

- 对服务请求进行加密签名
- 重试请求
- 处理错误响应

## 获取凭证来授予编程访问权限

如果用户需要在 AWS 管理控制台之外与 AWS 交互，则需要编程式访问权限。授予编程式访问权限的方法取决于访问 AWS 的用户类型。

要向用户授予编程式访问权限，请选择以下选项之一。

| 哪个主体需要编程访问权限？ | 目标   | 方式                 |
|---------------|--|--------------------|
| IAM 用户        | 限制对 AWS CLI、AWS SDK 或 AWS API 的编程请求进行签名的长期凭证的有效时长。 | 按照您希望使用的界面的说明进行操作。 |

| 哪个主体需要编程访问权限？                | 目标   | 方式  |
|------------------------------|--|---|
|                              |  | <ul style="list-style-type: none"> <li>有关 AWS CLI，请参阅《AWS 命令行界面用户指南》中的<a href="#">使用 IAM 用户凭证进行身份验证</a>。</li> <li>有关 AWS SDK 和工具的更多信息，请参阅《AWS SDK 和工具参考指南》中的<a href="#">使用长期凭证进行身份验证</a>。</li> <li>有关 AWS API 的更多信息，请参阅《IAM 用户指南》中的<a href="#">管理 IAM 用户的访问密钥</a>。</li> </ul> |
| IAM 角色                       | 使用临时凭证签署向 AWS CLI、AWS SDK 或 AWS API 发出的编程请求。 | 遵循《IAM 用户指南》中 <a href="#">对 AWS 资源使用临时凭证</a> 的相关说明。   |
| 在 IAM Identity Center 中托管的用户 | 使用临时凭证签署向 AWS CLI、AWS SDK 或 AWS API 发出的编程请求。 | <p>按照您希望使用的界面的说明进行操作。</p> <ul style="list-style-type: none"> <li>有关 AWS CLI，请参阅《AWS 命令行界面用户指南》中的<a href="#">将 AWS CLI 配置为使用 IAM Identity Center</a>。</li> <li>有关 AWS SDK、工具和 AWS API，请参阅《AWS SDK 和工具参考指南》中的<a href="#">IAM Identity Center 身份验证</a>。</li> </ul>               |

## 将 Amazon Bedrock 权限附加到用户或角色

在设置好编程访问凭证后，您需要为用户或 IAM 角色配置权限，以便其访问与 Amazon Bedrock 相关的操作。要设置这些权限，请执行以下操作：

1. 在 AWS 管理控制台主页上，选择 IAM 服务或进入 IAM 控制台，网址为 <https://console.aws.amazon.com/iam/>。
2. 选择用户或角色，然后选择您的用户或角色。

3. 在权限选项卡中，选择添加权限，然后选择添加 AWS 托管式策略。选择 [AmazonBedrockFullAccess](#) AWS 托管式策略。
4. 要允许用户或角色订阅模型，请选择创建内联策略，然后在 JSON 编辑器中指定以下权限：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "MarketplaceBedrock",
      "Effect": "Allow",
      "Action": [
        "aws-marketplace:ViewSubscriptions",
        "aws-marketplace:Unsubscribe",
        "aws-marketplace:Subscribe"
      ],
      "Resource": "*"
    }
  ]
}
```

## 使用 Amazon Nova 模型生成对文本提示的回复

满足所有先决条件后，您可以测试通过 [Converse](#) 请求来调用 Amazon Nova 模型。

要安装 AWS CLI，请按照[安装或升级到最新版本的 AWS CLI](#) 中的步骤操作。按照[获取凭证来授予编程访问权限](#)中的步骤，验证是否已设置凭证来使用 Boto3。

要使用 AWS CLI 在 Nova 2 Lite 中生成对文本提示的响应，请在终端中运行以下命令：

```
aws bedrock-runtime converse \
  --model-id us.amazon.nova-2-lite-v1:0 \
  --messages '[{"role":"user","content":[{"text":"Write a short poem"}]}' \
  --additional-model-request-fields '{"reasoningConfig":
{"type":"enabled","maxReasoningEffort":"low"}'}
```

## 核心推理

推理是指向 Amazon Nova 模型发送请求并接收生成响应的过程。Amazon Nova 模型支持通过两种 API 方式进行推理：

- Converse API ( [Converse](#)、[ConverseStream](#) )：提供适用于不同模型的统一接口，便于在模型间切换，或开发可兼容多款模型的应用程序。推荐用于大多数使用案例。
- Invoke API ( [InvokeModel](#)、[InvokeModelWithResponseStream](#) )：请求有效载荷按各模型原生格式专门构建。使用请求正文中提供的提示词与推理参数执行推理。

这两种 API 都支持相同的核心功能，包括：

- 多回合对话
- 多模态输入 ( 文本、图像、视频、音频 )
- 工具使用
- 护栏
- 流式响应

这两种 API 的请求结构几乎相同，仅在字节数据 ( 文档、图像、视频和音频 ) 的编码方式上有所不同。对于 Amazon Nova 模型专属的请求参数 ( 例如 `reasoningConfig` 和 `TopK` )，需将其置于 `additionalModelRequestFields` 中的另一个 `inferenceConfig` 对象中。这些是 `InvokeModel` 和 `InvokeModelWithResponseStream` 的顶级参数。

### Note

有关 Nova 2 模型的代码示例列表，请参阅“代码示例”和“故障排除”部分。

要使用 Amazon Nova 模型，请将 `modelId` 设置为以下任一值：

#### 模型

#### 模型 ID

Nova 2 Lite

- `global.amazon.nova-2-lite-v1:0`
- `us.amazon.nova-2-lite-v1:0`

| 模型                         | 模型 ID   |
|----------------------------|---|
| Nova 2 Sonic               | <ul style="list-style-type: none"><li>global.amazon.nova-2-sonic-v1:0</li><li>us.amazon.nova-2-sonic-v1:0</li></ul> |
| Nova Multimodal Embeddings | amazon.nova-2-multimodal-embeddings-v1:0  |

## 重要提示：超时配置

### Important

Amazon Nova 推理请求最长可能需要 60 分钟才能完成。请对客户端超时设置进行相应配置：以下示例为 Python 代码。用户可以在该 SDK 的 API 文档中查看其首选 SDK 语言版本的文档。

```
from botocore.config import Config

bedrock = boto3.client(
    'bedrock-runtime',
    region_name='us-east-1',
    config=Config(
        read_timeout=3600 # 60 minutes
    )
)
```

## 核心推理主题

本节将讨论以下主题：

- 使用 Converse API：提供与大多数 Bedrock 模型兼容的统一接口
- 使用 Invoke API：该接口为 Nova 模型专属，不兼容其他 Bedrock 模型
- 流式响应：生成实时响应
- 使用 Amazon Nova Embeddings：文本嵌入功能
- 按需推理：按使用量付费的推理模型

# 使用 Converse API

Converse API 为与 Amazon Nova 模型交互提供统一接口。该接口抽象封装了模型专属细节，并以统一方式处理所有 Amazon Nova 模型的多轮对话、系统提示与流式响应。

## 主题

- [请求结构](#)
- [使用系统提示词](#)
- [推理参数](#)
- [使用推理](#)

## 请求结构

- 多轮对话：在多次交互中保留上下文信息
- 系统提示：设定角色、回复规范等系统指令
- 文档聊天：与单个或批量文档进行交互、查询
- 视觉能力：处理并分析图像与视频内容
- 工具使用：支持模型使用外部工具和 API
- 安全护栏：应用内容过滤与安全控制策略
- 推理能力：通过扩展思考实现复杂问题求解

Converse API 基本请求包含模型 ID 和消息列表：

```
import boto3

bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')

response = bedrock.converse(
    modelId='us.amazon.nova-2-lite-v1:0',
    messages=[
        {
            'role': 'user',
            'content': [{'text': 'What is machine learning?'}]
        }
    ]
)
```

```
content_list = response["output"]["message"]["content"]
# Extract the first text block
text = next((item["text"] for item in content_list if "text" in item), None)
if text is not None:
    print(text)
```

## 使用系统提示词

系统提示词为模型提供上下文信息和操作指令：

```
import boto3

bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')

response = bedrock.converse(
    modelId='us.amazon.nova-2-lite-v1:0',
    system=[
        {'text': 'You are a helpful AI assistant specializing in cloud computing.'}
    ],
    messages=[
        {
            'role': 'user',
            'content': [{'text': 'Explain serverless computing.'}]
        }
    ]
)

# Print the response text
content_list = response["output"]["message"]["content"]
text = next((item["text"] for item in content_list if "text" in item), None)
if text is not None:
    print(text)
```

## 推理参数

使用推理参数控制模型的输出。以下推理参数可用。

- `maxTokens` ( 整数 ) : 要生成的最大词元数 ( 最多 65,000 个 )。未指定时, 模型将根据请求上下文使用动态默认值。
- `temperature` ( 浮点数 ) : 控制随机性 ( 0.0 – 1.0 , 默认值 0.7 )。数值越低, 输出结果越具确定性

- topP ( 浮点数 ) : 核采样阈值 ( 0 – 1 , 默认值 0.9 ) 。数值越低 , 输出结果越聚焦
- stopSequences ( 数组 ) : 遇到指定字符序列时停止生成

示例 :

```
import boto3
bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')

response = bedrock.converse(
    modelId='us.amazon.nova-2-lite-v1:0',
    messages=[
        {
            'role': 'user',
            'content': [{'text': 'Write a short story.'}]
        }
    ],
    inferenceConfig={
        'maxTokens': 512,
        'temperature': 0.7,
        'topP': 0.9,
        'stopSequences': ['END']
    }
)

content_list = response["output"]["message"]["content"]
text = next((item["text"] for item in content_list if "text" in item), None)
if text is not None:
    print(text)
```

## 使用推理

Nova 2 Lite 支持扩展思考能力 , 可用于复杂问题求解。使用 reasoningConfig 启用推理功能。

默认情况下 , 推理功能处于禁用状态 , 以便优化简单查询的速度和成本。如需处理更复杂的任务 , 可以启用推理功能。Nova 2 通过三个强度级别提供对推理深度的灵活控制 :

低强度 (maxReasoningEffort: "low")

适用场景 : 复杂度更高、需结构化思考的任务。例如代码审查与优化建议 ( 模型需细致评估现有代码质量 ) 、需综合考量多重要素的分析任务 , 或适合采用系统化方法解决的问题场景。低强度适用于复合型任务 , 仅需基础推理即可提升精度 , 无需开展深度多步骤规划。

## 中强度 (maxReasoningEffort: "medium")

适用场景：多步骤任务和编码 workflows。例如软件开发与调试（模型需在修改前理解现有代码结构）、跨多个文件或组件协同的代码生成、存在相互依赖关系的多步骤计算，以及带有多重约束条件的规划任务。中强度适用于需协调多个工具且模型需在一系列连续操作中保持上下文的智能代理 workflows，效果最佳。

## 高强度 (maxReasoningEffort: "high")

适用场景：STEM 推理和高级问题求解。例如需要严谨分步验证的高等数学问题与证明、需深度探究的科学分析研究任务、涉及多维度架构考量的复杂系统设计，以及影响重大的关键决策场景。高强度可在需要高阶推理、审慎评估备选方案并全面验证结论的任务中，实现最高精度。

以下示例展示了不同级别的推理强度：

### Low effort

```
import boto3

bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')

response = bedrock.converse(
    modelId='us.amazon.nova-2-lite-v1:0',
    system=[{"text": "You are a highly capable personal assistant"}],
    messages=[{
        "role": "user",
        "content": [{"text": "Provide a meal plan for a gluten free family of 4."}]
    }],
    inferenceConfig={
        "temperature": 0.7,
        "topP": 0.9,
        "maxTokens": 10000
    },
    additionalModelRequestFields={
        "reasoningConfig": {
            "type": "enabled",
            "maxReasoningEffort": "low"
        }
    }
)
```

## 推理参数：

以下为推理相关参数

- type : enabled 或 disabled ( 默认值 : disabled )
- maxReasoningEffort : low、medium 或 high。启用推理功能时，此参数为必填项。

### Note

将 maxReasoningEffort 设置为 high 时，不可同时使用 temperature、topP 和 topK 参数。因为组合使用会导致错误。

响应包含推理内容：

```
{
  "output": {
    "message": {
      "role": "assistant",
      "content": [
        {
          "reasoningContent": {
            "reasoningText": {
              "text": "[REDACTED]"
            }
          }
        },
        {
          "text": "Based on the premises, we can conclude..."
        }
      ]
    }
  },
  "stopReason": "end_turn"
}
```

### Note

在 Amazon Nova 2 中，推理内容会显示为 [REDACTED]。相关推理词元仍会计费，因其有助于提升输出质量。当前在响应结构中保留该字段，是为了便于未来开放显示推理内容。我们正积极与客户协作，确定展示模型推理过程的最优方案。

## 使用 Invoke API

Invoke API 可直接调用 Amazon Nova 模型，并能更精细地控制请求与响应格式。不同于对模型专属细节进行抽象封装的 Converse API，Invoke API 允许直接使用模型原生的请求与响应结构。

### Note

Invoke API 支持与 Converse API 相同的功能，但文档输入模式除外，该能力为 Converse API 独有。

## 请求结构

Invoke API 请求需要模型 ID 和 JSON 请求正文：

```
import boto3
import json

bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')

request_body = {
    'messages': [
        {
            'role': 'user',
            'content': [{'text': 'What is machine learning?'}]
        }
    ],
    'inferenceConfig': {
        'maxTokens': 512,
        'temperature': 0.7
    }
}

response = bedrock.invoke_model(
```

```
    modelId='us.amazon.nova-2-lite-v1:0',
    body=json.dumps(request_body)
)

response_body = json.loads(response['body'].read())
content_list = response_body["output"]["message"]["content"]
# Extract the first text block
text_block = next((item for item in content_list if "text" in item), None)
if text_block is not None:
    print(text_block["text"])
```

## 请求参数

Invoke API 支持以下关键参数：

- `messages`：对话消息数组，包含角色和内容
- `system`：可选系统提示词，用于提供上下文信息和操作指令
- `inferenceConfig`：控制模型输出的参数  
(`temperature`、`maxTokens`、`topP`、`topK`、`stopSequences`、`reasoningConfig`)
- `toolConfig`：函数调用相关配置，包含工具规范与工具选择策略

## 流式响应

流式传输可让您在模型生成响应的同时逐步接收内容，带来更流畅的交互体验。Converse API 和 Invoke API 均支持流式传输。

## 使用 ConverseStream 进行流式传输

使用 ConverseStream 以事件流形式接收响应：

```
import boto3

bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')

response = bedrock.converse_stream(
    modelId='us.amazon.nova-2-lite-v1:0',
    messages=[
        {
            'role': 'user',
            'content': [{'text': 'Write a short story about AI.'}]
        }
    ]
)
```

```
    }
  ]
)

for event in response['stream']:
    if 'contentBlockDelta' in event:
        delta = event['contentBlockDelta']['delta']
        if 'text' in delta:
            print(delta['text'], end='', flush=True)
```

## 使用 InvokeModelWithResponseStream 进行流式响应

使用 InvokeModelWithResponseStream 实现 Invoke API 的流式传输：

```
import boto3
import json

bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')

request_body = {
    'messages': [
        {
            'role': 'user',
            'content': [{'text': 'Explain quantum computing.'}]
        }
    ]
}

response = bedrock.invoke_model_with_response_stream(
    modelId='us.amazon.nova-2-lite-v1:0',
    body=json.dumps(request_body)
)

for event in response['body']:
    chunk = json.loads(event['chunk']['bytes'])
    if 'contentBlockDelta' in chunk:
        delta = chunk['contentBlockDelta']['delta']
        if 'text' in delta:
            print(delta['text'], end='', flush=True)
```

## 流事件类型

流式响应包含以下事件类型：

- `messageStart` : 表示消息开始
- `contentBlockStart` : 表示内容块开始
- `contentBlockDelta` : 包含增量文本或数据
- `contentBlockStop` : 表示内容块结束
- `messageStop` : 表示消息结束，并附有停止原因
- `metadata` : 包含用量信息 ( 词元计数 )

## 使用 Amazon Nova Embeddings

Amazon Nova Multimodal Embeddings 是一种多模态嵌入模型，可用于代理式 RAG 和语义搜索应用程序。其通过单个模型支持文本、文档、图像、视频和音频，从而实现跨模态检索。Nova Multimodal Embeddings 将各类内容映射到统一语义空间，支持执行单模态、跨模态和多模态向量操作。

当一段内容通过 Nova 嵌入传递时，模型会将该内容转换为通用的数值格式，称为向量。向量是一组数值，可用于各种搜索功能。与不太相似的内容相比，为相似内容分配的向量会更加接近。

应用程序：

- 语义内容检索和推荐：为内容生成嵌入，然后用来查找相似项目或提供个性化推荐。
- 多模态搜索：融合不同内容类型的嵌入向量，实现跨模态搜索能力。
- RAG：利用多模态内容（例如含有交错文本和图像的文档）生成嵌入，从而支持生成式人工智能应用程序的检索 workflow。

## 主要特征

这些关键特性使 Nova Embeddings 与众不同：

- 支持在统一语义空间中处理文本、图像、文档图像、视频和音频。最大上下文长度为 8K 词元或 30s 视频和 30s 音频。
- 同步和异步 API：该 API 支持同步和异步使用。
- 大文件分段：异步 API 通过为长文本、视频和音频提供 API 构建的分段，并由用户定义的控制，从而轻松处理大型输入。该模型将为每个分段生成一个嵌入。
- 带音频的视频：同时处理带音频的视频。指定是采用同时表示两种模态的单个嵌入，还是分别生成两个独立的嵌入。

- 嵌入用途：根据下游应用场景（检索/RAG/搜索、分类、聚类）对嵌入进行优化。
- 尺寸大小：4 个平衡嵌入准确度要求和向量存储成本的尺寸大小：3072；1024；384；256。
- 输入方式：通过指定 S3 URI 传入待嵌入内容，或以 Base64 编码形式内联传入。

## 生成嵌入

完成以下步骤以生成嵌入。

文本同步嵌入：

```
import boto3
import json

# Create the Bedrock Runtime client.
bedrock_runtime = boto3.client(
    service_name='bedrock-runtime',
    region_name='us-east-1',
)

# Define the request body.
request_body = {
    'taskType': 'SINGLE_EMBEDDING',
    'singleEmbeddingParams': {
        'embeddingPurpose': 'GENERIC_INDEX',
        'embeddingDimension': 3072,
        'text': {'truncationMode': 'END', 'value': 'Hello, World!'},
    },
}

try:
    # Invoke the Nova Embeddings model.
    response = bedrock_runtime.invoke_model(
        body=json.dumps(request_body, indent=2),
        modelId='amazon.nova-2-multimodal-embeddings-v1:0',
        accept='application/json',
        contentType='application/json',
    )

except Exception as e:
    # Add your own exception handling here.
    print(e)
```

```
# Print the request ID.
print('Request ID:', response.get('ResponseMetadata').get('RequestId'))

# Print the response body.
response_body = json.loads(response.get('body').read())
print(json.dumps(response_body, indent=2))
```

尝试执行：

```
response = bedrock_runtime.invoke_model(
#####body=json.dumps(request_body, indent=2),
#####modelId='amazon.nova-2-multimodal-embeddings-v1:0',
#####accept='application/json',
#####contentType='application/json',
####)
except Exception as e:
####print(e)

print('Request ID:', response.get('ResponseMetadata').get('RequestId'))
response_body = json.loads(response.get('body').read())
print(json.dumps(response_body, indent=2))
```

视频异步嵌入：

```
import boto3

bedrock_runtime = boto3.client(
####service_name='bedrock-runtime',
####region_name='us-east-1',
)

model_input = {
####'taskType': 'SEGMENTED_EMBEDDING',
####'segmentedEmbeddingParams': {
#####'embeddingPurpose': 'GENERIC_INDEX',
#####'embeddingDimension': 3072,
#####'video': {
#####'format': 'mp4',
#####'embeddingMode': 'AUDIO_VIDEO_COMBINED',
#####'source': {
#####'s3Location': {'uri': 's3://my-bucket/path/to/video.mp4'}
#####},
#####'segmentationConfig': {
```

```
#####'durationSeconds': 15### Segment into 15 second chunks
#####},
#####},
####},
}
```

尝试执行：

```
response = bedrock_runtime.start_async_invoke(
#####modelId='amazon.nova-2-multimodal-embeddings-v1:0',
#####modelInput=model_input,
#####outputDataConfig={
#####'s3OutputDataConfig': {
#####'s3Uri': 's3://my-bucket'
#####}
#####},
####)
except Exception as e:
####print(e)

print('Request ID:', response.get('ResponseMetadata').get('RequestId'))
print('Invocation ARN:', response.get('invocationArn'))
```

## 批量推理

使用批量推理，您可以提交多个请求并异步生成嵌入。批量推理通过发送单个请求并在 Amazon S3 存储桶中生成响应，帮助您高效地处理大量请求。

- 在您创建的文件中定义模型输入后，您需要将相应文件上传到 S3 存储桶。
- 然后，您需要提交批量推理请求并指定 S3 存储桶。
- 作业完成后，您可以从 S3 检索输出文件。
- 您可以使用批量推理来提高对大型数据集的模型推理性能。

## 格式化并上传批量推理数据

您必须将批量推理数据添加到在提交模型调用作业时要选择或指定的 S3 位置。S3 位置必须包含以下项目：

至少一个定义模型输入的 JSONL 文件。一个包含 JSON 对象行的 JSONL。JSONL 文件必须以扩展名 .jsonl 结尾，并且为以下格式：

```
{
  "recordId": "record001",
  "modelInput": {
    "taskType": "SINGLE_EMBEDDING",
    "singleEmbeddingParams": {
      "embeddingPurpose": "GENERIC_INDEX",
      "embeddingDimension": 3072,
      "text": {
        "source": {
          "s3Location": {
            "uri": "s3://batch-inference-input-bucket/text_001.txt",
            "bucketOwner": "111122223333"
          }
        },
        "truncationMode": "END"
      }
    }
  }
}
```

### 支持的输入文件类型

- 图像格式：PNG、JPEG、WEBP、GIF
- 音频格式：MP3、WAV、OGG
- 视频格式：MP4、MOV、MKV、WEBM、FLV、MPEG、MPG、WMV、3GP

## 按需型推理

按需推理提供无服务器方式访问 Amazon Nova 模型，无需预先配置算力资源。该模式可自动扩缩容以适配工作负载，并按实际使用量计费。

## 优势

按需推理具有以下多项优势：

- 无需容量规划：自动扩缩容以满足需求
- 按使用量付费：仅按处理的词元计费
- 即时可用：无需预置或预热时间
- 经济高效：适用于波动或不可预测的工作负载

## 使用按需推理

按需推理是 Amazon Nova 模型的默认模式。只需在进行 API 调用时指定模型 ID 即可：

```
import boto3

bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')

response = bedrock.converse(
    modelId='us.amazon.nova-2-lite-v1:0',
    messages=[
        {
            'role': 'user',
            'content': [{'text': 'Hello, Nova!'}]
        }
    ]
)

# Print the response text
content_list = response["output"]["message"]["content"]
text = next((item["text"] for item in content_list if "text" in item), None)
if text is not None:
    print(text)
```

## 定价

按需推理根据处理的输入和输出词元数量计费。有关当前定价详细信息，请参阅 [Amazon Bedrock 定价](#)。

## 限额和限制

按需推理具有默认配额，具体配额因模型和区域而异。要请求增加配额，可以使用 [服务配额控制台](#)。

## 使用 Amazon Nova 2 的功能

本指南提供了基于任务的方法和实际示例，可帮助您快速有效地将 Nova 的功能集成到应用程序。本指南专为想要实现以下目标的开发人员和工程师而设计：

- 通过 API 将 Nova 模型集成到应用程序
- 了解预处理要求与优化策略
- 构建多模态理解 workflow

每个部分都提供：

- 有关支持的格式、大小限制和词元估算的关键技术信息
- 演示如何通过 API 同时实现直接上传和 Amazon S3 调用的代码示例
- 展示实际落地应用的使用案例
- 设计应用程序时应考虑的限制

### 先决条件

在开始之前，请确保您满足以下条件：

- 可访问 Amazon Bedrock 的 AWS 账户
- 调用 Bedrock 模型所需的对应 IAM 权限
- 已安装 AWS SDK (适用于 Python 的 Boto3 示例)
- 基本了解 AWS 服务，例如 Amazon S3 (用于处理大型文件)

## 推理能力

Amazon Nova 2 Lite 支持扩展思考，默认情况下该功能处于禁用状态。启用后，模型会生成内部推理词元，从而提高回复质量。在 Amazon Nova 2 Lite 中，推理内容会在输出时被隐去并显示为 [REDACTED]，但系统仍会对这部分词元计费。

响应结构中包含推理字段，以便在后续版本中可选择开放该内容。有关扩展思考的更多信息，请参阅 [the section called “扩展思考”](#) 和 [the section called “代码库”](#)。

## 多模态理解

Amazon Nova 2 Lite 可以理解多种输入模态。此模型搭载视觉能力，可理解并分析图像、文档、视频与语音，依据输入内容进行推理并作答。

本节概述了在 Amazon Nova 中处理图像、文档和视频的指南，包括使用的预处理策略、代码示例和需考虑的相关限制。

### 按模态划分的支持内容类型

以下信息详细说明了每个媒体文件类型支持的文件格式和接受的输入方法。

| 媒体文件类型 | 支持的文件格式   | 输入方法   | 大小限制    | 对象数量 |
|--------|---|--|---------|------|
| 图像     | PNG、JPEG、GIF、WebP<br><br>注意：如果使用动画 GIF 或 WebP 文件，则只使用第一帧。 | 在请求中嵌入数据<br><br>如果使用 Converse API，请将数据编码为字节。<br><br>如果使用调用 API，请将数据编码为 Base64 字符串。 | 25 MB   | 5    |
|        |   | Amazon S3 URI  | 总计 2 GB | 1000 |
| 视频     | MP4、MOV、MKV、WebM、FLV、MPEG、MPEG-4、WMV、3GP                  | 在请求中嵌入数据<br><br>如果使用 Converse API，请将数据编码为字节。<br><br>如果使用调用 API，请将数据编码为 Base64 字符串。 | 25 MB   | 1    |
|        |   | Amazon S3 URI  | 1 GB    | 1    |

#### 主题

- [图像理解](#)
- [视频理解](#)
- [文档理解](#)

## 图像理解

图像理解是指 Amazon Nova 对图像进行处理并执行各类计算机视觉任务的能力，例如：

- 执行物体检测
- 通过视觉问答 ( VQA ) 回答有关图像的问题
- 对图像进行分类与汇总
- 执行边界框检测
- 光学字符识别 ( OCR )
- 物体计数

可将图像作为提示，以字节数组形式或通过 S3 URI 传递给 API。

### 关键技术信息

使用该功能时，请注意以下关键技术信息。

#### 图像尺寸与缩放处理

Amazon Nova 会自动对图像进行缩放，兼顾画质与性能：

- 确定最接近的宽高比 ( 例如 1:1、1:2、2:3 )
- 缩放规则：将图像缩放至其中一边不小于 896 像素，或与原图较短边等长，取两者中较大值
- 保持宽高比
- 最高支持 8000 x 8000 像素分辨率

边界框坐标：

- 适用于识别截图中的元素、图像定位等任务
- 后期处理中可对坐标进行缩放，以匹配图像原始尺寸
- 边界框返回值采用 [0, 1000] 尺度。

### 图像词元估计

Amazon Nova 可将每张图像转换为词元进行处理。词元数量取决于图像的分辨率和宽高比。

以下是基于图像分辨率的近似词元计数示例：

| 图像分辨率       | 估计的词元 |
|-------------|-------|
| 900 x 450   | 515   |
| 900 x 900   | ~1035 |
| 1400 x 900  | ~1600 |
| 1800 x 900  | ~2060 |
| 1300 x 1300 | ~2155 |

## 图像理解示例

有关如何直接在请求中嵌入图像数据的示例，请参阅[代码库](#)中的“使用嵌入式资产的多模态输入 – Converse API (非流式传输) 示例”。

要上传总有效载荷大于 25 MB 的大型图像文件或多个图像文件，可以使用 Amazon S3。有关如何使用 Amazon S3 URI 引用进行图像输入的完整示例，请参阅[代码库](#)中的“使用 S3 URI 的多模态输入 – Converse API (非流式传输) 示例”。

### Note

使用 S3 时，需确保 Amazon Bedrock 服务有权访问存储桶和对象。

## 关键限制

以下列表概述了图像理解模型的当前限制：

- 多语言图像理解：模型对多语言图像和视频帧的理解有限，在简单任务中可能会遇到困难或产生幻觉。
- 人物识别：Amazon Nova 2 模型不支持在图像、文档或视频中识别或命名个人的功能。
- 空间推理：Amazon Nova 2 模型的空间推理能力有限。它们可能很难完成需要精确本地化或布局分析的任务。
- 图像/视频中的小文本：如果图像或视频中的文本太小，可以考虑通过裁剪到相关部分来增加图像中文本的相对大小，同时保留必要的上下文。

## 视频理解

视频理解是指 Amazon Nova 对视频输入进行处理，并执行各类视频解析任务的能力，例如：

- 分析关键帧并总结视频内容
- 回答有关视频片段的问题（视频问答）
- 跨帧物体检测与跟踪
- 识别动作、场景与事件
- 执行时序分割，定位特定画面节点
- 视频片段生成描述性字幕或摘要

### 关键技术信息

使用该功能时，请注意以下关键技术信息。

#### 视频大小信息

Amazon Nova 视频理解功能支持多种宽高比。将所有视频的大小按畸变调整为（根据原始宽高比，可能是放大或缩小）672 x 672 平方尺寸，然后再将其输入到模型。

该模型根据视频长度使用动态采样策略。对于时长不超过 16 分钟的视频，Amazon Nova 2 Lite 采用每秒 1 帧（1 FPS）的采样率。对于时长超过 16 分钟的视频，系统将降低采样率，统一采样 960 帧，帧采样速率会随之调整。这种方法旨在为较短的视频提供比较长的视频内容更准确的场景级视频理解。

对于低动态视频，建议将视频时长保持在 1 小时以内；对于高动态视频，建议将视频时长保持在 16 分钟以内。

在分析 4k 版本的视频和全高清版本视频时，应该没有区别。同样，采样率为 1 FPS，因此 60 FPS 的视频应该与 30 FPS 的视频的表现一样好。受视频大小 1 GB 限制，采用超出实际需求的分辨率与帧率并无益处。否则会缩短满足大小限制的视频时长，建议对超过 1 GB 的视频进行预处理。

#### 视频词元

视频的时长是影响生成词元数量的主要因素。要计算近似成本，应将视频词元的估计数量乘以所使用的特定模型的每词元价格。

下表提供了 Amazon Nova 2 Lite 的每个视频长度的帧采样和词元利用率的一些近似值：

| 视频长度  | 采样帧数 | 采样率 (FPS) | 近似词元数  |
|-------|------|-----------|--------|
| 10 秒  | 10   | 1         | 2,880  |
| 30 秒  | 30   | 1         | 8640   |
| 16 分钟 | 960  | 1         | 276480 |
| 20 分钟 | 1200 | 1         | 345600 |
| 30 分钟 | 1800 | 1         | 518400 |
| 45 分钟 | 2700 | 1         | 777600 |

## 视频理解示例

有关如何直接在请求中嵌入视频数据的示例，请参阅[代码库](#)中的“使用嵌入式资产的多模态输入 – Converse API (非流式传输) 示例”。

有关如何在视频输入中使用 S3 URI 引用的示例，请参阅[代码库](#)中的“使用 S3 URI 的多模态输入 – Converse API (非流式传输) 示例”。

## 关键限制

以下是关键的模型限制，这些情况下可能无法保证模型的准确性和性能：

- 不支持音频：Amazon Nova 模型目前接受过训练，只能根据视觉帧来处理和理解视频内容。视频中的音轨不参与处理与分析。
- 多语言图像理解：Amazon Nova 模型对多语言图像和视频帧的理解有限。在类似的任务中可能会遇到困难或产生幻觉。
- 人物识别：Amazon Nova 模型不支持在图像、文档或视频中识别或命名个人的功能。模型不会提供视觉内容中人物的姓名与身份信息。
- 视频中的小文本：如果图像或视频中的文本太小，可以考虑增加视频中文本的相对大小。
- 空间推理：Amazon Nova 2 模型的空间推理能力有限。对于需要精准识别视频中物体位置、距离或空间关系的任务，模型效果可能有限。
- 不当内容：Amazon Nova 模型不会处理违反可接受使用政策的不当或露骨图像。

- 医疗保健应用：尽管 Amazon Nova 模型可以对某些医疗保健图像或视频进行一般分析，但由于这些构件的敏感性，我们不建议利用这些分析结果来解释复杂的诊断扫描等敏感的医疗图像。Amazon Nova 模型的回复绝不能替代专业的医学建议。

## 文档理解

借助 Amazon Nova 的文档理解功能，您可以在提示中包含整个文档（PDF、Word 文件、电子表格等）。这使模型能够分析、总结、提取信息或回答有关文档内容的问题。

Amazon Nova 2 Lite 可解释这些文档中的文本与视觉元素（例如图表或表格）。这支持多种用例，比如对冗长报告或扫描文档进行问答、概括和分析。

文档理解核心特性包括支持长文档的超大上下文窗口（100 万词元），以及单次查询处理多文档的能力。

### 支持的文档模式与格式

Amazon Nova 区分了以下两种类型的文档输入：

- 基于文本的文档（例如 TXT、CSV、HTML、Markdown 或 DOC 文件）主要处理其文本内容。Amazon Nova 可理解并提取这些文档的文本并从中提取信息。
- 基于媒体的文档（例如 PDF 或 DOCX 文件）可能含有复杂的布局、图像、图表或嵌入式图形。对于基于媒体的文档，Amazon Nova 采用基于视觉的理解功能来解释文档文本旁边的视觉内容，例如图表、表格、示意图或屏幕截图。

支持的文件格式包含各类常用文档，例如：

- 纯文本与结构化文本文件：CSV、TXT
- 电子表格：XLS、XLSX、HTML、Markdown
- 标准图像格式（适用于文档中的图像）：PNG、JPG、GIF、WebP
- 文档格式：DOC、DOCX、PDF
- 不支持包含图像编码（如 CMYK 或 SVG）的 PDF 文件。

## 文档大小限制和使用指南

| 约束          | 限制  |
|-------------|---|
| 最大文档数       | 每个请求最多 5 份文档 ( 适用于直接上传和 Amazon S3 )   |
| 基于文本的文档大小   | 每个文本文档必须等于或小于 4.5 MB  |
| 基于媒体的文档大小   | PDF 和 DOCX 文件没有单独的文件大小限制。使用直接上传时，所有媒体文档的总大小必须小于或等于 25 MB。使用 Amazon S3 时，所有媒体文档的总大小必须小于或等于 2 GB。 |
| 不支持的 PDF 内容 | 不支持含有 CMYK 颜色配置文件或 SVG 图像的 PDF。   |

## 定价

Amazon Nova 采用按词元计费模式：费用包含输入词元 ( 发送的全部内容，含附件文档 ) 与输出词元 ( 模型返回结果 )。

估算 PDF 的词元数：为进行规划，假设标准的 8.5 x 11 英寸 PDF 页面  $\approx$  2560 个输入词元 ( 此估计值涵盖典型页面上的文本和视觉元素 )。

### 示例：通过 API 和 S3 使用 Nova 的文档理解功能

如需查看通过 API 使用的示例，请参阅[代码库](#)中的“使用嵌入式资产的多模态输入 – Converse API ( 非流式传输 ) 示例”。

如需查看通过 S3 使用的示例，请参阅[代码库](#)中的“使用 S3 URI 的多模态输入 – Converse API ( 非流式传输 ) 示例”。

# 语音转语音 ( Amazon Nova 2 Sonic )

Amazon Nova 2 Sonic 支持带有语音输入和输出的实时对话式人工智能。以下章节介绍用于构建交互式语音助手、客户服务自动化系统及对话式应用程序的高级功能。

## 主要特征

Amazon Nova 2 Sonic 提供以下功能：

- 先进的流式语音理解技术，具有双向流 API，可实现实时、低延迟的多轮对话。
- 具有自动语言检测与切换功能的多语言支持。系统提供具备丰富表现力的语音，包含男性音色与女性音色，支持以下语言：
  - 英语 ( 美国、英国、印度、澳大利亚 )
  - 法语
  - 意大利语
  - 德语
  - 西班牙语
  - 葡萄牙语
  - 印地语
- 支持多语言的合成语音，可流畅使用全部支持语言，即使用户在同一会话中切换语言，也能提供连贯一致的用户体验。
- 针对真实部署场景具备出色的抗背景噪声能力。
- 对各支持语言的不同口音具备良好的适应能力。
- 为所有支持语言提供上下文感知丰富、自然拟人的对话式人工智能体验。
- 自适应语音回复，可根据输入语音的韵律动态调整传送方式。
- 智能对话交互，可精准识别用户语句结束时机与助手应答节点，打造自然流畅的对话节奏。
- 妥善处理用户打断情况，不会丢弃对话上下文。
- 使用检索增强生成 ( RAG ) 技术，基于企业数据进行知识基础构建。
- 支持函数调用和座席工作流程，用于构建复杂的人工智能应用程序。
- 异步工具处理，在维持对话流畅性的同时执行工具调用，使助手可在工具于后台处理期间持续发言。
- 跨模态输入支持同一对话中的音频和文本输入，实现灵活的交互模式。
- 连接时长限制为 8 分钟，代码示例中提供了连接续约与会话延续方案。

## 开始使用语音转语音功能

以下章节通过示例与分步讲解，介绍如何使用 Amazon Nova 2 Sonic 实现简易的实时音频流应用程序。这个简化版本演示了使用 Amazon Nova 2 Sonic 模型创建音频对话所需的核心功能。

我们的 [Nova 示例 GitHub 存储库](#) 中列出了以下示例。

连接时长限制为 8 分钟，[GitHub](#) 上提供了连接续约与对话持续方案。

### 说明导入和配置

本节会导入必要的库并设置音频配置参数：

- `asyncio`：用于异步编程
- `base64`：用于对音频数据进行编码和解码
- `pyaudio`：用于音频采集和播放
- Amazon Bedrock SDK 流组件
- 音频常量定义了音频采集的格式（16 kHz 采样率，单声道）

```
import os
import asyncio
import base64
import json
import uuid
import pyaudio
from aws_sdk_bedrock_runtime.client import BedrockRuntimeClient,
    InvokeModelWithBidirectionalStreamOperationInput
from aws_sdk_bedrock_runtime.models import
    InvokeModelWithBidirectionalStreamInputChunk, BidirectionalInputPayloadPart
from aws_sdk_bedrock_runtime.config import Config, HTTPAuthSchemeResolver,
    SigV4AuthScheme
from smithy_aws_core.identity import EnvironmentCredentialsResolver

# Audio configuration
INPUT_SAMPLE_RATE = 16000
OUTPUT_SAMPLE_RATE = 24000
CHANNELS = 1
FORMAT = pyaudio.paInt16
CHUNK_SIZE = 1024
```

## 定义 SimpleNovaSonic 类

SimpleNovaSonic 类是处理 Amazon Nova Sonic 交互的主类：

- `model_id` : Amazon Nova Sonic 模型 ID (`amazon.nova-2-sonic-v1:0`)
- `region` : AWS 区域，默认为 `us-east-1`
- 用于提示和内容跟踪的唯一 ID
- 用于音频播放的异步队列

```
class SimpleNovaSonic:
    def __init__(self, model_id='amazon.nova-2-sonic-v1:0', region='us-east-1'):
        self.model_id = model_id
        self.region = region
        self.client = None
        self.stream = None
        self.response = None
        self.is_active = False
        self.prompt_name = str(uuid.uuid4())
        self.content_name = str(uuid.uuid4())
        self.audio_content_name = str(uuid.uuid4())
        self.audio_queue = asyncio.Queue()
        self.display_assistant_text = False
```

## 初始化客户端

此方法使用以下信息来配置 Amazon Bedrock 客户端：

- 指定区域的相应端点
- 使用环境变量作为 AWS 凭证的身份证信息
- 适用于 AWS API 调用的 SigV4 身份证方案

```
def _initialize_client(self):
    """Initialize the Bedrock client."""
    config = Config(
        endpoint_uri=f"https://bedrock-runtime.{self.region}.amazonaws.com",
        region=self.region,
        aws_credentials_identity_resolver=EnvironmentCredentialsResolver(),
        auth_scheme_resolver=HTTPAuthSchemeResolver(),
```

```
        auth_schemes={"aws.auth#sigv4": SigV4AuthScheme(service="bedrock")}  
    )  
    self.client = BedrockRuntimeClient(config=config)
```

## 处理事件

这个辅助方法将 JSON 事件发送到双向流，该流用于与 Amazon Nova Sonic 模型的所有通信：

```
async def send_event(self, event_json):  
    """Send an event to the stream."""  
    event = InvokeModelWithBidirectionalStreamInputChunk(  
        value=BidirectionalInputPayloadPart(bytes_=event_json.encode('utf-8'))  
    )  
    await self.stream.input_stream.send(event)
```

## 启动会话

这个方法可启动会话并设置其余事件来启动音频流。这些事件需要按相同的顺序发送。

```
async def start_session(self):  
    """Start a new session with Nova Sonic."""  
    if not self.client:  
        self._initialize_client()  
  
    # Initialize the stream  
    self.stream = await self.client.invoke_model_with_bidirectional_stream(  
        InvokeModelWithBidirectionalStreamOperationInput(model_id=self.model_id)  
    )  
    self.is_active = True  
  
    # Send session start event  
    session_start = ''  
    {  
        "event": {  
            "sessionStart": {  
                "inferenceConfiguration": {  
                    "maxTokens": 1024,  
                    "topP": 0.9,  
                    "temperature": 0.7  
                },  
                "turnDetectionConfiguration": {  
                    "endpointingSensitivity": "HIGH"  
                }  
            }  
        }  
    }
```

```
    }
  }
}
...
await self.send_event(session_start)

# Send prompt start event
prompt_start = f'''
{{
  "event": {{
    "promptStart": {{
      "promptName": "{self.prompt_name}",
      "textOutputConfiguration": {{
        "mediaType": "text/plain"
      }}
    }},
    "audioOutputConfiguration": {{
      "mediaType": "audio/lpcm",
      "sampleRateHertz": 24000,
      "sampleSizeBits": 16,
      "channelCount": 1,
      "voiceId": "matthew",
      "encoding": "base64",
      "audioType": "SPEECH"
    }}
  }}
}}
}}
...
await self.send_event(prompt_start)

# Send system prompt
text_content_start = f'''
{{
  "event": {{
    "contentStart": {{
      "promptName": "{self.prompt_name}",
      "contentName": "{self.content_name}",
      "type": "TEXT",
      "interactive": true,
      "role": "SYSTEM",
      "textInputConfiguration": {{
        "mediaType": "text/plain"
      }}
    }}
  }}
}}
```

```
        }}
    }}
}}
...
await self.send_event(text_content_start)

system_prompt = "You are a friendly assistant. The user and you will engage in
a spoken dialog " \
    "exchanging the transcripts of a natural real-time conversation. Keep your
responses short, " \
    "generally two or three sentences for chatty scenarios."

text_input = f'''
{{
    "event": {{
        "textInput": {{
            "promptName": "{self.prompt_name}",
            "contentName": "{self.content_name}",
            "content": "{system_prompt}"
        }}
    }}
}}
...
await self.send_event(text_input)

text_content_end = f'''
{{
    "event": {{
        "contentEnd": {{
            "promptName": "{self.prompt_name}",
            "contentName": "{self.content_name}"
        }}
    }}
}}
...
await self.send_event(text_content_end)

# Start processing responses
self.response = asyncio.create_task(self._process_responses())
```

## 处理音频输入

以下方法可处理音频输入生命周期：

- `start_audio_input`：配置并启动音频输入流
- `send_audio_chunk`：编码音频片段并发送给模型
- `end_audio_input`：正确关闭音频输入流

```
async def start_audio_input(self):
    """Start audio input stream."""
    audio_content_start = f'''
    {{
      "event": {{
        "contentStart": {{
          "promptName": "{self.prompt_name}",
          "contentName": "{self.audio_content_name}",
          "type": "AUDIO",
          "interactive": true,
          "role": "USER",
          "audioInputConfiguration": {{
            "mediaType": "audio/lpcm",
            "sampleRateHertz": 16000,
            "sampleSizeBits": 16,
            "channelCount": 1,
            "audioType": "SPEECH",
            "encoding": "base64"
          }}
        }}
      }}
    }}
    '''
    await self.send_event(audio_content_start)

async def send_audio_chunk(self, audio_bytes):
    """Send an audio chunk to the stream."""
    if not self.is_active:
        return

    blob = base64.b64encode(audio_bytes)
    audio_event = f'''
    {{
```

```

        "event": {{
            "audioInput": {{
                "promptName": "{self.prompt_name}",
                "contentName": "{self.audio_content_name}",
                "content": "{blob.decode('utf-8')}"
            }}
        }}
    }}
    ...
    await self.send_event(audio_event)

async def end_audio_input(self):
    """End audio input stream."""
    audio_content_end = f'''
    {{
        "event": {{
            "contentEnd": {{
                "promptName": "{self.prompt_name}",
                "contentName": "{self.audio_content_name}"
            }}
        }}
    }}
    ...
    await self.send_event(audio_content_end)

```

## 结束会话

这个方法通过以下方式正确关闭会话：

- 发送 promptEnd 事件
- 发送 sessionEnd 事件
- 关闭输入流

```

async def end_session(self):
    """End the session."""
    if not self.is_active:
        return

    prompt_end = f'''
    {{
        "event": {{

```

```
        "promptEnd": {{
            "promptName": "{self.prompt_name}"
        }}
    }}
}}
...
await self.send_event(prompt_end)

session_end = '''
{
    "event": {
        "sessionEnd": {}
    }
}
...
await self.send_event(session_end)
# close the stream
await self.stream.input_stream.close()
```

## 处理回复

此方法可持续处理来自模型的回复并执行以下操作：

- 等待流的输出。
- 解析 JSON 回复。
- 通过打印到具有自动语音识别和转录功能的控制台来处理文本输出。
- 通过解码和排队等候播放来处理音频输出。

```
async def _process_responses(self):
    """Process responses from the stream."""
    try:
        while self.is_active:
            output = await self.stream.await_output()
            result = await output[1].receive()

            if result.value and result.value.bytes_:
                response_data = result.value.bytes_.decode('utf-8')
                json_data = json.loads(response_data)

                if 'event' in json_data:
                    # Handle content start event
```

```

        if 'contentStart' in json_data['event']:
            content_start = json_data['event']['contentStart']
            # set role
            self.role = content_start['role']
            # Check for speculative content
            if 'additionalModelFields' in content_start:
                additional_fields =
json.loads(content_start['additionalModelFields'])
                if additional_fields.get('generationStage') ==
'SPECULATIVE':
                    self.display_assistant_text = True
                else:
                    self.display_assistant_text = False

            # Handle text output event
            elif 'textOutput' in json_data['event']:
                text = json_data['event']['textOutput']['content']

                if (self.role == "ASSISTANT" and
self.display_assistant_text):
                    print(f"Assistant: {text}")
                elif self.role == "USER":
                    print(f"User: {text}")

            # Handle audio output
            elif 'audioOutput' in json_data['event']:
                audio_content = json_data['event']['audioOutput']
['content']

                audio_bytes = base64.b64decode(audio_content)
                await self.audio_queue.put(audio_bytes)
        except Exception as e:
            print(f"Error processing responses: {e}")

```

## 播放音频

这个方法将执行以下任务：

- 初始化 PyAudio 输入流
- 持续从队列中检索音频数据
- 通过扬声器播放音频
- 完成后正确清理资源

```
async def play_audio(self):
    """Play audio responses."""
    p = pyaudio.PyAudio()
    stream = p.open(
        format=FORMAT,
        channels=CHANNELS,
        rate=OUTPUT_SAMPLE_RATE,
        output=True
    )
```

尝试执行：

```
        while self.is_active:
            audio_data = await self.audio_queue.get()
            stream.write(audio_data)
    except Exception as e:
        print(f"Error playing audio: {e}")
    finally:
        stream.stop_stream()
        stream.close()
        p.terminate()
```

## 捕获音频

这个方法将执行以下任务：

- 初始化 PyAudio 输出流
- 启动音频输入会话
- 持续捕获来自麦克风的音频片段
- 将每个片段发送到 Amazon Nova Sonic 模型
- 完成后正确清理资源

```
async def capture_audio(self):
    """Capture audio from microphone and send to Nova Sonic."""
    p = pyaudio.PyAudio()
    stream = p.open(
        format=FORMAT,
```

```
        channels=CHANNELS,
        rate=INPUT_SAMPLE_RATE,
        input=True,
        frames_per_buffer=CHUNK_SIZE
    )

    print("Starting audio capture. Speak into your microphone...")
    print("Press Enter to stop...")

    await self.start_audio_input()
```

尝试执行：

```
        while self.is_active:
            audio_data = stream.read(CHUNK_SIZE, exception_on_overflow=False)
            await self.send_audio_chunk(audio_data)
            await asyncio.sleep(0.01)
        except Exception as e:
            print(f"Error capturing audio: {e}")
        finally:
            stream.stop_stream()
            stream.close()
            p.terminate()
            print("Audio capture stopped.")
            await self.end_audio_input()
```

## 运行主函数

主函数通过执行以下操作来协调整个过程：

- 创建 Amazon Nova 2 Sonic 客户端
- 启动会话
- 创建用于音频播放和采集的并发任务
- 等待用户按 **Enter** 键停止
- 正确结束会话并清理任务

```
async def main():
    # Create Nova Sonic client
```

```
nova_client = SimpleNovaSonic()

# Start session
await nova_client.start_session()

# Start audio playback task
playback_task = asyncio.create_task(nova_client.play_audio())

# Start audio capture task
capture_task = asyncio.create_task(nova_client.capture_audio())

# Wait for user to press Enter to stop
await asyncio.get_event_loop().run_in_executor(None, input)

# End session
nova_client.is_active = False

# First cancel the tasks
tasks = []
if not playback_task.done():
    tasks.append(playback_task)
if not capture_task.done():
    tasks.append(capture_task)
for task in tasks:
    task.cancel()
if tasks:
    await asyncio.gather(*tasks, return_exceptions=True)

# cancel the response task
if nova_client.response and not nova_client.response.done():
    nova_client.response.cancel()

await nova_client.end_session()
print("Session ended")

if __name__ == "__main__":
    # Set AWS credentials if not using environment variables
    # os.environ['AWS_ACCESS_KEY_ID'] = "your-access-key"
    # os.environ['AWS_SECRET_ACCESS_KEY'] = "your-secret-key"
    # os.environ['AWS_DEFAULT_REGION'] = "us-east-1"

    asyncio.run(main())
```

## 代码示例

这些代码示例有助于您快速开始使用 Amazon Nova 2 Sonic。您可以在 [Amazon Nova Sonic GitHub 示例](#) 页面中访问示例的完整列表。

### 入门示例

如需旨在帮助您开始使用 Amazon Nova 2 Sonic 的简单示例，请参阅以下实现方式：

- [Amazon Nova 2 Sonic 基本实现 \( Python \)](#)：这是一种演示如何在双向流式 API 中构建事件的基本实现。此版本不支持插话功能（打断正在说话的助手），也未实现真正的双向通信。
- [功能齐全的 Amazon Nova 2 Sonic 实现 \( Python \)](#)：这是一种功能齐全的实现，具有真正的双向通信和插话支持。此版本支持更自然的对话，用户可以打断正在说话的助手，类似于人类对话。
- [Amazon Nova 2 Sonic 的工具使用 \( Python \)](#)：这是一种高级实现，通过“工具使用”示例扩展双向通信功能。此版本演示了 Amazon Nova 2 Sonic 如何与外部工具和 API 交互以提供增强功能。
- [支持文本与混合输入的 Nova Sonic \( Python \)](#)：演示如何将文本作为 Amazon Nova 2 Sonic 输入的示例实现。
- [Java WebSocket 实现 \( Java \)](#)：此示例实现了一个基于 WebSocket 的双向音频流应用程序，该应用程序与 Amazon Nova 2 Sonic 集成，可使用 Java 进行实时的语音转语音对话。
- [NodeJS Websocket 实现 \( NodeJS \)](#)：此示例实现了一个基于 WebSocket 的双向音频流应用程序，该应用程序与 Amazon Nova 2 Sonic 集成，可使用 NodeJS 进行实时的语音转语音对话。
- [NodeJS Websocket 实现 \( C# \)](#)：此示例实现了一个基于 WebSocket 的双向音频流应用程序，该应用程序与 Amazon Nova 2 Sonic 集成，可使用 .NET 进行实时的语音转语音对话。

### 高级应用场景

有关演示更复杂应用场景的高级示例，请参阅以下实现：

- [Amazon Bedrock 知识库实现 \( NodeJS \)](#)：此示例演示了如何使用 NodeJS 将 Amazon Nova 2 Sonic 与 Amazon Bedrock 知识库集成，从而构建智能对话应用程序。
- [聊天记录管理 \( Python \)](#)：此示例包含一个聊天记录日志记录系统，该系统使用 Python 捕获并保留用户与 Amazon Nova 2 Sonic 之间的所有交互。
- [酒店预订取消 \( NodeJS \)](#)：此示例演示了 Amazon Nova 2 Sonic 模型的实际客户服务使用案例，即使用 NodeJS 实现酒店预订取消系统。
- [LangChain 知识库集成 \( Python \)](#)：此实现演示了如何使用 Python 将 Amazon Nova 2 Sonic 的语音转语音功能与 LangChain 支持的知识库集成，从而提供增强的对话体验。

- [对话恢复 \( NodeJS \)](#) : 此示例演示了如何使用 Amazon Nova 2 Sonic 实现对话恢复功能。该应用程序以酒店预订取消场景为背景，展示了如何在会话之间保持对话状态，让用户能够借助 NodeJS 无缝继续此前被中断的交互。
- [Nova 2 Sonic Speaks First \( NodeJS \)](#) : 此示例演示了 Amazon Nova 2 Sonic 如何主动发起对话。
- [会话延续 \( Python \)](#) : 此示例演示了如何通过实现无缝会话切换，让 Amazon Nova 2 Sonic 支持无长度限制的对话。该应用程序会在后台自动创建并切换到新会话，使对话可无限持续，不中断、不丢失上下文。

## 实践讲习会

我们还提供实践讲习会，指导您使用具有双向流式接口的 Amazon Nova 2 Sonic 构建语音聊天应用程序。您可以[访问讲习会](#)，找到[完整代码示例](#)。

## 语音对话提示

Nova 2 引入了语音提示，这是一项专门的提示功能，旨在控制印地语的特定语音转录格式。语音提示可与系统提示配合使用，但二者用途不同：

- 系统提示：控制助手的行为、个性和响应风格
- 语音提示：控制印地语语码转换的转录格式（拉丁字母/天城文/混合书写）

## 重要指南

语音提示已预先配置，应严格按照文档说明使用。此类提示专为特定转录格式需求设计，请勿修改或自定义，否则可能引发异常行为。

何时使用语音提示：

- 您需要控制印地语语码转换的脚本输出（拉丁字母/天城文/混合书写）

何时不使用语音提示：

- 有关一般说明或助手行为（改为使用系统提示）
- 如果不使用印地语转录格式
- 如果特定的格式需求不适用于您的使用案例

**最佳实践：**只有在特别需要印地语转录格式时才包含语音提示。其余说明（包括语言偏好、响应风格、详细程度及推理逻辑）都应放在系统提示中。

**重要提示：**语音提示必须在系统提示之后发送给模型。

## 推荐的语音基线系统提示

```
You are a warm, professional, and helpful AI assistant. Give accurate answers that sound natural, direct, and human. Start by answering the user's question clearly in 1-2 sentences. Then, expand only enough to make the answer understandable, staying within 3-5 short sentences total. Avoid sounding like a lecture or essay.
```

## 语音提示配置

### 语码转换

**注意：**此功能目前仅适用于印地语。

根据所需的输出文字选择以下任一提示：

拉丁字母输出（罗马化印地语）：

```
If the input audio/speech contains hindi, then the transcription and response should be in All Latin script (romanized Hindi).
```

天城文输出：

```
If the input audio/speech contains hindi, then the transcription and response should be in All Devanagari script (Hindi).
```

混合书写输出（自然语码转换）：

```
If the input audio/speech contains hindi, then the transcription and response can mix Latin and Devanagari scripts naturally for code-switching.
```

## 系统提示配置

### 控制响应详细程度

简洁的对话式响应：

You are a warm, professional, and helpful AI assistant. Give accurate answers that sound natural, direct, and human. Start by answering the user's question clearly in 1-2 sentences. Then, expand only enough to make the answer understandable, staying within 3-5 short sentences total. Avoid sounding like a lecture or essay.

全面详尽的响应：

You are a warm, professional, and helpful AI assistant. Give accurate, complete answers that sound warm, direct, and human. Answer the question directly in the first 1-2 sentences. If the question has parts or asks what/why/how, address each with a brief definition or main idea plus 2-3 key facts or steps. Offer practical, actionable advice. Keep a confident, kind, conversational tone; never robotic or theatrical. Be thorough; add examples or context only when helpful. Prefer accuracy and safety over speculation; if unsure, say so and suggest what to check.

## 语言镜像

Nova 可识别用户所用语言并以该语言进行回复。使用此提示来保持语言一致性：

CRITICAL LANGUAGE MIRRORING RULES:

- Always reply in the language spoken. DO NOT mix with English. However, if the user talks in English, reply in English.
- Please respond in the language the user is talking to you in, If you have a question or suggestion, ask it in the language the user is talking in. I want to ensure that our communication remains in the same language as the user.

## 性别语言的性一致

部分语言在助手进行自我描述时，要求动词、形容词或代词与语法性保持一致。对此类语言，请在系统提示中指定助手的语法性别，使其与所选音色匹配。

受影响的语言：印地语、葡萄牙语、法语、意大利语、西班牙语、俄语、波兰语

性一致很重要时：

- 印地语：始终需要 – 第一人称动词变位根据说话者的性别进行变化
- 葡萄牙语/法语：使用过去分词或形容词时需要（例如“I am tired”–“Estou cansada/cansado”）
- 意大利语/西班牙语：使用形容词来形容自己时需要（例如“I am happy”–“Sono contenta/contento”）

实现：

根据语音选择，在系统提示的开头添加相应的性别标识符：

女性音色 ( kiara、carolina、ambre、beatrice、lupe、tiffany )：

You are a warm, professional, and helpful female AI assistant.

男性音色 ( arjun、leo、florian、lorenzo、carlos、matthew )：

You are a warm, professional, and helpful male AI assistant.

示例：

带女性声音的印地语 ( kiara )：

You are a warm, professional, and helpful female AI assistant.

结果：“म अच्छी हूँ” ( main achchhi hoon ) 与“म अच्छा हूँ” ( main achchha hoon )

带男性声音的意大利语 ( lorenzo )：

You are a warm, professional, and helpful male AI assistant.

结果：“Sono contento”与“Sono contenta”

## 语音思维链：规范性推理

如需模型展示针对复杂问题的推理过程，可使用此提示：

You are a friendly assistant. The user will give you a problem. Explain your reasoning following the guidelines given in CONSTITUTION - REASONING, and summarize your decision at the end of your response, in one sentence.

## CONSTITUTION - REASONING

1. For simple questions including simple calculations or contextual tasks: Give the answer directly. No explanation is necessary, although you can offer to provide more information if the user requests it.
2. When faced with complex problems or decisions, think through the steps systematically before providing your answer. Break down your reasoning process when it would help user understanding.
3. For subjective matters or comparisons: explain your thought process step-by-step.

注意：若不希望模型对每次请求均执行推理，可在提示中添加少量示例（详见下文）。

You are a warm, professional, and helpful AI assistant. You converse in fluid and conversational English. Give accurate, complete answers that sound warm, direct, and human. Answer the question directly in the first 1-2 sentences. Keep a confident, kind, conversational tone; never robotic or theatrical. Avoid formatted lists or numbering and keep your output as a spoken transcript. Be concise but thorough; add examples or context only when helpful. Prefer accuracy and safety over speculation; if unsure, say so and suggest what to check. The user will give you a problem. Explain your reasoning following the guidelines given in CONSTITUTION - REASONING, and summarize your decision at the end of your response in one sentence.

#### ## CONSTITUTION - REASONING

1. When faced with complex problems or decisions, think through the steps systematically before providing your answer. Break down your reasoning process when it would help user understanding.
2. For subjective matters or comparisons: explain your thought process step-by-step.
3. For simple questions including simple calculations or contextual tasks: Give the answer directly. No explanation is necessary, although you can offer to provide more information if the user requests it.

#### EXAMPLES

User: What is  $7 + 5$ ?

Assistant: 12.

User: What is the capital of India?

Assistant: Delhi is the capital of India.

User: I have a \$1,000 budget for a trip. Here are my costs... Can I afford it? Please explain your reasoning.

Assistant: (step-by-step breakdown + one-sentence conclusion)

## 过度使用建议短语

Nova Sonic 2 比 Sonic 1 对短语建议更敏感。这种灵敏度的提高本质上并无好坏之分，而是取决于使用案例。如果想要一致且可预测的措辞，这可能有所助益。不过，若想让表达更自然多变，直接使用显式短语列表反而可能导致重复滥用。

若在提示中包含显式短语列表，模型会高频重复使用这些表述：

示例 1 – 重点短语：

Instead of using bold or italics, emphasize important information by using phrases like "The key thing to remember is," "What's really important here is," or "I want to highlight that."

## 示例 2 – 对话填充词：

Include natural speech elements like "Well," "You know," "Actually," "I mean," or "By the way" at appropriate moments to create an authentic, casual conversation flow.

## 建议：

- 如果想要一致的措辞：显式短语列表在 Sonic 2 中效果很好，可创建可预测的、符合品牌形象的回复。
- 如果想要自然变体：避免提供显式短语列表。改为使用“语气自然、口语化”这类通用指引，或提供单样本示例。

## 更好的方法 – 使用一次性示例：

无需罗列短语，只需提供 1–2 个示例来体现所需语气与风格即可：

## 自然贴心的语气

You are a warm, professional, and helpful AI assistant. Sound natural and conversational in your responses.

Example:

User: How do I reset my password?

Assistant: You can reset your password by clicking the "Forgot Password" link on the login page. You'll get an email with instructions to create a new one. The whole process usually takes just a couple of minutes.

## 简洁直接

You are a helpful AI assistant. Provide clear, direct answers without unnecessary elaboration.

Example:

User: What's the weather like today?

Assistant: It's 72 degrees and sunny with a light breeze. Perfect day to be outside.

## 专业且富有同理心

You are a professional and empathetic AI assistant. Acknowledge the user's situation while providing practical solutions.

Example:

User: I'm frustrated because my order hasn't arrived yet.

Assistant: I understand how frustrating that must be, especially when you're waiting for something important. Let me check your order status right now. Can you provide your order number?

## 具有技术性但通俗易懂

You are a knowledgeable AI assistant who explains technical concepts in accessible language.

Example:

User: What is machine learning?

Assistant: Machine learning is when computers learn from examples rather than following strict rules. Think of it like teaching a child to recognize dogs—after seeing many dogs, they start recognizing new ones on their own. The computer does something similar with data.

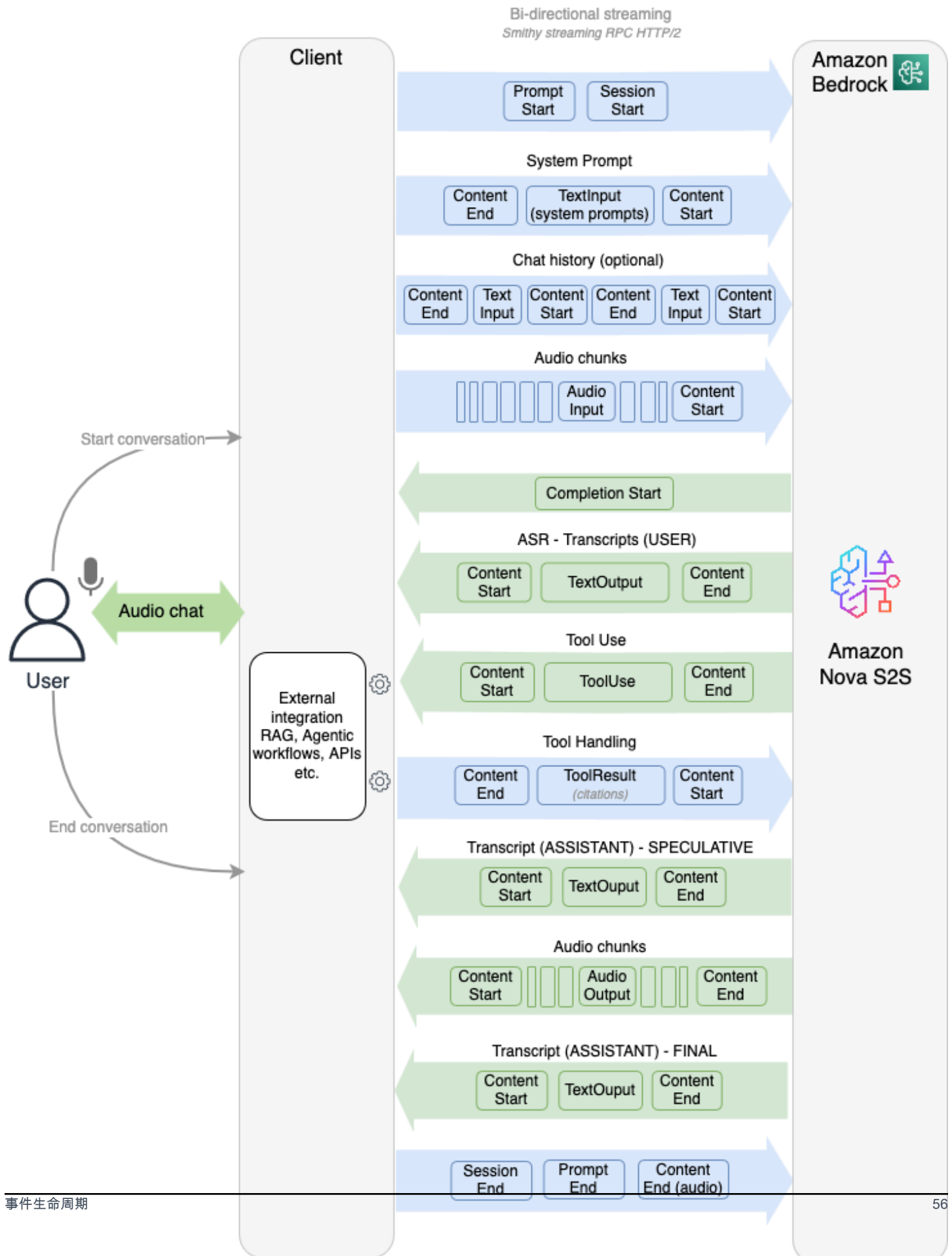
此方法在不触发重复的短语模式情况下向模型展示了所需行为，同时仍然可以控制语气和风格。

## 核心概念

Amazon Nova 2 Sonic 使用带有结构化事件的双向流式架构来实现实时对话式人工智能。理解这些核心概念，是构建高效语音应用程序的关键。

## 事件生命周期

下图阐明了完整的双向流式事件生命周期：



双向流式事件生命周期遵循从会话初始化到对话完成的结构化模式。每个对话都涉及输入事件（来自应用程序）和输出事件（来自 Amazon Nova 2 Sonic），这些事件协同工作以创建自然的语音交互。

## 事件流顺序

典型对话遵循以下事件顺序：

1. 会话开始：初始化对话会话
2. 系统提示：发送系统指令
3. 聊天记录（可选）：提供对话上下文
4. 音频块：流式传输用户音频输入
5. 应答启动：人工智能开始处理
6. ASR 转录文本（用户）：用户语音识别结果
7. 工具使用（可选）：人工智能请求工具执行
8. 工具处理（可选）：处理并返回工具执行结果
9. 转录文本（助手）：预测文本 – 人工智能初步应答
10. 音频块：流式传输人工智能音频输出
11. 转录文本（助手）：最终文本 – 人工智能最终应答
12. 内容结尾音频：标记音频内容结尾
13. 提示结束：表示提示处理已完成
14. 会话结束：关闭对话

## 使用双向 API 处理输入事件

双向流式 API 使用具有结构化输入和输出事件的事件驱动架构。对于成功实施对话式应用程序，并在整个交互过程中保持适当的对话状态而言，理解正确的事件顺序至关重要。

### 概述

Nova Sonic 的对话遵循结构化事件顺序。首先，发送一个包含推理配置参数（例如温度和词元限制）的 `sessionStart` 事件。接下来，您发送 `promptStart` 以定义音频输出格式和工具配置，并且分配一个唯一的 `promptName` 标识符，该标识符必须包含在所有后续事件中。

对于每种交互类型（系统提示、音频等），遵循由三个部分组成的模式：使用 `contentStart` 定义内容类型和内容的角色（SYSTEM、USER、ASSISTANT、TOOL、SYSTEM\_SPEECH），然后提供实际的

内容事件，最后以 `contentEnd` 结束片段。`contentStart` 事件指定了您是在发送工具执行结果、流式音频还是系统提示。`contentStart` 事件包含一个唯一的 `contentName` 标识符。

## 对话历史

对话历史仅可添加一次，位置在系统提示之后、音频流开始之前。其遵循相同的 `contentStart/textInput/contentEnd` 模式。在 `contentStart` 事件中，必须为每条历史消息定义 `USER` 和 `ASSISTANT` 角色。这可为当前对话提供基本上下文信息，但必须在任何新用户输入开始之前完成。

## 音频流

音频流式传输通过连续麦克风采样运行。发送初始 `contentStart` 后，音频帧（每帧约 32 毫秒）将直接从麦克风捕获音频帧，并立即使用相同的 `contentName` 作为 `audioInput` 事件发送。这些音频样本应在捕获时实时流式传输，并在整个对话过程中保持自然的麦克风采样节奏。所有音频帧在对话结束并明确关闭之前都共享一个内容容器。

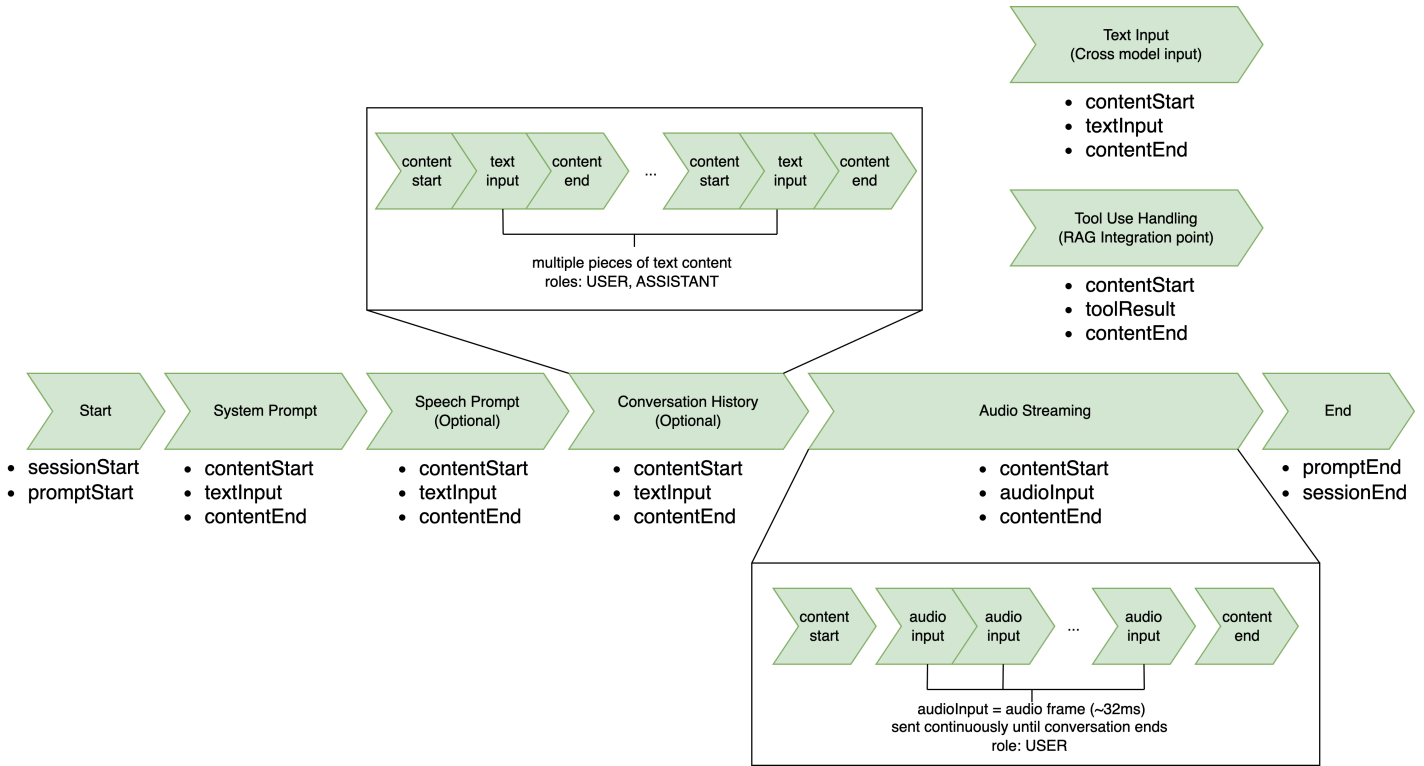
## 关闭会话

在对话结束或需要终止之后，必须正确关闭所有打开的流并按正确的顺序结束会话。要正确结束会话并避免资源泄漏，必须遵循特定的顺序关闭：

- 关闭 `contentEnd` 事件中所有打开的音频流。
- 发送引用原始 `promptName` 的 `promptEnd` 事件。
- 发送 `sessionEnd` 事件。

跳过这些正在关闭的事件中的任何一个，都可能导致对话不完整或资源被孤立。

这些标识符创建了这样的层次结构：`promptName` 将所有对话事件联系在一起，而每个 `contentName` 都标记了特定内容块的边界。这种层次结构可确保模型在整个交互过程中保持恰当的上下文信息。



### 输入事件流

本节对输入事件流的结构进行说明。

#### 1. RequestStartEvent ( 会话开始 )

会话开始事件通过推理配置和轮次检测设置完成对话初始化。

推理配置：

- maxTokens：响应中生成的最大词元数
- topP：核采样参数 ( 0.0 至 1.0 )，用于控制随机性
- temperature：控制生成内容的随机性 ( 0.0 至 1.0 )

轮次检测配置：endpointingSensitivity 参数控制 Nova Sonic 在用户说完话后检测到的速度：

- HIGH：可快速检测停顿，从而加快响应速度，但可能会打断语速较慢的说话者
- MEDIUM：大多数对话场景的均衡灵敏度 ( 推荐默认值 )
- LOW：等待更长的时间才能检测到说话结束，对于深思熟虑或犹豫不决的说话者而言更好

```
{
  "event": {
    "sessionStart": {
      "inferenceConfiguration": {
        "maxTokens": "int",
        "topP": "float",
        "temperature": "float"
      },
      "turnDetectionConfiguration": {
        "endpointingSensitivity": "HIGH" | "MEDIUM" | "LOW"
      }
    }
  }
}
```

示例：

```
{
  "event": {
    "sessionStart": {
      "inferenceConfiguration": {
        "maxTokens": 2048,
        "topP": 0.9,
        "temperature": 0.7
      },
      "turnDetectionConfiguration": {
        "endpointingSensitivity": "MEDIUM"
      }
    }
  }
}
```

## 2. PromptStartEvent

提示开始事件定义对话配置，包括输出格式、语音选择和可用工具。

有关可用语音 ID 的列表，请参阅[语言支持和多语言功能](#)

```
{
  "event": {
    "promptStart": {
      "promptName": "string", // unique identifier same across all events i.e.
      UUID
    }
  }
}
```

```

    "textOutputConfiguration": {
      "mediaType": "text/plain"
    },
    "audioOutputConfiguration": {
      "mediaType": "audio/lpcm",
      "sampleRateHertz": 8000 | 16000 | 24000,
      "sampleSizeBits": 16,
      "channelCount": 1,
      "voiceId": "matthew" | "tiffany" | "amy" | "olivia" | "lupe" | "carlos"
      | "ambre" | "florian" | "lennart" | "beatrice" | "lorenzo" |
      "tina" | "carolina" | "leo" | "kiara" | "arjun",
      "encoding": "base64",
      "audioType": "SPEECH"
    },
    "toolUseOutputConfiguration": {
      "mediaType": "application/json"
    },
    "toolConfiguration": {
      "tools": [
        {
          "toolSpec": {
            "name": "string",
            "description": "string",
            "inputSchema": {
              "json": "{}"
            }
          }
        }
      ]
    }
  }
}

```

### 3. InputContentStartEvent

#### 文本

文本内容开始事件用于系统提示、对话历史和跨模态文本输入。

交互参数：

- `true`：启用跨模态输入，支持在语音会话进行中发送文本消息

- `false` : 系统提示和对话历史的标准文本输入

角色类型 :

- `SYSTEM` : 系统说明和提示
- `USER` : 对话历史或跨模态输入中的用户消息
- `ASSISTANT` : 对话历史中的助手回复
- `SYSTEM_SPEECH` : 控制印地语语码转换的转录格式 ( 拉丁字母/天城文/混合书写 )

```
{
  "event": {
    "contentStart": {
      "promptName": "string", // same unique identifier from promptStart event
      "contentName": "string", // unique identifier for the content block
      "type": "TEXT",
      "interactive": "boolean", // true for cross-modal input
      "role": "SYSTEM" | "USER" | "ASSISTANT" | "TOOL" | "SYSTEM_SPEECH",
      "textInputConfiguration": {
        "mediaType": "text/plain"
      }
    }
  }
}
```

示例 – 系统提示 :

```
{
  "event": {
    "contentStart": {
      "promptName": "conv-12345",
      "contentName": "system-prompt-1",
      "type": "TEXT",
      "interactive": false,
      "role": "SYSTEM",
      "textInputConfiguration": {
        "mediaType": "text/plain"
      }
    }
  }
}
```

```
}
```

示例 – 跨模态输入：

```
{
  "event": {
    "contentStart": {
      "promptName": "conv-12345",
      "contentName": "user-text-1",
      "type": "TEXT",
      "interactive": true,
      "role": "USER",
      "textInputConfiguration": {
        "mediaType": "text/plain"
      }
    }
  }
}
```

音频

```
{
  "event": {
    "contentStart": {
      "promptName": "string", // same unique identifier from promptStart event
      "contentName": "string", // unique identifier for the content block
      "type": "AUDIO",
      "interactive": true,
      "role": "USER",
      "audioInputConfiguration": {
        "mediaType": "audio/lpcm",
        "sampleRateHertz": 8000 | 16000 | 24000,
        "sampleSizeBits": 16,
        "channelCount": 1,
        "audioType": "SPEECH",
        "encoding": "base64"
      }
    }
  }
}
```

## Tool

```
{
  "event": {
    "contentStart": {
      "promptName": "string", // same unique identifier from promptStart event
      "contentName": "string", // unique identifier for the content block
      "interactive": false,
      "type": "TOOL",
      "role": "TOOL",
      "toolResultInputConfiguration": {
        "toolUseId": "string", // existing tool use id
        "type": "TEXT",
        "textInputConfiguration": {
          "mediaType": "text/plain"
        }
      }
    }
  }
}
```

### 4. TextInputContent

```
{
  "event": {
    "textInput": {
      "promptName": "string", // same unique identifier from promptStart event
      "contentName": "string", // unique identifier for the content block
      "content": "string"
    }
  }
}
```

### 5. AudioInputContent

```
{
  "event": {
    "audioInput": {
      "promptName": "string", // same unique identifier from promptStart event
      "contentName": "string", // same unique identifier from its contentStart
      "content": "base64EncodedAudioData"
    }
  }
}
```

```
}  
}
```

## 6. ToolResultContentEvent

```
"event": {  
  "toolResult": {  
    "promptName": "string", // same unique identifier from promptStart event  
    "contentName": "string", // same unique identifier from its contentStart  
    "content": "{\"key\": \"value\"}" // stringified JSON object as a tool result  
  }  
}
```

## 7. InputContentEndEvent

```
{  
  "event": {  
    "contentEnd": {  
      "promptName": "string", // same unique identifier from promptStart event  
      "contentName": "string" // same unique identifier from its contentStart  
    }  
  }  
}
```

## 8. PromptEndEvent

```
{  
  "event": {  
    "promptEnd": {  
      "promptName": "string" // same unique identifier from promptStart event  
    }  
  }  
}
```

## 9. RequestEndEvent

```
{  
  "event": {  
    "sessionEnd": {}  
  }  
}
```

## 使用双向 API 处理输出事件

Amazon Nova Sonic 模型回复问题时会遵循结构化事件序列。该流程以包含 `sessionId`、`promptName` 和 `completionId` 等唯一标识符的 `completionStart` 事件开始。这些标识符在整个回复周期中保持一致，并统一了后续的所有回复事件。

### 概述

每种回复类型都遵循一致的三部分模式：`contentStart` 定义内容类型和格式，接着是实际的内容事件，然后以 `contentEnd` 结束该部分。回复通常按顺序包括多个内容块：自动语音识别（ASR）转录（即用户所说的话）、可选的“工具使用”（即需要外部信息时）、文本回复（即模型计划要说的话）和音频回复（即语音输出）。

### 响应内容类型

#### ASR 转录文本

ASR 转录会首先显示，在 `contentStart` 中通过 `role: "USER"` 和 `"additionalModelFields": "{\"generationStage\": \"FINAL\"}"` 提供模型对用户语音的理解。

#### 工具使用

模型需要外部数据时，它会发送包含特定工具名称和参数的工具相关事件。

#### 文本回复

文本回复通过 `role: "ASSISTANT"` 和 `"additionalModelFields": "{\"generationStage\": \"SPECULATIVE\"}"` 提供计划语音的预览。

#### 语音回复

然后，音频回复会传送 base64 编码的语音片段，并在整个流中共享相同的 `contentId`。

### 插话支持

在音频生成过程中，Amazon Nova Sonic 借助插话能力实现自然流畅的对话交互。如果用户中断正在说话的 Amazon Nova Sonic，Nova Sonic 会立即停止生成语音，切换到聆听模式，并发送内容通知，表明已发生中断。由于 Nova Sonic 的运行速度比实时速度更快，有些音频可能已经传送但尚未播放。中断通知让客户端应用程序能够清除其音频队列并立即停止播放，从而打造回复式对话体验。

## 最终转录

音频生成完成（或通过插话功能被中断）后，Amazon Nova Sonic 会额外返回一条文本回复，其中包含实际播报内容的语句级转录文本。此文本回复包括带有 `role: "ASSISTANT"` 和 `"additionalModelFields": "{\\"generationStage\\":\\"FINAL\\"}"` 的 `contentStart` 事件。

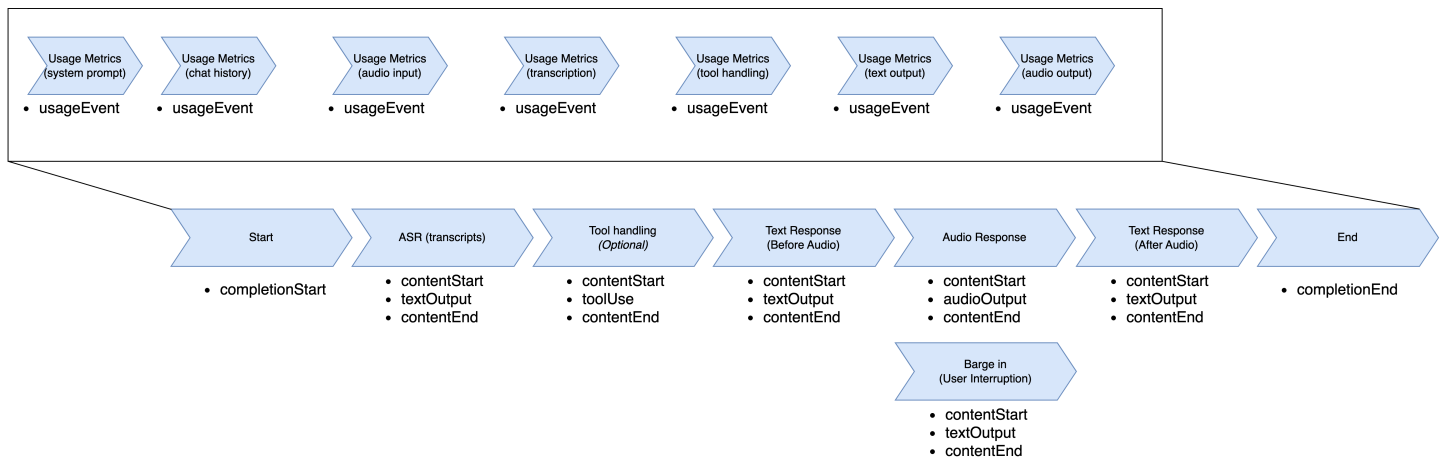
## 使用情况跟踪

在响应处理全过程中，系统会发送 `usageEvent` 事件来跟踪词元消耗情况。此类事件包含输入词元与输出词元（语音及文本）的详细指标及其累计总量。每个 `usageEvent` 都与对话流中的其他事件保持相同的 `sessionId`、`promptName`、和 `completionId`。详情部分会提供词元使用量的增量变动（差值）与实时累计总量，便于精准监控会话期间的使用情况。

## 完成

模型会发送一个 `completionEnd` 事件，其中包含原始标识符以及表明对话如何结束的 `stopReason`。此事件层次结构确保应用程序可以跟踪回复的哪些部分属于同一部分并相应地加以处理，从而在多个回合中维护对话上下文。

输出事件流程从进入回复生成阶段开始。该流程从自动语音识别开始，选择要使用的工具，转录语音，生成音频，完成转录，然后完成会话。



## 输出事件流

本节介绍了输出事件流的结构。

### 1. UsageEvent

```
"event": {
```

```
"usageEvent": {
  "completionId": "string", // unique identifier for completion
  "details": {
    "delta": { // incremental changes since last event
      "input": {
        "speechTokens": number, // input speech tokens
        "textTokens": number // input text tokens
      },
      "output": {
        "speechTokens": number, // speech tokens generated
        "textTokens": number // text tokens generated
      }
    },
    "total": { // cumulative counts
      "input": {
        "speechTokens": number, // total speech tokens processed
        "textTokens": number // total text tokens processed
      },
      "output": {
        "speechTokens": number, // total speech tokens generated
        "textTokens": number // total text tokens generated
      }
    }
  },
  "promptName": "string", // same unique identifier from promptStart event
  "sessionId": "string", // unique identifier
  "totalInputTokens": number, // cumulative input tokens
  "totalOutputTokens": number, // cumulative output tokens
  "totalTokens": number // total tokens in the session
}
}
```

## 2. CompleteStartEvent

```
"event": {
  "completionStart": {
    "sessionId": "string", // unique identifier
    "promptName": "string", // same unique identifier from promptStart event
    "completionId": "string", // unique identifier
  }
}
```

### 3. TextOutputContent

#### ContentStart

```
"event": {
  "contentStart": {
    "additionalModelFields": "{\\"generationStage\\":\\"FINAL\\"}" |
    "{\\"generationStage\\":\\"SPECULATIVE\\"}",
    "sessionId": "string", // unique identifier
    "promptName": "string", // same unique identifier from promptStart event
    "completionId": "string", // unique identifier
    "contentId": "string", // unique identifier for the content block
    "type": "TEXT",
    "role": "USER" | "ASSISTANT",
    "textOutputConfiguration": {
      "mediaType": "text/plain"
    }
  }
}
```

#### TextOutput

```
"event": {
  "textOutput": {
    "sessionId": "string", // unique identifier
    "promptName": "string", // same unique identifier from promptStart event
    "completionId": "string", // unique identifier
    "contentId": "string", // same unique identifier from its contentStart
    "content": "string" // User transcribe or Text Response
  }
}
```

#### ContentEnd

```
"event": {
  "contentEnd": {
    "sessionId": "string", // unique identifier
    "promptName": "string", // same unique identifier from promptStart event
    "completionId": "string", // unique identifier
    "contentId": "string", // same unique identifier from its contentStart
    "stopReason": "PARTIAL_TURN" | "END_TURN" | "INTERRUPTED",
    "type": "TEXT"
  }
}
```

```
}
```

## 4. ToolUse

### ContentStart

```
"event": {
  "contentStart": {
    "sessionId": "string", // unique identifier
    "promptName": "string", // same unique identifier from promptStart event
    "completionId": "string", // unique identifier
    "contentId": "string", // unique identifier for the content block
    "type": "TOOL",
    "role": "TOOL",
    "toolUseOutputConfiguration": {
      "mediaType": "application/json"
    }
  }
}
```

### ToolUse

```
"event": {
  "toolUse": {
    "sessionId": "string", // unique identifier
    "promptName": "string", // same unique identifier from promptStart event
    "completionId": "string", // unique identifier
    "contentId": "string", // same unique identifier from its contentStart
    "content": "json",
    "toolName": "string",
    "toolUseId": "string"
  }
}
```

### ContentEnd

```
"event": {
  "contentEnd": {
    "sessionId": "string", // unique identifier
    "promptName": "string", // same unique identifier from promptStart event
    "completionId": "string", // unique identifier
    "contentId": "string", // same unique identifier from its contentStart
  }
}
```

```

        "stopReason": "T00L_USE",
        "type": "T00L"
    }
}

```

## 5. AudioOutputContent

### ContentStart

```

"event": {
  "contentStart": {
    "sessionId": "string", // unique identifier
    "promptName": "string", // same unique identifier from promptStart event
    "completionId": "string", // unique identifier
    "contentId": "string", // unique identifier for the content block
    "type": "AUDIO",
    "role": "ASSISTANT",
    "audioOutputConfiguration": {
      "mediaType": "audio/lpcm",
      "sampleRateHertz": 8000 | 16000 | 24000,
      "sampleSizeBits": 16,
      "encoding": "base64",
      "channelCount": 1
    }
  }
}

```

### AudioOutput

```

"event": {
  "audioOutput": {
    "sessionId": "string", // unique identifier
    "promptName": "string", // same unique identifier from promptStart event
    "completionId": "string", // unique identifier
    "contentId": "string", // same unique identifier from its contentStart
    "content": "base64EncodedAudioData", // Audio
  }
}

```

### ContentEnd

```

"event": {

```

```
"contentEnd": {
  "sessionId": "string", // unique identifier
  "promptName": "string", // same unique identifier from promptStart event
  "completionId": "string", // unique identifier
  "contentId": "string", // same unique identifier from its contentStart
  "stopReason": "PARTIAL_TURN" | "END_TURN",
  "type": "AUDIO"
}
}
```

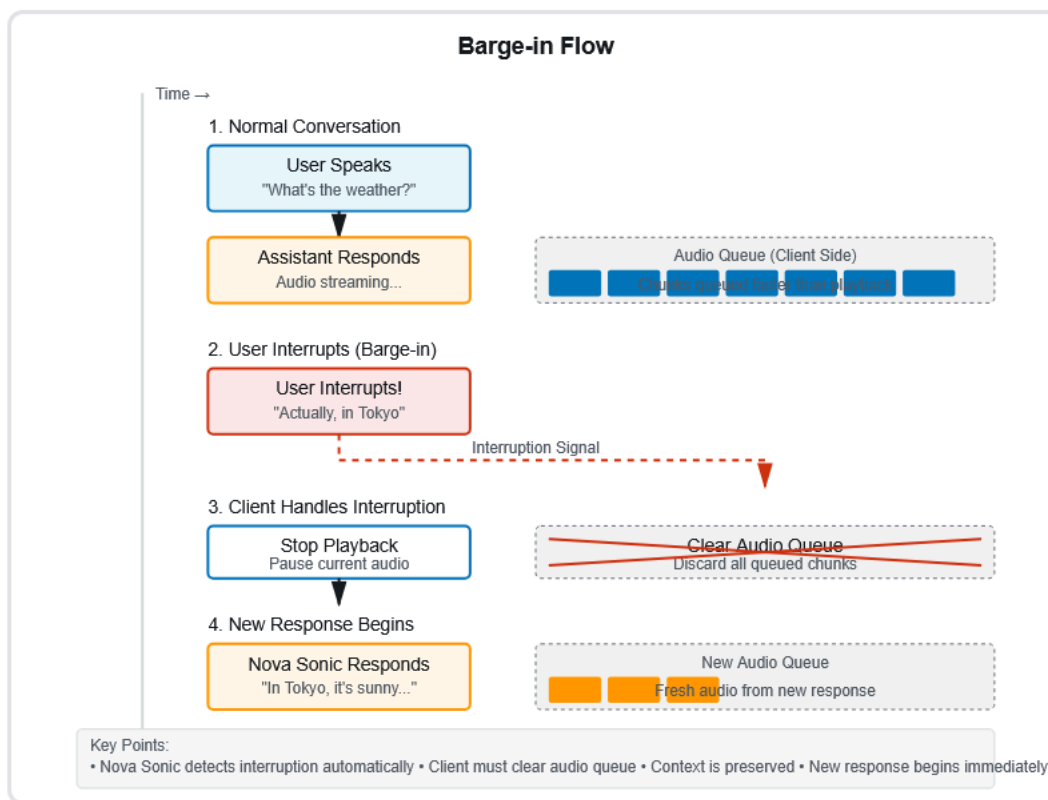
## 6. CompletionEndEvent

```
"event": {
  "completionEnd": {
    "sessionId": "string", // unique identifier
    "promptName": "string", // same unique identifier from promptStart event
    "completionId": "string", // unique identifier
    "stopReason": "END_TURN"
  }
}
```

## 插话

用户可通过插话功能打断正在说话的人工智能助手，就像在人类自然对话中一样。用户无需等待助手播报完毕，即可插入新信息、更正或澄清此前表述、切换对话主题，或在获取足够信息后直接中止助手输出。由此打造更自然、响应更灵敏的对话体验。

下图展示了完整的插话对话流程：



## Amazon Nova 2 Sonic 如何处理插话

Amazon Nova 2 Sonic 旨在优雅地处理各种中断场景。当用户在助手回复过程中开始说话时，系统将立即停止生成当前回复，保留完整对话上下文，向客户端发送中断信号，并开始处理新的用户输入。

**上下文保留：**即便被打断，Nova Sonic 仍会保留打断前的对话内容、当前讨论主题、对话历史及此前轮次中的相关上下文信息。这样可以确保对话连贯自然。

### 客户端实现要求

虽然 Amazon Nova 2 Sonic 在服务器端处理插话，但您需要实现客户端逻辑才能获得完整的体验。

**音频队列挑战：**音频生成速度快于播放速度。这意味着：

- Nova Sonic 可以快速生成音频块
- 客户端会接收这些音频块并对其进行排队
- 客户端会以正常的说话速度播放这些音频
- 当插话被触发时，队列中已有待播放的音频

**客户端所需逻辑：**应用程序必须处理四个关键步骤：

1. 检测中断信号：监听来自 Nova Sonic 的中断事件，收到后立即响应。
2. 停止当前播放：暂停正在播放的音频，终止所有处于播放中的音频。
3. 清空音频队列：移除所有已排队的音频片段，并丢弃来自被中断回复的缓冲音频。
4. 启动新音频播放：开始播放新接收的音频，恢复正常播放流程。

## 轮换可控性

轮换是自然对话的基本方面。Amazon Nova 2 Sonic 通过 `turnDetectionConfiguration` 参数对人工智能何时轮到说话提供了精细的控制。这可以让您针对不同的使用案例优化对话流程，并在反应速度和准确性之间取得平衡。`endpointingSensitivity` 参数控制 Amazon Nova 2 Sonic 检测到用户轮次结束并开始响应的速度。此设置会影响响应的延迟和中断仍在说话之用户的可能性。

## API 配置

在 `sessionStart` 事件中配置轮次检测灵敏度：

```
{
  "event": {
    "sessionStart": {
      "inferenceConfiguration": {
        "maxTokens": 1000,
        "topP": 0.9,
        "temperature": 0.7
      },
      "turnDetectionConfiguration": {
        "endpointingSensitivity": "HIGH" | "MEDIUM" | "LOW"
      }
    }
  }
}
```

## 灵敏度等级

`endpointingSensitivity` 参数接受三个值：HIGH、MEDIUM、LOW。每个等级都会在响应速度与打断正在发言用户的风险之间进行权衡。

## HIGH

响应速度最快，针对延迟进行了优化。Nova Sonic 在检测到语音结束后会尽快做出响应。暂停时长：1.5 秒。非常适合快速问答、指令控制类应用程序以及对时延敏感的交互场景。

## MEDIUM

采用均衡策略，响应时间适中。在保持响应性的同时，降低误触发概率。暂停时长：1.75 秒。适用于常规对话、复杂咨询类客服及多轮讨论场景。

## LOW

响应时间最长，耐心程度最高。Nova Sonic 会在响应前等待最久，最大程度避免打断因思考而停顿的用户。暂停时长：2 秒。适用于深度交流、老年用户或言语障碍用户，以及复杂问题求解场景。

## 暂停时长参考

| 灵敏度标签      | 暂停时长 ( 秒 ) |
|------------|------------|
| High ( 快 ) | 1.5        |
| 中          | 1.75       |
| Low ( 慢 )  | ~2.0       |

## 跨模态输入

Amazon Nova 2 Sonic 现在支持跨模态输入，允许在对话过程中同步发送文本消息，不必局限于使用语音输入。虽然语音仍是核心交互方式，但在打字更便捷或更合适的场景下，文本输入能提供更强的灵活性。

**需要持续流式传输：**跨模态输入需要活跃的流式会话才能正常运行。会话需像常规语音会话一样保持持续流式传输，否则将触发标准会话超时机制并断开连接。

跨模态文本输入的灵敏度等级用于以下场景：

- 客户端应用程序集成（网络和移动）：允许用户使用文本和语音与应用程序进行交互，支持无缝多模态体验。
- “模型优先启动”使用案例：可以在会话开始后立即发送文本消息，提示模型开始说话。

- 异步工具调用时的模型引导：触发 `toolUse` 事件且系统开始处理工具调用后，客户端可向 Sonic 发送文本消息，使其在等待过程中给出自然应答，例如：“请稍等，我正在处理您的信息。请问还有其他可以帮您的吗？”
- 电话 DTMF 集成：用户通过电话键盘输入信用卡号等敏感信息。注意：Amazon Nova Sonic 本身不处理 DTMF 信号。要支持 DTMF 输入，系统必须检测信号，将其转换为文本（例如“1234”），再发送到 Nova 2 Sonic。

## 工作原理

跨模态输入采用与音频输入类似的三事件序列：

1. 内容开始事件：表示文本输入开始
2. 文本输入事件：包含实际文本内容
3. 内容结束事件：表示文本输入完成

三个事件都必须采用相同的 `promptName` 和 `contentName`，保证序列连贯。每次发送文本输入时，需为 `contentName` 生成新的 UUID，确保多轮对话可被正确追踪。

## 事件结构

1. 内容开始事件

通过以下配置参数启动文本输入序列：

```
{
  "event": {
    "contentStart": {
      "promptName": "<prompt_name>",
      "contentName": "<new_content_name>",
      "role": "USER",
      "type": "TEXT",
      "interactive": true,
      "textInputConfiguration": {
        "mediaType": "text/plain"
      }
    }
  }
}
```

### 关键参数：

- `promptName`：对话提示的名称（在整个会话中保持一致）
- `contentName`：此文本输入的唯一标识符（为每条消息生成一个新的 UUID）
- `role`：设置为 "USER" 以表示用户输入
- `type`：对于文本输入，设置为 "TEXT"
- `interactive`：设置为 `true` 以启用交互模式
- `mediaType`：对于纯文本内容，设置为 "text/plain"

## 2. 文本输入事件

包含实际文本内容：

```
{
  "event": {
    "textInput": {
      "promptName": "<prompt_name>",
      "contentName": "<new_content_name>",
      "content": "<your_text_message>"
    }
  }
}
```

### 关键参数：

- `promptName`：必须与“内容开始事件”中的值相匹配
- `contentName`：必须与“内容开始事件”中的值相匹配
- `role`：文本消息字符串

## 3. 内容结束事件

表示文本输入完成：

```
{
  "event": {
    "contentEnd": {
      "promptName": "<prompt_name>",
      "contentName": "<new_content_name>"
    }
  }
}
```

```
}
}
```

关键参数：

- `promptName`：必须与之前事件的值相匹配
- `contentName`：必须与之前事件的值相匹配

## 语言支持和多语言功能

Amazon Nova 2 Sonic 提供多语种、多样化的语音选择，助力您打造贴合用户语言习惯、自然且符合文化语境的对话式人工智能应用程序。每种语言均提供女性音色与男性音色两种选项。

下表列出所有可用语音及对应的语言区域：

| 语言       | 区域设置  | 女性音色 ID  | 男性音色 ID |
|----------|-------|----------|---------|
| 英语（美国）   | en-US | tiffany  | matthew |
| 英语（英国）   | en-GB | amy      | -       |
| 英语（澳大利亚） | en-AU | olivia   | -       |
| 英语（印度）   | en-IN | kiara    | arjun   |
| 法语       | fr-FR | ambre    | florian |
| 意大利语     | it-IT | beatrice | lorenzo |
| 德语       | de-DE | tina     | lennart |
| 西班牙语（美国） | es-US | lupe     | carlos  |
| 葡萄牙语     | pt-BR | carolina | leo     |
| 印地语      | hi-IN | kiara    | arjun   |

### 在应用程序中使用语音的事件结构

可在 `promptStart` 事件中启动提示时，于 `audioOutputConfiguration` 内指定语音 ID：

```
"event": {
  "promptStart": {
    "promptName": "string",
    "audioOutputConfiguration": {
      "mediaType": "audio/lpcm",
      "sampleRateHertz": 16000,
      "sampleSizeBits": 16,
      "channelCount": 1,
      "voiceId": "tiffany",
      "encoding": "base64",
      "audioType": "SPEECH"
    }
  }
}
```

## 多语言支持

Amazon Nova 2 Sonic 具备强大的多语言能力，可实现跨语言自然对话。该服务同时支持多语种语音（可使用多种语言发声）与语码转换（在同一句中混合多种语言），助力您打造真正全球化的对话式应用程序。

TIFFANY（en-US，女声）和 MATTHEW（en-US，男声）为专属多语种音色，支持朗读所有支持的语言：

1. 英语
2. 法语
3. 意大利语
4. 德语
5. 西班牙语
6. 葡萄牙语
7. 印地语

这使得 Tiffany 和 Matthew 适合需要在多种语言之间无缝切换的应用程序。

## 管理聊天记录

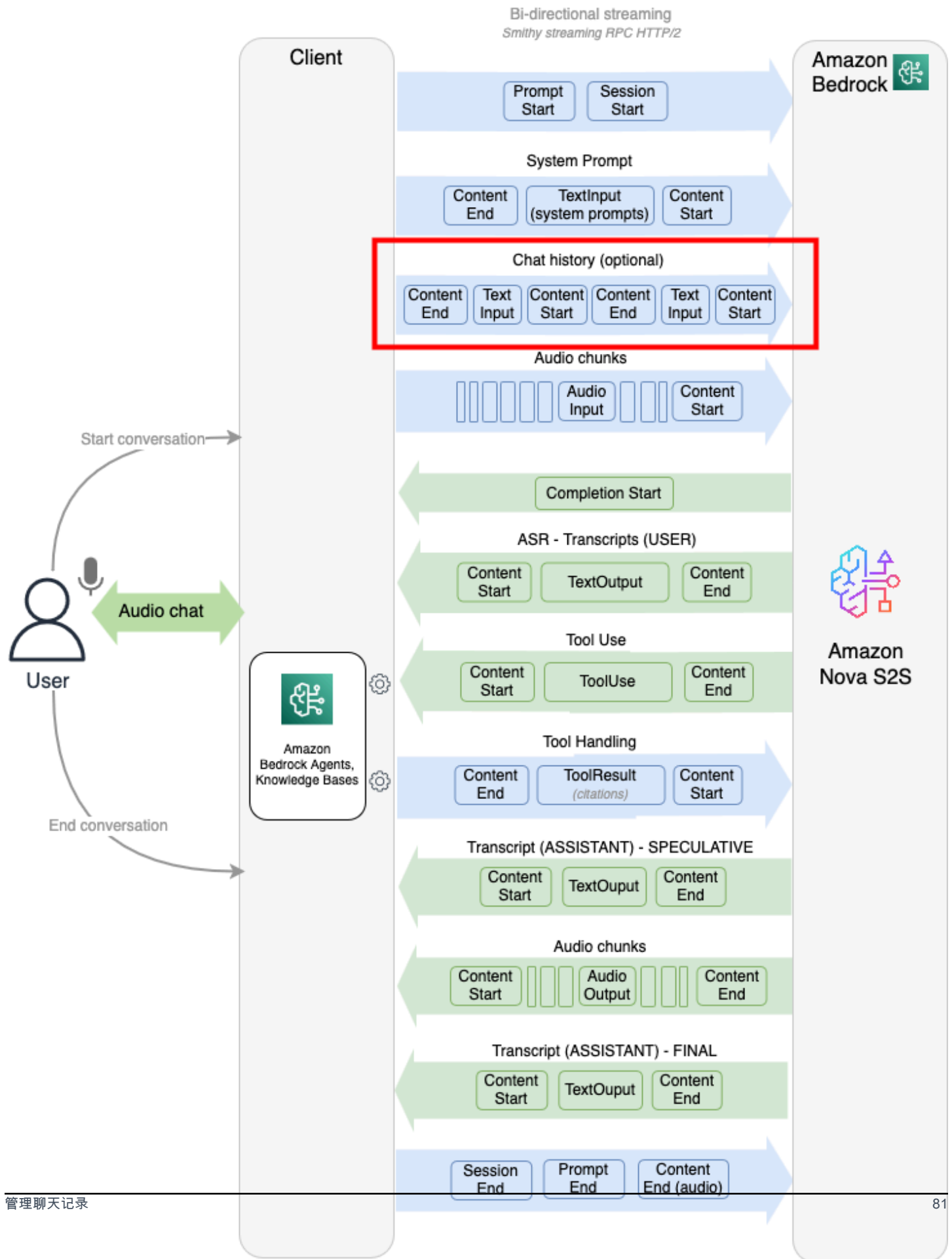
Amazon Nova 2 Sonic 的响应包括用户和助手语音的 ASR（自动语音识别）转录。存储聊天记录是一种最佳实践，不仅可用于日志记录，还可在连接意外关闭时恢复会话。这使得客户端能够将上下文回传到 Nova Sonic，实现对话无缝接续。

有关管理聊天记录的更多信息，请参阅以下资源：

1. [聊天记录日志](#)
2. [对话恢复](#)

## 发送聊天记录

对话历史仅可添加一次，且需在系统/语音提示之后、音频流开始之前传入。整体聊天记录不能超过 40 KB。下图展示了在事件生命周期中何时传入聊天记录：



每条历史消息都需要三个事件：contentStart、textInput 和 contentEnd。

每条消息的事件架构：

- contentStart：定义消息角色和配置

```
{
  "event": {
    "contentStart": {
      "promptName": "<prompt-id>",
      "contentName": "<content-id>",
      "type": "TEXT",
      "interactive": true,
      "role": "ASSISTANT",
      "textInputConfiguration": {
        "mediaType": "text/plain"
      }
    }
  }
}
```

- textInput：包含实际的消息内容。一个 textInput 不能大于 1 KB。如果大小超过限制，请在同一个内容块中拆分成多个 textInput。如果对话大于 40 KB，请修剪整个聊天记录。

```
{
  "event": {
    "textInput": {
      "promptName": "<prompt-id>",
      "contentName": "<content-id>",
      "content": "Take your time, Don. I'll be here when you're ready."
    }
  }
}
```

- contentEnd：标记消息的结束

```
{
  "event": {
    "contentEnd": {
      "promptName": "<prompt-id>",
      "contentName": "<content-id>"
    }
  }
}
```

```
}
```

对聊天记录中的每条消息重复这三个事件，交替使用用户和助手角色。

**重要注意事项：**

- 每次会话只能包含一次聊天记录
- 聊天记录必须在系统提示之后和音频流式传输开始之前发送
- 必须先发送所有历史消息，然后才能开始音频流式传输
- 每条消息都必须指定用户或助手角色
- 将 textOutput 事件中存储的转录内容用作 textInput 中的内容值

## 接收 ASR 转录文本

在对话期间，Amazon Nova 2 Sonic 通过输出事件发送 ASR 转录文本。每份转录文本都以三个事件的序列形式交付：contentStart、textOutput 和 contentEnd。

示例：用户演讲转录文本：

1. contentStart – 表示转录文本开始：

```
{
  "event": {
    "contentStart": {
      "additionalModelFields": "{\"generationStage\":\"FINAL\"}",
      "completionId": "<completion-id>",
      "contentId": "<content-id>",
      "promptName": "<prompt-id>",
      "role": "USER",
      "sessionId": "<session-id>",
      "textOutputConfiguration": {
        "mediaType": "text/plain"
      },
      "type": "TEXT"
    }
  }
}
```

2. textOutput – 包含实际转录文本内容：

```
{
  "event": {
    "textOutput": {
      "completionId": "<completion-id>",
      "content": "hello how are you",
      "contentId": "<content-id>",
      "promptName": "<prompt-id>",
      "role": "USER",
      "sessionId": "<session-id>"
    }
  }
}
```

### 3. contentEnd – 标记转录文本结束：

```
{
  "event": {
    "contentEnd": {
      "completionId": "<completion-id>",
      "contentId": "<content-id>",
      "promptName": "<prompt-id>",
      "sessionId": "<session-id>",
      "stopReason": "PARTIAL_TURN",
      "type": "TEXT"
    }
  }
}
```

相同的三事件模式适用于用户角色和助手角色。从 textOutput 事件中提取 content 字段，从 contentStart 事件中提取 role 字段，用于构建聊天记录。

## 最佳实践

务必存储聊天记录以实现：

- 跨设备恢复会话
- 对话日志与审计
- 为后续交互保留上下文

**重要提示：**保存聊天记录时，请根据 generationStage 使用文本输出：

- 预测文本 – 音频合成开始前生成的 Nova 2 Sonic 拟播报内容预览
- 最终文本 – 对音频回复中实际播报内容的句子级转录结果

务必将最终文本输出存入聊天记录，因其为对话的准确记录。

最终输出示例（请将此内容保存到聊天记录）：

```
ContentStart event: {
  "additionalModelFields": "{\"generationStage\":\"FINAL\"}",
  "completionId": "<completion-id>",
  "contentId": "<content-id>",
  "role": "ASSISTANT",
  "sessionId": "<session-id>",
  "type": "TEXT"
}
```

预测输出示例（仅作可选预览，不记入历史）：

```
ContentStart event: {
  "additionalModelFields": "{\"generationStage\":\"SPECULATIVE\"}",
  "completionId": "<completion-id>",
  "contentId": "<content-id>",
  "role": "ASSISTANT",
  "sessionId": "<session-id>",
  "type": "TEXT"
}
```

## 工具配置

Amazon Nova 2 Sonic 支持工具使用（亦称函数调用），模型可在对话过程中请求外部信息或执行操作，例如调用 API、查询数据库或运行自定义代码函数。这让语音助手能够根据用户指令执行操作、获取信息并对接外部服务。

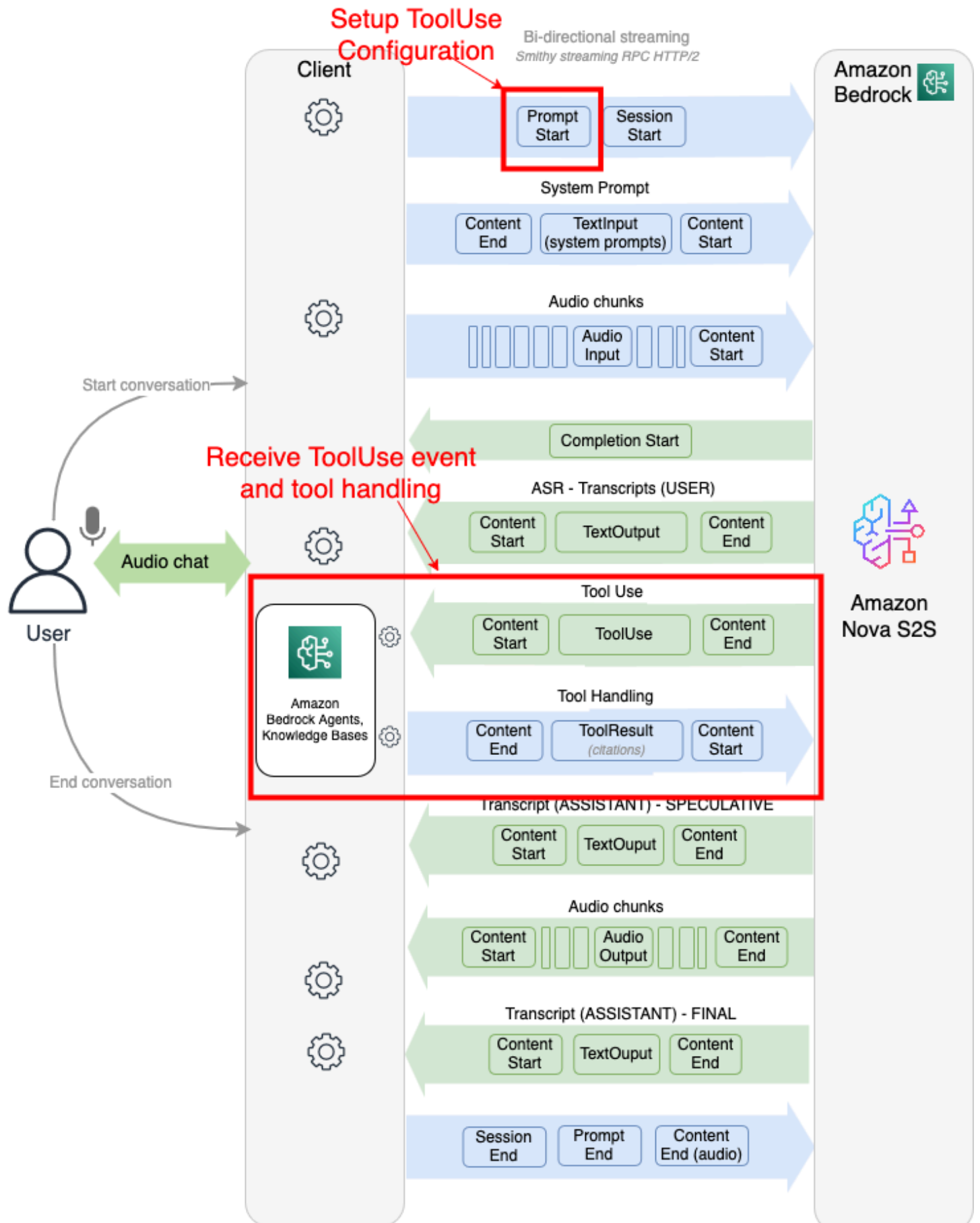
Nova 2 Sonic 采用异步工具调用机制，人工智能可在工具于后台运行的同时继续自然对话，带来更流畅、更灵敏的使用体验。

以下为使用工具的简化流程：

1. 定义工具：在 `promptStart` 事件中，指定可用工具及其参数

2. 用户发话：用户提出需要调用工具的请求（例如“西雅图天气如何？”）
3. 工具调用：Nova 2 Sonic 识别需求并发送 toolUse 事件
4. 执行工具：应用程序执行对应工具并返回结果
5. 回复生成：Nova 2 Sonic 将结果整合进语音回复中

下图展示了工具使用的工作流程：



## 定义工具

工具通过 JSON 架构定义，用于描述其用途、参数及预期输入。

以下为工具定义的组成部分及说明：

- Name：工具的唯一标识符（使用 snake\_case）
- Description：清晰说明该工具的功能，辅助人工智能判断调用时机
- InputSchema：定义该工具接受的参数的 JSON 架构
- Properties：带有类型和描述的单个参数
- Required：必须提供的参数名称数组

### 工具定义示例

以下为简易天气工具定义示例

```
{
  "toolSpec": {
    "name": "get_weather",
    "description": "Get current weather information for a specific location",
    "inputSchema": {
      "json": {
        "type": "object",
        "properties": {
          "location": {
            "type": "string",
            "description": "City name or zip code"
          },
          "units": {
            "type": "string",
            "enum": ["celsius", "fahrenheit"],
            "description": "Temperature units"
          }
        },
        "required": ["location"]
      }
    }
  }
}
```

## 在 PromptStart 中配置工具

在 promptStart 事件中，工具配置与音频和文本输出设置一起传递给 Nova 2 Sonic：

```
{
  "event": {
    "promptStart": {
      "promptName": "<prompt-id>",
      "textOutputConfiguration": {
        "mediaType": "text/plain"
      },
      "audioOutputConfiguration": {
        "mediaType": "audio/lpcm",
        "sampleRateHertz": 16000,
        "sampleSizeBits": 16,
        "channelCount": 1,
        "voiceId": "matthew",
        "encoding": "base64",
        "audioType": "SPEECH"
      },
      "toolUseOutputConfiguration": {
        "mediaType": "application/json"
      },
      "toolConfiguration": {
        "tools": [
          {
            "toolSpec": {
              "name": "get_weather",
              "description": "Get current weather information for a
specific location",
              "inputSchema": {
                "json": {
                  "type": "object",
                  "properties": {
                    "location": {
                      "type": "string",
                      "description": "City name or zip code"
                    },
                    "units": {
                      "type": "string",
                      "enum": ["celsius", "fahrenheit"],
                      "description": "Temperature units"
                    }
                  }
                }
              }
            }
          ]
        }
      }
    }
  }
}
```



```
"toolChoice": {
  "tool": {
    "name": "get_weather"
  }
}
```

## 接收并处理工具使用事件

当 Amazon Nova 2 Sonic 判断需要使用工具时，会发送一个 toolUse 事件，包含以下内容：

1. toolUseID：本次工具调用的唯一标识符
2. ToolName：待执行的工具名称
3. Content：JSON 字符串，包含从用户请求中提取的参数
4. SessionID：当前会话标识符
5. Role：工具使用事件固定设为 TOOL

## 工具使用事件示例

```
{
  "event": {
    "toolUse": {
      "completionId": "<completion-id>",
      "content": "{\"location\": \"Seattle\", \"units\": \"fahrenheit\"}",
      "contentId": "<content-id>",
      "promptName": "<prompt-id>",
      "role": "TOOL",
      "sessionId": "<session-id>",
      "toolName": "get_weather",
      "toolUseId": "<tool-use-id>"
    }
  }
}
```

## 处理步骤

1. 接收来自 Nova 2 Sonic 的 toolUse 事件
2. 从事件中提取工具名称和参数
3. 执行工具逻辑（API 调用、数据库查询等）
4. 使用 toolResult 事件返回结果

## ToolResult 事件示例

```
{
  "event": {
    "toolResult": {
      "promptName": "<prompt-id>",
      "contentName": "<content-id>",
      "content": "{\"temperature\": 72, \"condition\": \"sunny\", \"humidity\": 45}"
    }
  }
}
```

## 最佳实践

- 清晰的描述：撰写详细的工具描述，帮助 Nova 2 Sonic 了解何时使用每种工具。
- 验证参数：务必在执行之前验证工具参数，防止出错。尽可能使用带有结构化数据类型（例如枚举、数字或布尔值）的正确 JSON 架构（而不是开放式字符串）来定义工具参数。
- 错误处理：工具执行失败时，在 toolResult 事件中返回有效的错误信息。
- 异步执行：利用异步工具调用，保障对话流程顺畅。
- 工具命名：使用描述性、动作导向的名称（如 get\_weather、search\_database、send\_email）。

## 异步工具调用

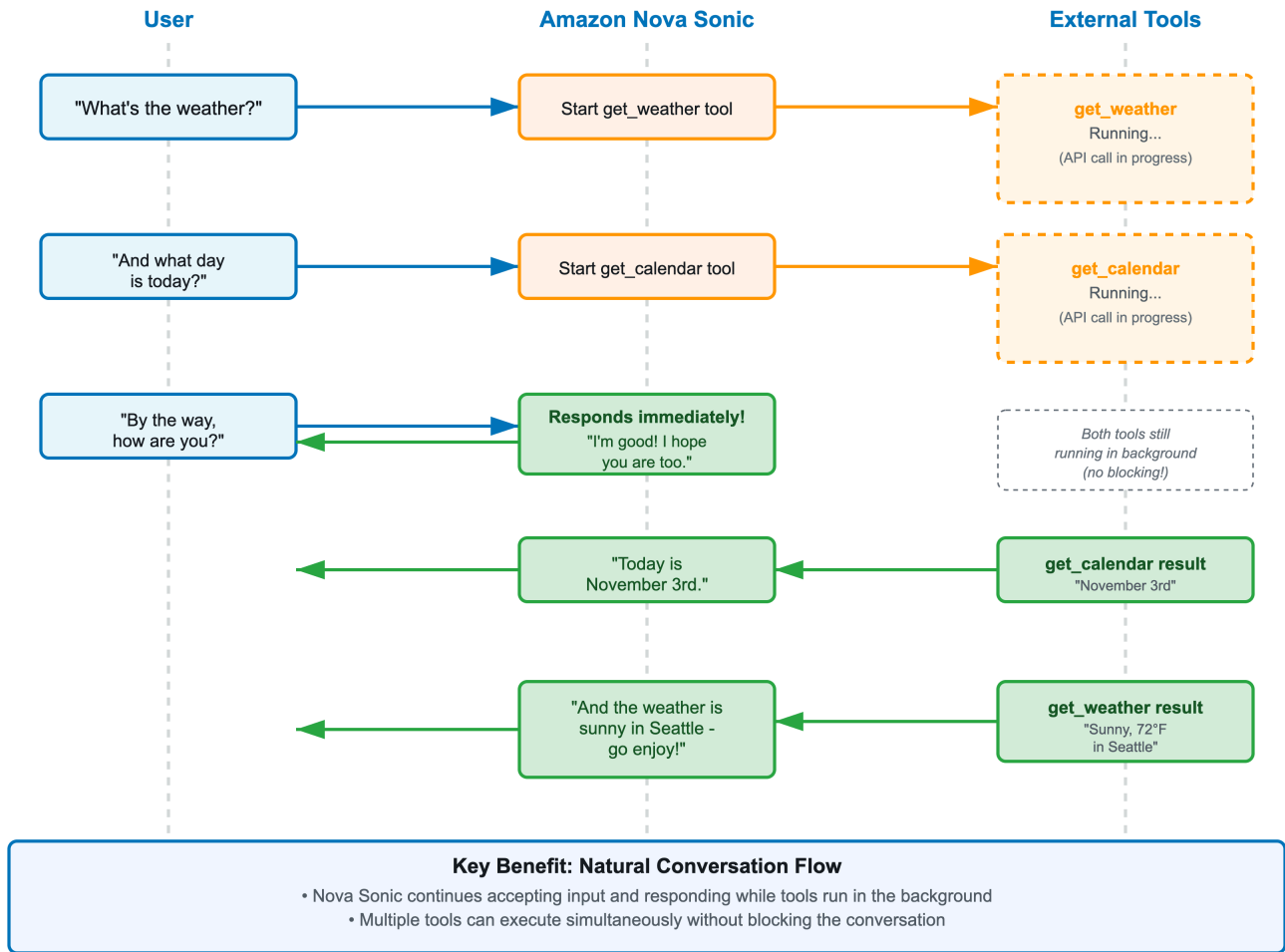
与传统同步工具调用中人工智能需静默等待工具执行结果不同，Amazon Nova 2 Sonic 的异步方案可实现：

- 工具运行期间可持续接收用户输入
- 无需等待处理工具执行结果，即可对新问题作出回应
- 同时处理多个工具调用
- 保持自然对话节奏，避免出现生硬停顿
- 无需额外配置。异步工具调用开箱即用。

## 工作原理

当 Nova 2 Sonic 发起工具调用时，不会中断对话。相反，其会持续正常监听并作出响应，直至工具返回结果。

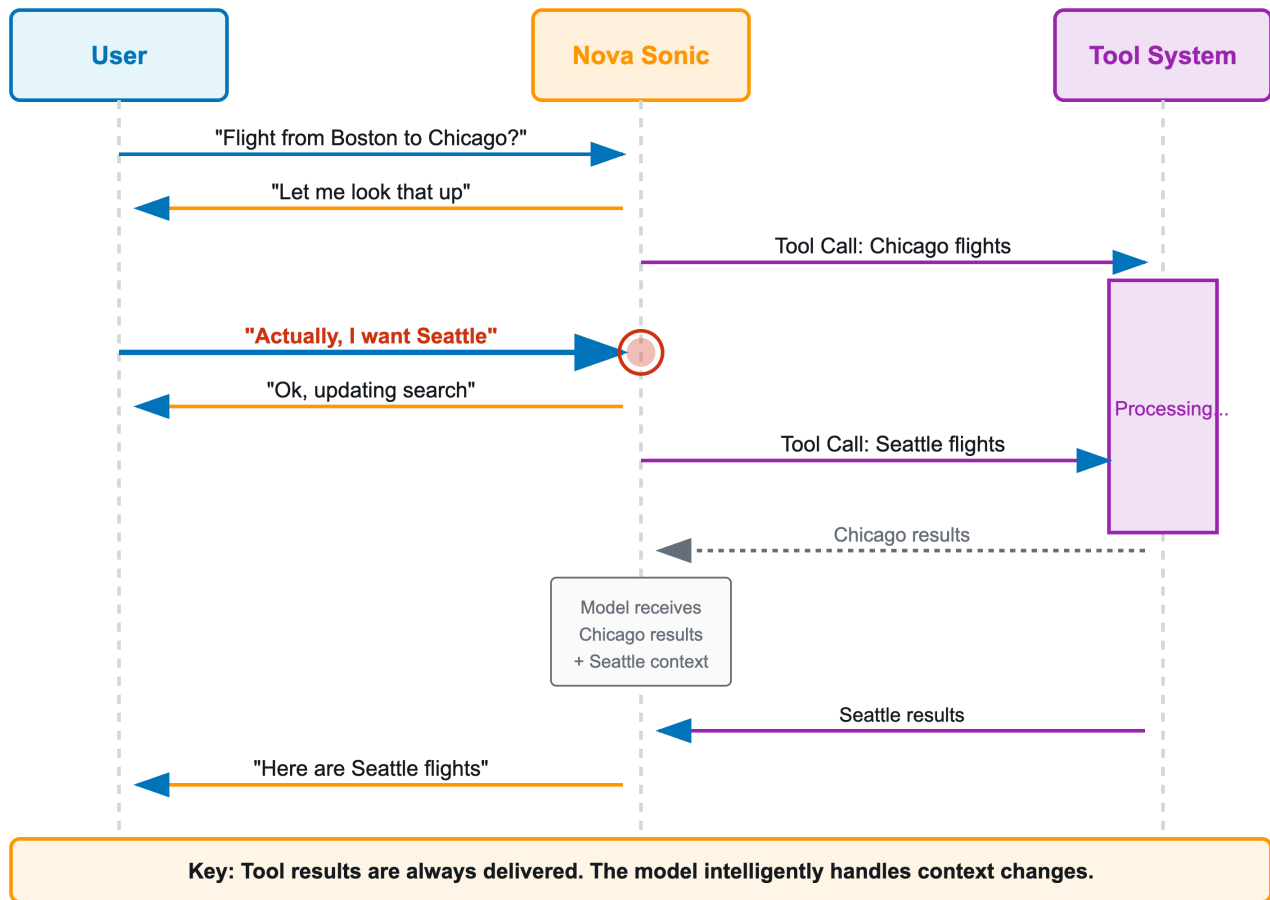
### Asynchronous Tool Calling Flow



### 处理用户中断

如果用户在工具执行过程中更改请求，Nova 2 Sonic 会智能处理，且不会取消正在等待的工具调用。

## Async Tool Calling with User Interruption



### 示例方案

```

User: "Can I book a flight from Boston to Chicago?"
Agent: "Sure, let me look that up for you."
Agent: [initiates tool call for Chicago flights]
User: "Actually, I want to go to Seattle"
Agent: "Ok let me update that search"
Agent: [initiates tool call for Seattle flights]
[First tool returns with Chicago flight results]
Agent: [receives Chicago results and processes them contextually]
  
```

### 工作原理

始终交付工具执行结果：工具调用完成后，即使用户更改了请求，工具执行结果也始终发送到模型。模型会使用其推理功能来确定如何处理信息。

上下文感知处理：模型了解对话上下文，可以适当地处理过时的工具执行结果。例如：

- 如果用户在改变主意后说“谢谢”，则模型仍然需要原始的上下文结果
- 如果用户更改了请求，模型可以确认原始结果，同时专注于新请求

不自动取消：系统不会依据用户新输入内容自动取消或忽略已发起的工具调用。这可确保模型获取完整信息，从而智能决策应答方式。

## 集成

Amazon Nova 2 Sonic 可与多种框架和平台集成，用于构建对话式人工智能应用程序。这些集成为常见使用案例提供了预先构建的组件和简化的 API。

### Strands Agents

Strands Agents 是一个简单但功能强大的 SDK，采用模型驱动的方法来构建和运行人工智能代理。从简易对话助手到复杂自主 workflows，从本地开发到生产部署，Strands Agents 均可随您的需求灵活扩展。

有关 Strands 框架的完整文档，请访问 [Strands 官方文档](#)。

Strands BidiAgent 通过持久化流式连接，实现实时音频与文本交互。与传统的请求-响应模式不同，此代理可保持长时间运行的对话，支持中断、并行处理和连续音频响应。

先决条件：

- 已安装 Python 3.8 或更高版本。
- 配置有对 Amazon Bedrock 的访问权限的 AWS 凭证
- 基本了解 Python async/await 语法

代码示例：

代码示例

安装：

安装所需的软件包：

```
pip install strands-agents strands-agents-tools
```

运行本示例：

```
import asyncio
from strands.experimental.bidi.agent import BidiAgent
from strands.experimental.bidi.io.audio import BidiAudioIO
from strands.experimental.bidi.io.text import BidiTextIO
from strands.experimental.bidi.models.novasonic import BidiNovaSonicModel
from strands_tools import calculator

async def main():
    """Test the BidirectionalAgent API."""
    # Audio and Text input/output utility
    audio_io = BidiAudioIO(audio_config={})
    text_io = BidiTextIO()

    # Nova Sonic model
    model = BidiNovaSonicModel(region="us-east-1")

    async with BidiAgent(model=model, tools=[calculator]) as agent:
        print("New BidiAgent Experience")
        print("Try asking: 'What is 25 times 8?' or 'Calculate the square root of
144'")

        await agent.run(
            inputs=[audio_io.input()],
            outputs=[audio_io.output(), text_io.output()]
        )

if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        print("\nConversation ended by user")
    except Exception as e:
        print(f"Error: {e}")
        import traceback
        traceback.print_exc()
```

## 1. 导入：所需模型

```
from strands.experimental.bidi.agent import BidiAgent
from strands.experimental.bidi.io.audio import BidiAudioIO
from strands.experimental.bidi.io.text import BidiTextIO
```

```
from strands.experimental.bidi.models.novasonic import BidiNovaSonicModel
from strands_tools import calculator
```

- BidiAgent : 编排双向对话的主代理类
- BidiAudioIO : 处理语音交互的音频输入和输出
- BidiTextIO : 为转录和响应提供文本输出
- BidiNovaSonicModel : Nova 2 Sonic 模型包装器
- Calculator : 用于数学运算的预构建工具

## 2. 配置音频与文本 I/O

```
audio_io = BidiAudioIO(audio_config={})
text_io = BidiTextIO()
```

BidiAudioIO 管理麦克风输入和扬声器输出，而 BidiTextIO 则在控制台中显示文本转录和响应。

## 3. 初始化模型

```
model = BidiNovaSonicModel(region="us-east-1")
```

创建 Nova Sonic 模型实例。region 参数指定部署模型所在的 AWS 区域。

## 4. 创建并运行代理

```
async with BidiAgent(model=model, tools=[calculator]) as agent:
    await agent.run(
        inputs=[audio_io.input()],
        outputs=[audio_io.output(), text_io.output()]
    )
```

创建代理时需配置以下项：

- Model : 要使用的 Nova 2 Sonic 模型
- Tools : 代理可调用的工具列表 ( 如计算器 )
- Inputs : 麦克风音频输入
- Outputs : 音频输出到扬声器，文本输出到控制台

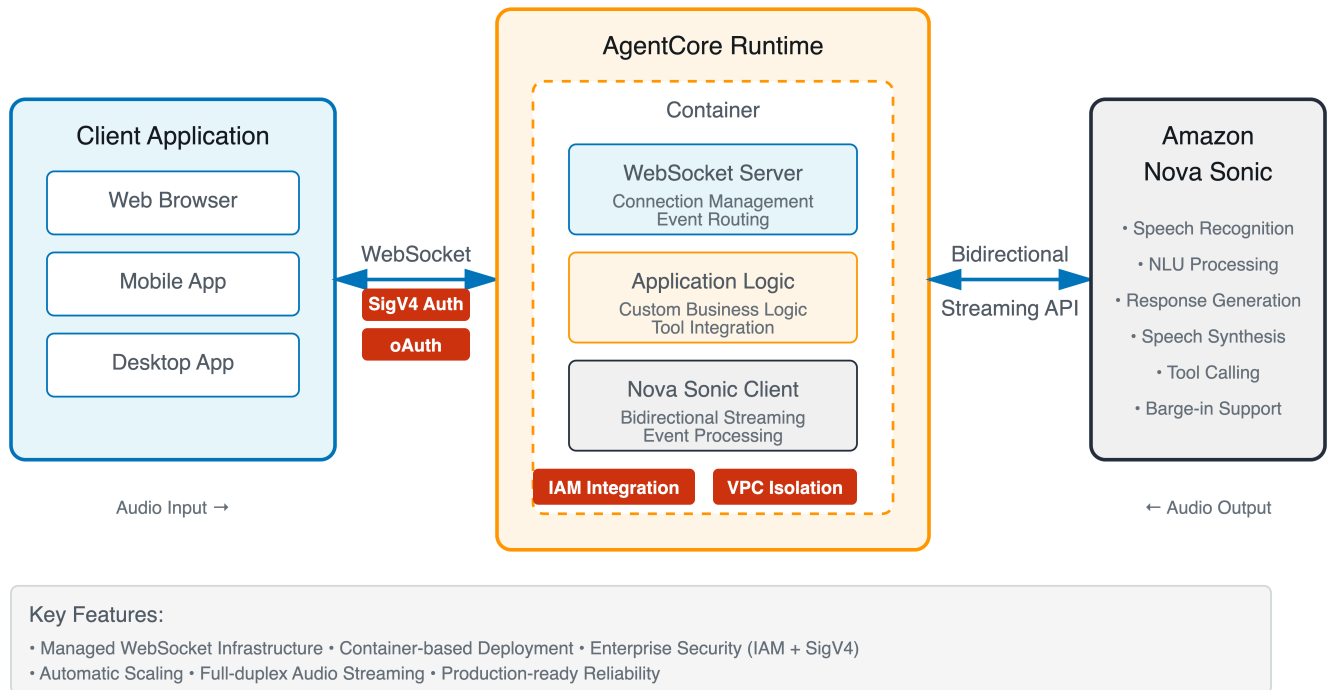
## 框架集成

Amazon Nova 2 Sonic 可与各种框架和平台集成，用于构建高阶语音应用程序。以下示例展示了与主流框架的集成方案。

### Amazon Bedrock AgentCore

Amazon Bedrock AgentCore 为部署具有企业级安全性和可扩展性的 Nova 2 Sonic 应用程序，提供了托管运行时环境。AgentCore 负责底层基础设施、身份验证及 WebSocket 连接管理，简化了实时语音人工智能应用程序的部署流程。

### AgentCore Architecture



### 主要特征：

- **双向流式传输：**原生支持 Nova Sonic 全双工流接口，支持实时事件处理与低时延通信。
- **WebSocket 基础设施：**可直接用于生产环境的 WebSocket 服务器，具备自动扩缩容、连接管理与故障恢复能力。
- **容器部署：**支持通过水平扩缩容与独立版本控制，将 Nova Sonic 应用程序以容器形式部署到托管基础设施。

- 企业级安全：通过 IAM 与 SigV4 实现精细化身份验证，支持 VPC 隔离及完备的审计日志记录。

该架构展示了客户端应用程序如何通过 SigV4 身份验证，经由 WebSocket 连接到 AgentCore 运行时。容器化环境包含 WebSocket 服务器、应用逻辑与 Nova Sonic 客户端，所有组件均通过双向流式 API 与 Nova Sonic 进行通信。

优点：

- 简化运维：专注于应用逻辑，由 AgentCore 负责基础设施、扩缩容与可靠性保障。
- 企业级安全：内置身份验证、授权与合规特性，适配生产环境部署。
- 成本效益：按需付费，支持自动扩缩容与资源优化。
- 开发人员工作效率：通过托管式 WebSocket 基础设施和容器部署缩短投产时间。

使用案例

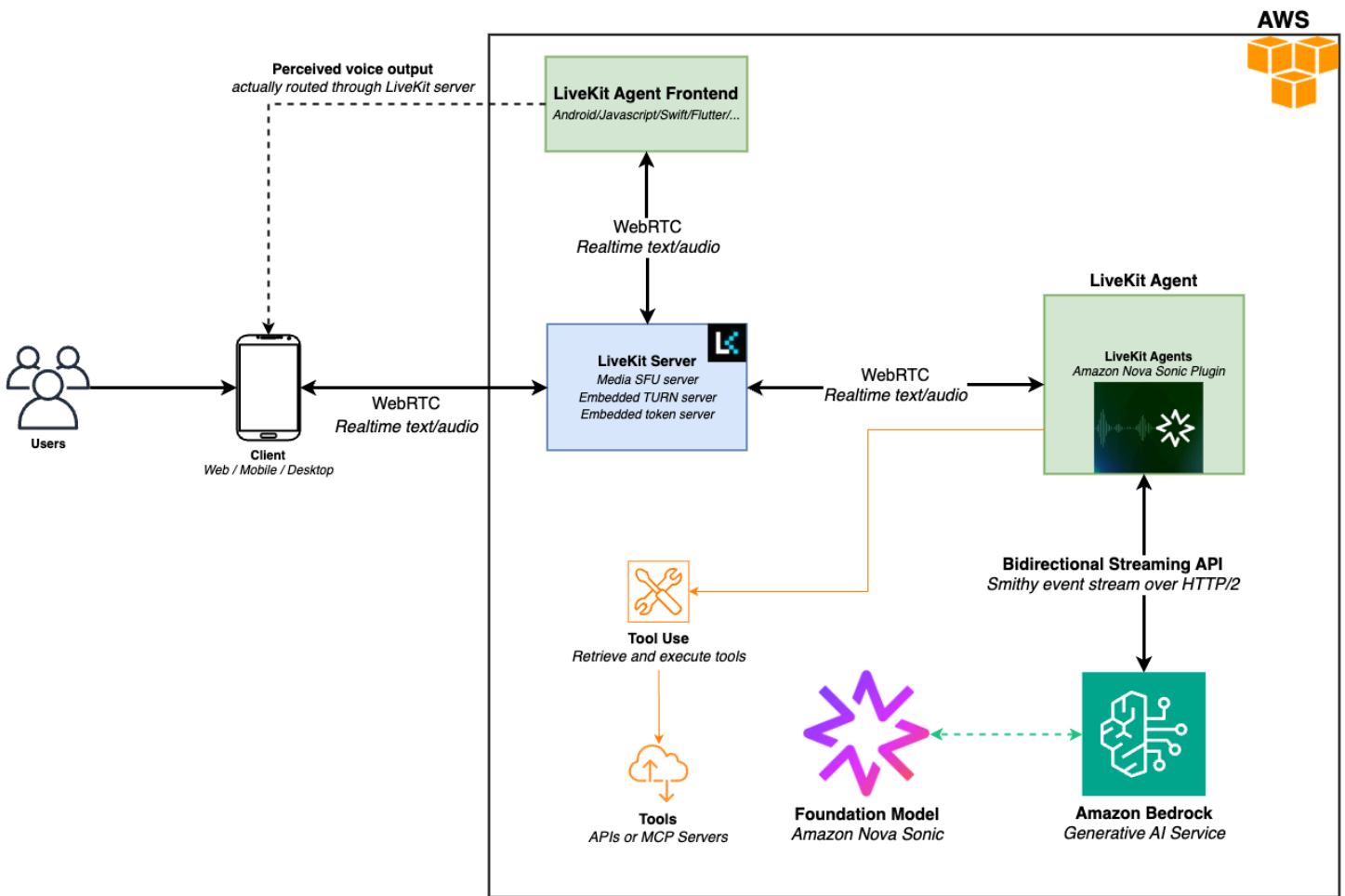
- 具有安全身份验证功能的客户服务语音助手
- 需要 IAM 集成的企业语音应用程序
- 具有隔离部署功能的多租户语音平台
- 需要合规性和审计跟踪记录的支持语音的应用程序

有关使用 AgentCore 部署 Nova Sonic 的详细文档，请访问 [Amazon Bedrock AgentCore 文档](#)。

LiveKit

LiveKit 是一款开源平台，用于构建实时音频和视频应用程序。与 Amazon Nova 2 Sonic 的集成使开发人员无需管理复杂的音频管道或信令协议，即可构建对话式语音接口。

有关详细的实现示例与代码示例，请访问 [LiveKit AWS 集成文档](#)。



## 工作原理：

- 客户端层：网络、移动或桌面应用程序使用 LiveKit 的客户端 SDK 进行连接，这些软件开发工具包处理音频采集、WebRTC 流式传输和播放。
- LiveKit 服务器：充当实时通信中心，管理 WebRTC 连接，路由音频流，并通过低延迟优化处理会话状态。
- LiveKit Agent：基于 Python 的代理，从服务端接收音频，经 Nova Sonic 插件处理后，以流式方式返回响应。包括语音活动检测与轮次管理等内置功能。
- Amazon Nova 2 Sonic：通过双向流式 API 处理音频流，执行语音识别、自然语言理解以及使用合成语音生成对话回复。

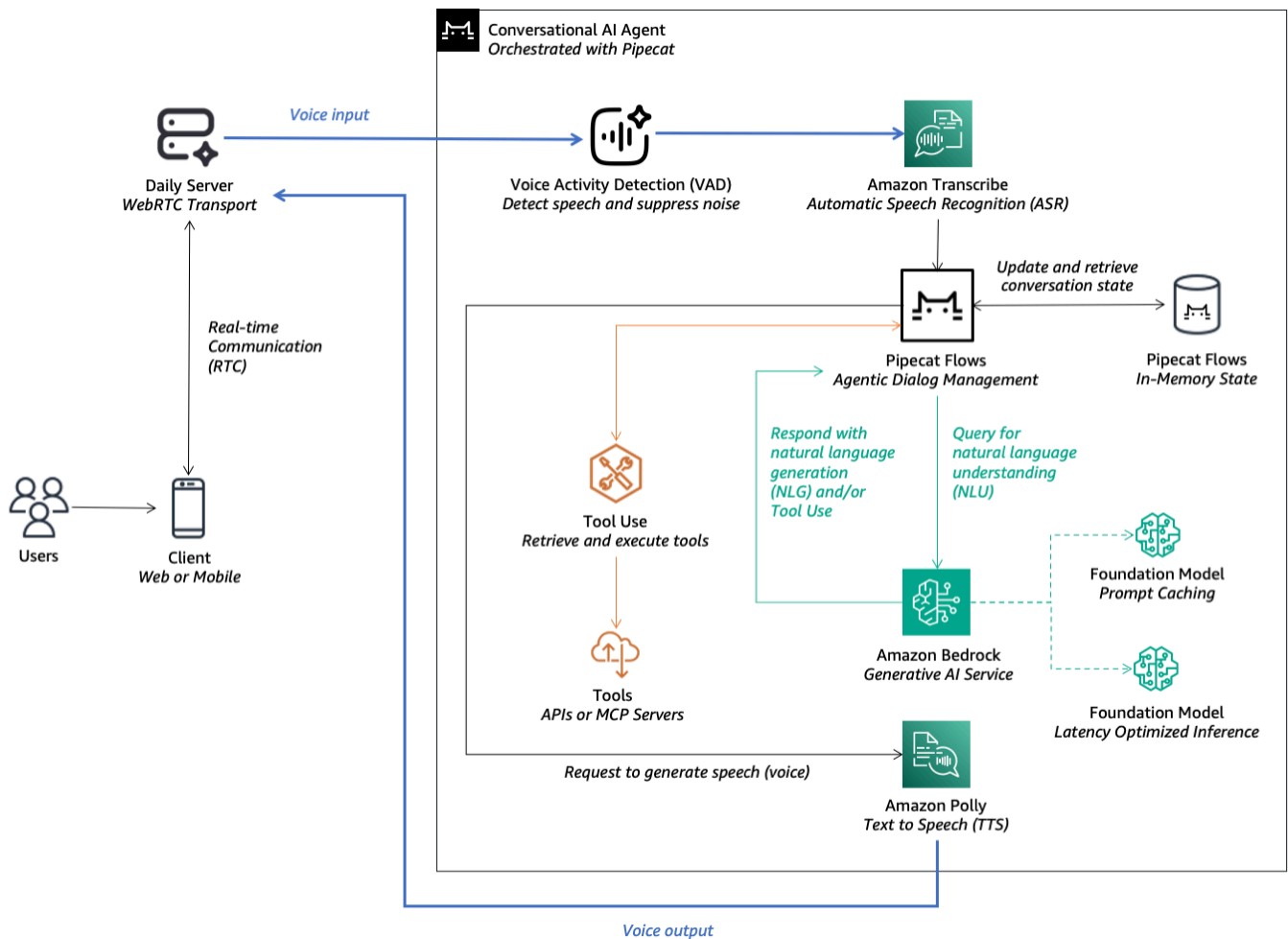
## Pipecat

Pipecat 是一个用于构建语音和多模态对话式人工智能应用程序的框架。该框架提供了基于管道的模块化架构，可编排多个组件，使用 Amazon Nova Sonic 和其他 AWS 服务创建智能语音应用程序。

有关详细的实现示例与代码示例，请访问 [PipeCat AWS 集成文档](#)。

主要特征：

- 管道架构：基于 Python 的模块化框架，用于合成语音人工智能组件，包括 ASR、NLU、TTS 等。
- Pipecat 流程：用于构建复杂对话逻辑和工具执行的状态管理框架。
- WebRTC 支持：内置与 Daily 和其他 WebRTC 提供商的集成，用于实时音频流式传输。
- AWS 集成：原生支持 Amazon Bedrock、Amazon Transcribe 与 Amazon Polly。



此架构包括：

- WebRTC 传输：实现客户端设备与应用程序服务器间的实时音频流式传输。
- 语音活动检测 (VAD)：Silero VAD 支持可配置的语音检测与噪声抑制。
- 语音识别：Amazon Transcribe 实现精准、实时的语音转文本。

- 自然语言理解：部署在 Bedrock 上的 Amazon Nova Pro，推理经过低延迟优化。
- 工具执行：采用 Pipecat Flows 实现 API 集成与后端服务调用。
- 回复生成：采用 Amazon Nova Pro，生成连贯、贴合上下文的应答内容。
- 文本转语音：搭载生成式语音的 Amazon Polly，输出自然逼真的语音。

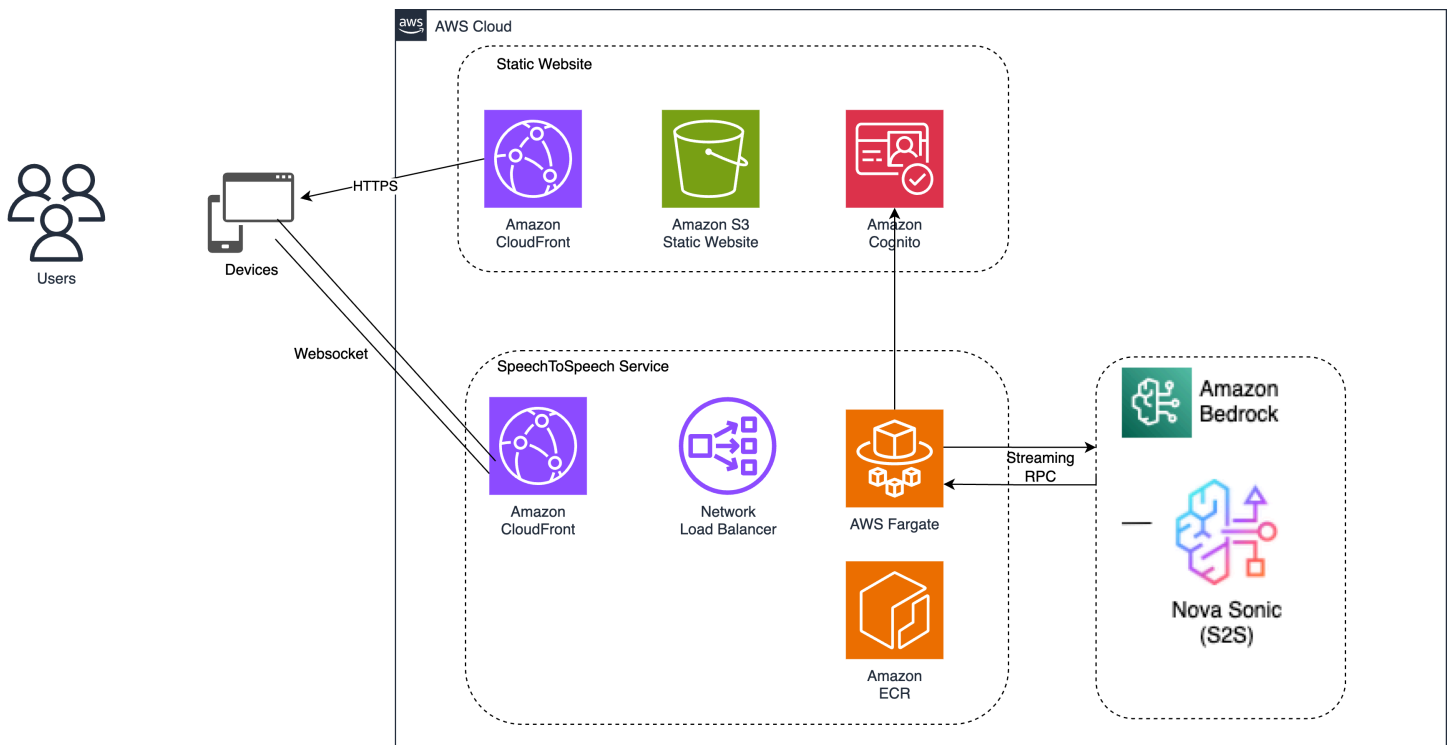
## 在 AWS 上部署

借助 AWS CDK (云开发工具包)，以基础设施即代码方式将 Nova Sonic 应用程序部署到 AWS。该方案可实现可复用、可版本控制的部署，并内置行业最佳实践。

## 部署选项

- Amazon ECS (弹性容器服务)：全托管式容器编排服务，集成应用程序负载均衡器，支持自动扩缩容与无服务器 Fargate 运行模式。
- Amazon EKS (弹性 Kubernetes 服务)：托管式 Kubernetes，适用于复杂编排、高级网络、多区域部署及丰富的工具生态体系。
- AWS CDK：AWS CDK 支持使用熟悉的编程语言定义云基础设施。

有关使用 AWS CDK 部署 Nova Sonic 的完整生产就绪型示例，请参阅 GitHub 上的[语音转语音 CDK 示例](#)。此示例演示：



- 使用 TypeScript 完成 CDK 基础设施设置
- 用于实时通信的 WebSocket 服务器实现
- 使用 ECS 与 Fargate 部署容器
- 用于提供 WebSocket 支持的应用程序负载均衡器配置
- VPC 网络与安全组设置
- CloudWatch 监控与日志记录
- 生产部署最佳实践

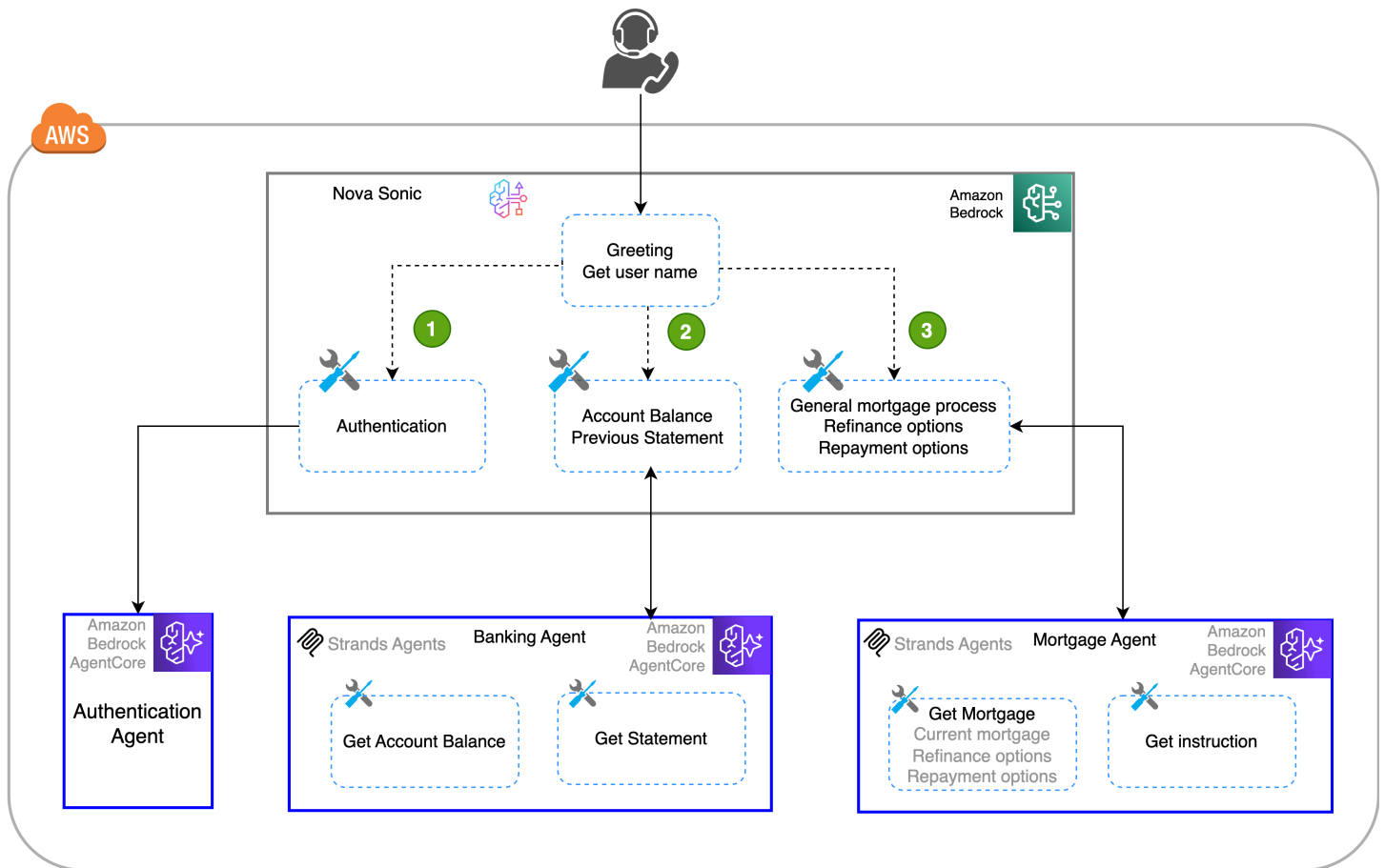
## 多代理系统

多代理架构是一种广泛采用的模式，用于设计处理复杂任务的人工智能助手。在基于 Nova 2 Sonic 构建的语音助手中，该架构可协调多个专用代理；各代理独立运行，从而实现并行处理、模块化设计与可扩展方案。

Nova Sonic 在多代理系统中充当编排工具，主要执行两大核心功能：

**对话流程管理：**确保在进入对话下一环节前，完成所有必要信息的收集。

**意图分类：**分析用户咨询内容，并将其分转接到对应的专用子代理处理。



上图展示了使用多代理架构的银行语音助手。对话流程先以问候开场并采集用户姓名，再通过专用子代理处理银行或按揭贷款相关咨询。

对话流程示例：

1. 用户接入语音助手。
2. Nova 2 Sonic：“您好！请问您怎么称呼？”
3. 用户：“我叫 John”
4. Nova 2 Sonic：“您好，John。今天有什么可以帮您？”
5. 用户：“我想查询账户余额”
6. Nova 2 Sonic：[转接到身份验证代理]
7. 身份验证代理：“请提供您的账号”
8. 用户：“12345”
9. 身份验证代理：[正在验证身份]
10. Nova 2 Sonic：[转接到银行代理]

## 11. 银行代理：“账户当前余额为 5,431,10 美元”

虽然本示例展示了基于部署在 Amazon Bedrock AgentCore 上的 Strands Agents 框架实现的子代理，但该架构具备高度灵活性。您可以选择：

- 首选代理框架
- 任何 LLLM 提供商
- 自定义托管选项
- 不同的编排模式

优点：

- 模块化：各代理专注于特定领域，便于系统维护与更新。
- 可扩展性：需改动现有代理即可新增代理，支持系统按需扩展。
- 并行处理：多代理可同步运行，缩短复杂查询的响应时长。
- 专业化：各代理可针对专属任务优化，适配最优工具与知识库。
- 故障隔离：单个代理故障不影响其他模块，提升系统整体可靠性。

有关更多详情与代码示例，请参阅[此博客](#)。

有关实操示例，请参阅 [Nova Sonic 讲习会多代理实验室](#)。

### 电话集成

Amazon Nova 2 Sonic 可与电话服务提供商对接，实现可通过通话访问的人工智能语音应用程序。本指南介绍其与 Twilio、Vonage 及其他基于 SIP 的系统的集成方案，用于搭建联络中心解决方案与语音代理。

Twilio：云通信平台，提供可编程语音与媒体流能力。

Vonage：全球通信 API 服务商，支持语音、WebSocket 音频流及 SIP 连接。

AWS 提供完整的示例实现方案，在联络中心环境中结合实时分析与电话集成功能展示 Nova Sonic 的应用程序。

存储库：[支持电话功能的 Sonic 联络中心示例](#)

# 使用 Amazon Nova 构建高级系统

## 概述

本节面向高阶用户，提供使用 Nova 模型实现高级模式的完整指南。这些功能可支撑构建高阶应用程序，包括工具集成、检索增强生成 ( RAG ) 和自主人工智能代理。

## Amazon Nova 2 中的扩展思考

Amazon Nova 2 Lite 新增扩展思考能力，使模型能够针对复杂问题开展更深度的推理。该可选功能让您控制模型在何时、以何种方式分配额外计算资源，以深入思考并处理高难度任务。

### 扩展思考工作原理

Amazon Nova 2 将扩展思考作为一项混合能力推出。您可完全自主控制：

- 扩展思考关闭 ( 默认 )：Amazon Nova 2 采用高效隐式推理，适用于日常任务与高吞吐量应用场景。
- 扩展思考开启：Amazon Nova 2 执行显式、分步式推理，最适合需要深度分析的复杂问题。

### 推理强度

以下展示了不同级别的推理强度。

#### 低强度 (maxReasoningEffort:"low")

适用场景：复杂度更高、需结构化思考的任务。例如代码审查与优化建议 ( 模型需细致评估现有代码质量 )、需综合考量多重要素的分析任务，或适合采用系统化方法解决的问题场景。低强度适用于复合型任务，仅需基础推理即可提升精度，无需开展深度多步骤规划。

#### 中强度 (maxReasoningEffort: "medium")

适用场景：多步骤任务和编码 workflow。例如软件开发与调试 ( 模型需在修改前理解现有代码结构 )、跨多个文件或组件协同的代码生成、存在相互依赖关系的多步骤计算，以及带有多重约束条件的规划任务。中强度适用于需协调多个工具且模型需在一系列连续操作中保持上下文的智能代理 workflow，效果最佳。

## 高强度 (maxReasoningeffort:"high")

适用场景：STEM 推理和高级问题求解。例如需要严谨分步验证的高等数学问题与证明、需深度探究的科学分析研究任务、涉及多维度架构考量的复杂系统设计，以及影响重大的关键决策场景。高强度可在需要高阶推理、审慎评估备选方案并全面验证结论的任务中，实现最高精度。

## 快速入门：启用扩展思考

扩展思考通过 reasoningConfig 参数进行控制。

```
import boto3

bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')

response = bedrock.converse(
    modelId='us.amazon.nova-2-lite-v1:0',
    system=[{"text": "You are a highly capable personal assistant"}],
    messages=[{
        "role": "user",
        "content": [{"text": "Provide a meal plan for a gluten free family of 4."}]
    }],
    inferenceConfig={
        "temperature": 0.7,
        "topP": 0.9,
        "maxTokens": 10000
    },
    additionalModelRequestFields={
        "reasoningConfig": {
            "type": "enabled",
            "maxReasoningEffort": "low"
        }
    }
)

content_list = response["output"]["message"]["content"]

for item in content_list:
    if "reasoningContent" in item:
        reasoning_text = item["reasoningContent"]["reasoningText"]["text"]
        print("=== REASONING ===")
        print(reasoning_text)
        print()
    elif "text" in item:
```

```
print("=== ANSWER ===")
print(item["text"])
```

推理参数：

- type: enabled 或 disabled ( 默认值 : disabled )
- maxReasoningEffort : low、medium 或 high

### Note

将 maxReasoningEffort 设置为 high 时，不可同时使用 temperature、topP 和 topK 参数。因为组合使用这些参数会导致错误。

如需查看使用扩展思考功能的完整代码示例，请参阅[the section called “代码库”](#)。

## 响应结构

启用扩展思考后，响应内容将先包含 reasoningContent 块，随后返回 text 内容块：

```
{
  "output": {
    "message": {
      "role": "assistant",
      "content": [
        {
          "reasoningContent": {
            "reasoningText": {
              "text": "[REDACTED]"
            }
          }
        },
        {
          "text": "Based on the premises, we can conclude..."
        }
      ]
    }
  },
  "stopReason": "end_turn"
```

```
}
```

### Note

在 Amazon Nova 2 中，推理内容会显示为 [REDACTED]。相关推理词元仍会计费，因其有助于提升输出质量。当前在响应结构中保留该字段，是为了便于未来开放显示推理内容。我们正积极与客户协作，确定展示模型推理过程的最优方案。推理内容会显示为 [REDACTED]。相关推理词元仍会计费，因其有助于提升响应质量。

## 配置选项

Amazon Nova 2 新增了 `reasoningConfig` 参数，可添加到现有的 `Converse` 请求结构中以启用推理功能：

```
additionalModelRequestFields={
  "reasoningConfig": {
    "type": "enabled", # or "disabled" (default)
    "maxReasoningEffort": "high" # "low", "medium", or "high"
  }
}
```

参数：

- `type`：在 "enabled" 和 "disabled" 之间切换（默认值 "disabled"）
- `maxReasoningEffort`：启用后可控制推理深度
- "low"：适用于中等复杂度任务
- "medium"：适用于需要大量分析的复杂问题
- "high"：为极高复杂度任务提供最详尽的推理

### Note

使用 "high" 时，必须取消设置 `temp`、`topP` 和 `maxToken`。在此模式下，模型执行更深度的分析以得出最优解。这种更详尽的处理可能生成超过 65K 词元的输出内容；具体长度取决于请求复杂度，在部分场景中最高可达 128K 词元。这可以确保您获得完整、高质量的推理，而不是被截断的结果。

## 支持的模型

扩展思考目前可在以下模型中使用：Amazon Nova 2 Lite (us.amazon.nova-2-lite-v1:0)

## 结合工具使用的扩展思考

扩展思考可与工具调用无缝协同，使 Amazon Nova 能够推理出应使用哪些工具，以及如何解析工具返回结果。

## 了解推理词元和定价

### 词元类型

扩展思考耗用的词元将按输出词元计费：

- 输入词元：原始请求内容（标准输入定价）
- 输出词元：包含推理词元与最终可见响应内容（标准输出定价）

### 使用量明细

三类词元均计入使用量指标与计费范围。推理词元按输出词元标准计费，并会在模型响应中显示为 REDACTED。

```
{
  "usage": {
    "inputTokens": 45,
    "outputTokens": 1240,
    "totalTokens": 1285
  }
}
```

## 常见问题

为什么 Amazon Nova 2 Lite 将推理内容显示为 [REDACTED]，而非展示模型的思考过程？

本次发布的核心目标是确保 Nova 2 为任务提供顶尖的智能处理能力，这一点将体现在准确率的显著提升上。

我们深知推理过程可见性的重要价值，也收到了大量客户希望了解模型解题思考逻辑的强烈需求。

我们正在研究方案，力求尽快开放该功能。

推理词元仍会正常计费，因为这些词元对应提升输出质量的实际运算开销，

并会与应答词元一同计入 `outputTokens` 统计。

如果推理内容已被隐去，如何判断扩展思考是否生效？

您可通过检查响应中是否存在 `reasoningContent` 块，来确认扩展思考正常工作（这些块仅在启用推理时才会出现）

## 使用工具（函数调用）

工具可将模型与 API、数据库及代码执行环境等外部功能相连，从而拓展 Amazon Nova 的能力。通过使用工具，Amazon Nova 能够访问实时信息、进行计算并与外部系统交互。

### 了解工具使用工作流程

在 Amazon Nova 中使用工具涉及三个关键阶段：

#### 用户查询与工具定义

通过提供描述各工具功能与输入要求的 JSON 架构来定义工具。工具配置必须包括有关何时以及如何使用每种工具的详细描述。

#### 工具选择

用户发送消息时，Amazon Nova 将对消息进行分析，确定是否需要使用工具。这种自动工具选择会检查上下文并决定要调用的工具（如有）。如果 Amazon Nova 确定了合适的工具，就会返回工具名称和必填参数。

您负责根据模型的请求执行该工具。这意味着需要编写代码来调用工具功能并处理模型提供的输入参数。

#### 返回结果

工具执行完毕后，以 JSON 或文本 + 图像组合的结构化格式，将结果回传给 Amazon Nova。Amazon Nova 可以将工具输出整合到对最终回复中。如果在执行过程中出现错误，请在工具回复中注明这一点，以便 Amazon Nova 可以进行相应调整。

## 创建工具

通过工具配置定义工具，该配置包含工具数组，以及可选的工具选择参数。每种工具规范必须包括：

- 名称：工具的清晰标识符
- 描述：工具功能的简要说明
- 输入架构：定义必填参数与选填参数的 JSON 架构

### Example工具配置示例

```
tool_config = {
  "tools": [
    {
      "toolSpec": {
        "name": "calculator",
        "description": "A calculator tool that can execute a math equation",
        "inputSchema": {
          "json": {
            "type": "object",
            "properties": {
              "equation": {
                "type": "string",
                "description": "The full equation to evaluate"
              }
            },
            "required": ["equation"]
          }
        }
      }
    }
  ]
}
```

### 工具定义最佳实践

- 确保名称与描述明确传达工具的确切功能；避免使用语义过于相似的工具
- 在描述中包含关键差异化因素，帮助模型区分相似的工具
- 为获得最佳性能，将 JSON 架构限制为两层嵌套
- 使用架构类型（例如 enum、int、float）来限制输入，而不是用纯文本描述结构

- 使用 JSON 架构表示法标注必填参数与选填参数 ( 例如 "required": ["param1", "param2"] )
- 提交前使用标准验证器验证 JSON 架构
- 将长字符串参数置于架构末尾，避免嵌套

## 带约束解码的结构化输出

Amazon Nova 模型利用约束解码来确保生成的输出具有较高可靠性。此技术使用语法在每个生成步骤限制可能的词元，防止密钥无效，并根据您定义的架构强制使用正确的数据类型。

### Example结构化输出示例

```
tool_config = {
  "tools": [
    {
      "toolSpec": {
        "name": "ProductAnalysis",
        "description": "Analyze product information from text.",
        "inputSchema": {
          "json": {
            "type": "object",
            "properties": {
              "name": {
                "type": "string",
                "description": "Product name"
              },
              "rating": {
                "maximum": 5,
                "description": "Customer rating 1-5",
                "type": ["number", "null"],
                "minimum": 1
              },
              "features": {
                "description": "Key product features",
                "type": "array",
                "items": {"type": "string"}
              },
              "category": {
                "type": "string",
                "description": "Product category"
              },
              "price": {
                "type": "number",
```

```
        "description": "Price in USD"
      }
    },
    "required": ["name", "category", "price", "features"]
  }
}
}
],
"toolChoice": {
  "tool": {"name": "ProductAnalysis"}
}
}
```

## 工具选择选项

Amazon Nova 支持三个工具选择参数：

### Tool

指定工具将被调用一次，适用于结构化输出使用案例

### Any

至少调用所提供工具中的某一个，适用于 API 选择场景

### Auto

由模型决定是否调用工具以及调用工具的数量（默认行为）

## 调用工具

Amazon Nova 决定调用工具时，会将工具使用块作为助手消息的一部分返回，且 `stopReason` 设置为 `tool_use`。工具块包含工具的名称及其输入。

### Note

在单个 Python 会话中依次运行以下代码段：调用工具 → 处理工具调用 → 返回工具执行结果。  
如需再次运行示例，请重启 Python 会话。

## Example调用工具

```
import boto3
import json

# Create Bedrock client
bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')

# Complex calculation that benefits from precise computation
messages = [{
    "role": "user",
    "content": [{
        "text": "Calculate the compound interest on $10,000 invested at 4.75% annual
rate for 7 years, compounded quarterly. Use the formula  $A = P(1 + r/n)^{(nt)}$  where
 $P=10000$ ,  $r=0.0475$ ,  $n=4$ ,  $t=7$ "
    }]
}]

# Define tool configuration with calculator
tool_config = {
    "tools": [{
        "toolSpec": {
            "name": "calculator",
            "description": "Perform mathematical calculations",
            "inputSchema": {
                "json": {
                    "type": "object",
                    "properties": {
                        "expression": {
                            "type": "string",
                            "description": "Mathematical expression to evaluate"
                        }
                    }
                },
                "required": ["expression"]
            }
        }
    }]
}

# Invoke Model
response = bedrock.converse(
    modelId="us.amazon.nova-2-lite-v1:0",
    messages=messages,
```

```

    toolConfig=tool_config
)

# Extract tool use from response
tool = next(
    block["toolUse"]
    for block in response["output"]["message"]["content"]
    if "toolUse" in block
)

print(f"Tool: {tool['name']}")
print(f"Expression: {tool['input']['expression']}")

```

## 处理工具调用

从消息中提取工具名称与参数，然后调用该工具：

```

def calculate(expression):
    """Evaluate mathematical expression"""
    print(f"Calculating: {expression}")
    P = 10000
    r = 0.0475
    n = 4
    t = 7
    result = P * (1 + r/n) ** (n*t)
    return result

stop_reason = response["stopReason"]

if stop_reason == "tool_use":
    if tool["name"] == "calculator":
        result = calculate(tool["input"]["expression"])

```

## 返回工具执行结果

使用 ToolResultBlock 架构返回工具执行结果：

```

messages.append(response["output"]["message"])

# Add the tool result
messages.append({
    "role": "user",

```

```

    "content": [{
        "toolResult": {
            "toolUseId": tool['toolUseId'],
            "content": [{"json": {"result": result}}],
            "status": "success"
        }
    ]
}]
}))

# Send the tool result to the model
response = bedrock.converse(
    modelId="us.amazon.nova-2-lite-v1:0",
    messages=messages,
    toolConfig=tool_config
)

# Extract and display final response
final_text = next(
    block["text"]
    for block in response["output"]["message"]["content"]
    if "text" in block
)

print(f"\nFinal Response:\n{final_text}")

```

## 错误处理

将错误信息回传到 Amazon Nova，以便对请求进行修改并重试：

```

tool_result_message = {
    "role": "user",
    "content": [
        {
            "toolResult": {
                "toolUseId": tool["toolUseId"],
                "content": [{"text": "A validation exception occurred on field:
sample.field"}],
                "status": "error"
            }
        }
    ]
}

```

## 安全注意事项

- 在调用工具之前，请确认工具是否存在
- 确保输入的格式正确
- 在执行工具之前，请确认是否有适当的权限
- 基于会话详情进行处理，禁止 Amazon Nova 在工具调用中注入用户信息
- 注意：LLM 可能会虚构工具调用，务必在执行前进行校验

## 内置系统工具

Amazon Nova 2.0 模型包含完全托管的内置工具，无需自定义实现。通过 Converse API 中的简易开关即可启用这些工具。

### 代码解释器

通过代码解释器，Nova 可以在隔离的沙盒环境中安全地执行 Python 代码。此工具专为数学计算、逻辑运算和迭代算法而设计。

#### Note

代码解释器在 IAD、PDX 和 NRT AWS 区域中可用。为确保请求路由至支持的区域，请使用全局 CRIS。使用 Bedrock API 密钥时，需要手动将 InvokeTool 权限添加到策略定义。默认 Bedrock 角色不允许 InvokeTool 操作。

通过指定 systemTool 参数启用代码解释器：

```
import boto3
import json

bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')

tool_config = {
    "tools": [{
        "systemTool": {
            "name": "nova_code_interpreter"
        }
    }]
}
```

```

}

response = bedrock.converse(
    modelId="us.amazon.nova-2-lite-v1:0",
    messages=[{
        "role": "user",
        "content": [{
            "text": "What is the average of 10, 24, 2, 3, 43, 52, 13, 68, 6, 7, 902,
82"
        }]
    }],
    toolConfig=tool_config,
    inferenceConfig={"maxTokens": 10000, "temperature": 0}
)

# Pretty print the response
for block in response["output"]["message"]["content"]:
    if "toolUse" in block:
        print("=== Tool Use ===")
        print(f"Tool: {block['toolUse']['name']}")
        print(f"Code:\n{block['toolUse']['input']['snippet']}\n")

    elif "toolResult" in block:
        print("=== Tool Result ===")
        result = block['toolResult']['content'][0]['json']
        print(f"Output: {result['stdOut']}")
        if result['stdErr']:
            print(f"Error: {result['stdErr']}")
        print(f"Exit Code: {result['exitCode']}\n")

    elif "text" in block:
        print("=== Final Answer ===")
        print(block["text"])

```

该解释器在沙盒中运行代码，并以标准架构的形式返回结果：

```

{
    "stdOut": "String",
    "stdErr": "String",
    "exitCode": "int",
    "isError": "boolean"
}

```

## Web Grounding

通过 Web Grounding 功能，Amazon Nova 能够访问互联网中的实时信息，从而提供最新的回复并减少幻觉。通过指定 `nova_grounding` 系统工具来启用：

```
tool_config = {
  "tools": [{
    "systemTool": {"name": "nova_grounding"}
  ]
}
```

有关 Web Grounding 的详细信息，请参阅 [the section called “Web Grounding”](#)。

## 模型上下文协议 (MCP)

模型上下文协议 (MCP) 是一种开放标准，支持在数据来源和人工智能驱动的工具之间建立安全的双向连接。可运行 MCP 服务器，让 Amazon Nova 自动通过客户端网桥发现其工具，而不必为每个 API 或服务编写自定义适配器。

连接后，Amazon Nova 会像其他任何外部集成一样对待 MCP 工具：决定何时调用、发送所需的参数以及将结果纳入其回复中。通过内置的 MCPClient，搭配 Amazon Nova 与 Strands 使用可简化操作，该客户端能自动管理发现、连接与结果映射。

## 构建人工智能代理

Amazon Nova 模型针对使用 Amazon Nova Act 构建人工智能代理进行了优化。这些模型优化了工具使用能力，提升了多步骤任务推理效果，增强了在复杂代理工作流程中维持上下文的能力，并支持远程 MCP 工具。

## 创建代理

使用 Nova 构建的人工智能代理可协调多次工具调用，在持续交互过程中保持上下文，并在需要时进行纠错调整。扩展思考能力通过对复杂目标开展系统性推理，可优化代理工作流程。建议采用 Strands Agents 等规划框架 SDK，使代理系统的规划与执行流程更加稳定可靠。

## 代理设计模式

使用 Nova 设计代理时：

- 对于需要规划与验证的复杂多步骤工作流程，建议将推理等级设为中级或高级，以便获得最佳效果

- 启用 auto 工具选择，支持在代理交互过程中灵活选用工具
- 设计错误处理机制，使代理能够恢复执行并采用调整后的方案重试
- 保留对话历史，在代理交互全过程中维持上下文信息
- 针对代理系统处理的非受控内容，部署完善的内容过滤与审核机制。例如，Amazon 提供 Amazon Bedrock 护栏功能，可在多个基础模型、知识库和代理中应用安全防护策略。此类护栏可过滤有害内容、屏蔽违规话题，并对个人身份信息等敏感数据进行脱敏处理。

## 多工具代理示例

```
tool_config = {
  "tools": [
    #{
      "toolSpec": {
        "name": "calculator",
        "description": "Perform mathematical calculations",
        "inputSchema": {
          "json": {
            "type": "object",
            "properties": {
              "expression": {
                "type": "string",
                "description": "Mathematical expression to evaluate"
              }
            }
          },
          "required": ["expression"]
        }
      }
    },
    #{
      "toolSpec": {
        "name": "database_query",
        "description": "Query financial database for historical data",
        "inputSchema": {
          "json": {
            "type": "object",
            "properties": {
              "query": {
                "type": "string",
                "description": "SQL query to execute"
              }
            }
          }
        }
      }
    }
  ]
}
```



```

    # "text": "Design a scalable microservices architecture for an e-commerce
platform handling 1M+ daily transactions. Consider data consistency, fault tolerance,
performance, security, and cost optimization."
    #}]
#}],
#inferenceConfig={"maxTokens": 10000, "temperature": 10},
#additionalModelRequestFields={
  #"reasoningConfig": {
    #"type": "enabled",
    #"maxReasoningEffort": "low"
  #}
#}
)

# Process the streaming response
reasoning_complete = False
for event in response["stream"]:
    #if "contentBlockDelta" in event:
        #delta = event["contentBlockDelta"]["delta"]

        #if "reasoningContent" in delta:
            #reasoning_text = delta["reasoningContent"]["reasoningText"]["text"]
            #print(f"{reasoning_text}", end="", flush=True)
        #elif "text" in delta:
            #if not reasoning_complete:
                #print(f"
Final Architecture Design:
")
    #reasoning_complete = True
    #print(f"{delta['text']}", end="", flush=True)

```

## 代理状态管理

维护对话历史与工具执行结果以保留上下文；以下示例仅演示单轮对话场景，但开发人员可根据 workflow 需求自行编排整个代理系统。此外，Strands 等 Amazon Web Services 工具可代表开发人员管理代理上下文和工具状态。

```

messages = []

messages = []

# Initial user query

```

```
messages.append({
    # "role": "user",
    # "content": [{"text": user_query}]
})

# Get agent response
response = client.converse(
    # modelId=" us.amazon.nova-2-lite-v1:0",
    # messages=messages,
    # toolConfig=tool_config,
    # inferenceConfig=inf_params
)

# Add assistant response to history
messages.append(response["output"]["message"])

# Process tool calls and add results
if response["stopReason"] == "tool_use":
    # tool = next(
    #     # block["toolUse"]
    #     # for block in response["output"]["message"]["content"]
    #     # if "toolUse" in block
    # )

    ## Execute tool
    # result = execute_tool(tool["name"], tool["input"])

    ## Add tool result to conversation
    # messages.append({
    #     # "role": "user",
    #     # "content": [{
    #         # "toolResult": {
    #             # "toolUseId": tool["toolUseId"],
    #             # "content": [{"json": result}],
    #             # "status": "success"
    #         }
    #     }
    #     #}]
    # })

    ## Continue conversation
    # response = client.converse(
    #     # modelId=" us.amazon.nova-2-lite-v1:0",
    #     # messages=messages,
    #     # toolConfig=tool_config,
```

```
#inferenceConfig=inf_params
#)
```

## 代理最佳实践

有关代理最佳实践的更多信息，请参阅[the section called “最佳实践”](#)。

有关开发对话式人工智能代理的指南，请参阅[语音转语音](#)。

## Web Grounding

Web Grounding 使 Amazon Nova 能够在网络上搜索当前信息，并提供带引文的回复。该功能适用于需要获取模型训练数据以外最新信息的查询。

### Web Grounding 的工作原理

为提示启用 Web Grounding 后，将会执行以下步骤：

1. 请求配置：应用程序向 Amazon Bedrock API 发送一条用户提示，并将 `nova_grounding` 作为 `systemTool` 启用。
2. 搜索与分析：模型确定是否需要进行搜索、一次或多次搜索相关信息、评估是否需要进行额外搜索以扩大理解范围或更深入地研究特定子主题。
3. 回复生成：Amazon Nova 会自动合成搜索结果中的信息，以生成基于实时信息的最终 API 响应，并附上对相关资料来源的引文。

### 如何使用 Web Grounding

如需查看使用 Web Grounding 的完整代码示例，请参阅“代码示例”部分。

要在结果中包含 Web Grounding，请在 `toolConfig` 块中指定以下 `systemTool` 参数：

```
import boto3
from botocore.config import Config

# Create the Bedrock Runtime client with extended timeout
bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
```

```
    config=Config(read_timeout=3600)
)

# Define the tool configuration
tool_config = {
    "tools": [{
        "systemTool": {
            "name": "nova_grounding"
        }
    }]
}

# Send the request
response = bedrock.converse(
    modelId="us.amazon.nova-2-lite-v1:0",
    messages=[{
        "role": "user",
        "content": [{"text": "What are the latest developments in quantum computing?"}]
    }],
    toolConfig=tool_config
)

# Extract text with interleaved citations
output_with_citations = ""
content_list = response["output"]["message"]["content"]
for content in content_list:
    if "text" in content:
        output_with_citations += content["text"]
    elif "citationsContent" in content:
        citations = content["citationsContent"]["citations"]
        for citation in citations:
            url = citation["location"]["web"]["url"]
            output_with_citations += f" [{url}]"

print(output_with_citations)
```

## 区域可用性

Web Grounding 目前仅面向美国区域开放，且仅支持美国 CRIS 配置文件。

## 响应结构

以下为响应示例。为简便起见，响应已经精简：

```
{
  "output": {
    "message": {
      "content": [
        {
          "text": "Recent quantum computing developments include...",
          "citationsContent": [
            {
              "location": {
                "web": {
                  "url": "https://example.com/quantum-news",
                  "domain": "example.com"
                }
              }
            }
          ]
        }
      ]
    }
  }
}
```

每个引文包含以下内容：

- `text`：模型生成响应中的一个文本片段。
- `citationsContent`：与文本片段相关的引文数据的主容器。
- `citations`：`citationsContent` 内的容器，用于存储引文的来源位置。
- `location`：`citations` 内的容器，用于存储引文的具体来源。
- `web`：`location` 内的容器，用于存储网页来源的详细信息。
- `url`：引文来源的完整网址（URL）。
- `domain`：来源 URL 的根域。

## Grounding 安全

您的数据永远不会离开 AWS 基础设施。模型生成的查询保留在 AWS 服务中，永远不会发送到外部互联网。我们依托大规模内部 Web 搜索索引与知识图谱，优先采用可信、优质的数据来源，并在数据接入阶段过滤恶意内容。最后，我们通过运行时过滤来保护您的应用程序，使其免受间接提示注入和错误信息的侵害（请注意，这种缓解措施对于非英语语言的效果较为有限）。

## 错误处理

请勿包含名称为 `nova_grounding` 的 `toolSpec` 条目。添加使用此名称的工具将会导致错误。

使用 Web Grounding 时可能出现的潜在错误列表如下：

- `malformed_tool_use`
- `max_tokens`
- `malformed_model_output`

## 内置工具所需的权限

为确保角色可以在 Amazon Bedrock 上访问 Web Grounding，您可以通过两种方式来实现：

1. 为 IAM 角色启用 `BedrockFullAccess` 权限：若角色已具备 `BedrockFullAccess` 权限，将自动获得 Web Grounding 访问权限。
2. 添加特定权限（如果需要）：如需更精细的访问控制，可将此策略添加到角色的 IAM 策略，并将账户 ID 替换为您的 AWS 账户 ID：

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["bedrock:InvokeTool"],
      "Resource": ["arn:aws:bedrock::{111122223333}:system-tool/
amazon.nova_grounding"]
    }
  ]
}
```

Web Grounding 将 `aws:requestedRegion` 条件键设置为“未指定”。如果您的现有策略或服务控制策略（SCP）强制执行此条件，则可能会出现访问权限问题。将该条件更新为允许“未指定”的 `requestedRegion`，即可解决此问题。

**Note**

如果您启用了 Web Grounding 工具，则需要为您自己以及您的最终用户使用的包含所连接信息的输出负责。当您的输出包含来自引用或来源材料链接的连接信息时，系统将会提示您。您必须在提供给最终用户的输出中保留并显示这些引用和链接。

**Note**

使用 Web Grounding 会产生额外的成本。有关更多信息，请转到 [AWS Bedrock 定价](#)。

# 提示工程指南

提示工程是指通过设计指令，与 Amazon Nova 等大语言模型 ( LLM ) 实现高效交互的技术。提示工程能让您控制模型行为、提升输出质量并构建可靠的人工智能应用程序。

高效的提示工程，在于精心构造输入，引导模型生成符合预期的输出。精妙的提示，应包含清晰的指令、相关的上下文及格式规范的输入数据。

## 什么是提示工程

提示工程是指优化对大型语言模型 ( LLM ) 的文本输入以改进输出并获得所需回复的做法。提示可以帮助 LLM 执行各种任务，包括分类、问题解答、代码生成、创意编写等。您向 LLM 提供的提示的质量可能会影响模型的回复质量。本节为您提供了开始使用提示工程所需的信息。此外，本文还介绍了相关工具，帮助您在 Amazon Bedrock 上使用大语言模型时，找到最适配自身使用案例的提示词格式

提示的有效性取决于所提供信息的质量和提示本身的技巧。提示可能包括说明、问题、上下文详细信息、输入和示例，以有效指导模型并提高结果质量。本文档阐述了优化 Amazon Nova 2 Sonic 系列模型性能的策略与方法。本文介绍的方法可通过多种方式组合使用，以增强其有效性。建议您通过实验，确定最贴合自身特定需求的实施方案。

## 开始使用提示工程

在开始提示工程之前，建议准备好以下元素，这样就可以针对自己的应用场景迭代开发出最优的提示：

### 定义使用案例

从四个维度定义使用案例：

- **任务**：明确您希望模型达成的目标。这决定了应使用的提示词技术。
- **角色**：指定模型在完成任务时应扮演的角色。Amazon Nova 模型支持三个角色 ( 系统、用户或助手 ) 。
- **响应风格**：根据受众设定模型的响应结构或风格，例如 JSON、Markdown 或对话式。
- **指令**：给出模型为满足成功标准而需遵循的具体指令。

### 制定成功标准

定义成功标准或评估指标。可提供标准列表或提供具体的评测指标，例如长度、BLEU 分数、ROUGE、格式、事实和忠诚度。

## 撰写提示词

撰写初始提示词，明确包含任务、角色、输出格式与具体指令。根据实际效果迭代优化。

提示词的有效性，取决于所提供信息的质量。

## 一般最佳实践

以下最佳实践主要适用于 Amazon Nova 文本模型，在遵循各模态专属最佳实践的基础上，也适用于其他模型。

有关如何提示多模态输入的更多信息，请参阅[多模态输入提示](#)。有关如何提示语音输入的更多信息，请参阅[语音对话提示](#)。

## 了解角色

Amazon Nova 模型允许您使用三个不同的角色来构造提示：系统、用户和助手。

- 系统（选填）：虽非必填，但可用于设定助手的整体行为规则。它还可以用来提供用户希望模型在整个对话中遵守的额外说明和指南。
- 用户：可根据需要附带上下文、任务、指令、预期输出及用户查询。
- 助手：有助于引导模型朝着预期的响应前进。

### 主题

- [创建精准提示](#)
- [聚焦提示的各个部分](#)
- [使用系统角色](#)
- [提供示例（少样本提示）](#)
- [工具调用系统](#)
- [进阶提示技巧](#)

## 创建精准提示

在提示工程中，创建具体的用户查询至关重要。精心设计的查询可引导 Amazon Nova 文本理解模型生成准确的相关回复。要设计此类查询，必须先在查询中提供上下文信息。所提供的上下文通过与清晰的目标用户情景保持一致，更好地定位模型，从而提升输出的相干性与一致性。

## 提供具体的上下文信息

若指令清晰、结构合理、简短且具体，模型将发挥最佳性能。此提示模板指导适用于用户提示和系统提示。

### 提示模板

```
# Use this to clearly define the task and job needed by the model
task_summary = f"""
## Task Summary:
{{Task summary}}
"""

# Use this to provide contextual information related to the task
context_information = f"""
## Context Information:
- {{Context and content information 1}}
- {{Context and content information 2}}
...
"""

# Use this to provide any model instructions that you want model to adhere to
model_instructions = f"""
## Model Instructions:
- {{ Other Model Instructions }}
...
"""

# Use this to provide response style and formatting guidance
response_style = f"""
## Response style and format requirements:
- {{Style and format requirement 1}}
- {{Style and format requirement 2}}
...
"""

# Concatenate to final prompt
final_prompt = f"""{task_summary}
{context_information}
{model_instructions}
{response_style}"""
```

## 示例：具体的上下文信息

以下是结合具体上下文信息重写的部分提示示例。添加更多上下文信息可帮助模型尽力实现预期行为。

### Unclear prompt

User:

```
Write me a meeting invite to the project team
```

Assistant:

```
Certainly! Here's a sample meeting invite for your project team:
```

```
---
```

```
**Subject:** Project Team Meeting - [Project Name]
```

```
**Date:** [Insert Date]
```

```
**Time:** [Insert Time]
```

```
**Duration:** [Insert Duration]
```

```
**Location:** [Insert Location] / [Zoom/Teams Link if Online]
```

```
**Agenda:**
```

```
1. **Welcome and Introductions**
```

```
- Quick round of introductions for new team members
```

```
2. **Project Overview**
```

```
- Objectives and goals
```

```
- Current status and milestones achieved
```

```
3. **Task Allocation and Deadlines**
```

```
- Detailed breakdown of tasks
```

```
- Assignment of responsibilities
```

```
- Discussion of deadlines
```

```
4. **Risk Management**
```

```
- Identification of potential risks
```

```
- Mitigation strategies
```

```
5. **Resource Allocation**
```

```
- Required tools and materials
```

- Budget review

6. **\*\*Q&A Session\*\***

- Open floor for questions and clarifications

7. **\*\*Next Steps and Closing\*\***

- Summary of action items
- Schedule for the next meeting

**\*\*Additional Notes:\*\***

- Please come prepared with any relevant documents or updates regarding your assigned tasks.
- If you anticipate any issues or require additional support, please let us know in advance so we can address them during the meeting.

**\*\*Contact Information:\*\***

For any questions or concerns before the meeting, please feel free to reach out to [Your Name] at [Your Email] or [Your Phone Number].

We look forward to a productive discussion and to kicking off our project with enthusiasm!

Best regards,

[Your Full Name]

[Your Position]

[Your Contact Information]

---

Feel free to customize this template to better fit your specific project and team needs!

Clear prompt

User:

Write me a meeting invite to the project team. **The meeting is about project planning, and should be held on October 16th, from 10 AM to 11 AM in Conference Room B. Please include an agenda that covers our progress on the project thus far, as well as any upcoming milestones and deadlines.**

## Assistant:

**\*\*Meeting Invitation: Project Planning Session\*\***

**\*\*Date:\*\*** October 16th

**\*\*Time:\*\*** 10:00 AM - 11:00 AM

**\*\*Location:\*\*** Conference Room B

**\*\*Agenda:\*\***

- \*\*Welcome and Introduction (10:00 AM - 10:05 AM)\*\***
  - Brief overview by the meeting facilitator.
- \*\*Project Progress Update (10:05 AM - 10:30 AM)\*\***
  - Review of completed tasks and achievements to date.
  - Detailed discussion on current project status and any challenges encountered.
- \*\*Upcoming Milestones and Deadlines (10:30 AM - 10:50 AM)\*\***
  - Overview of the next phase of the project.
  - Identification of key milestones and deadlines.
- \*\*Action Items and Responsibilities (10:50 AM - 11:00 AM)\*\***
  - Assignment of tasks for the upcoming period.
  - Confirmation of individual responsibilities and follow-up items.
- \*\*Q&A Session (11:00 AM)\*\***
  - Open floor for any questions or concerns from team members.

Please ensure your attendance to discuss and finalize our project planning for the next phase. Your insights and contributions are highly valued.

Best regards,

[Your Name]

[Your Position]

[Your Contact Information]

- **清晰性原则**：向模型提供提示时，需以人类可理解的方式（而非机器逻辑，如 JSON 样式的列表）进行表述，使指令尽可能清晰。
- **具体性原则**：为模型提供具体说明，明确任务内容、预期输出以及任何其他上下文信息，以便引导模型生成符合预期的回复。

## 指定输出格式和样式

如果希望 Amazon Nova 2 生成贴合具体需求的响应，可以指定输出的格式和样式。例如，只希望生成 JSON 样式的回复，或者限制回复长度（如最大词数、项目符号数、段落数）。此方式可帮助 Amazon Nova 2 更精准地理解需求，生成易于解析的相关输出。

通过加粗和强烈措辞（如“务必”“切勿”“必须”）来强调希望模型严格遵守的规则。例如，若希望回复仅包含一个 JSON 对象，可对模型下达如下指令。“必须仅以 JSON 格式回答，切勿使用其他格式。”

以下是结合具体上下文信息重写的部分提示示例。添加更多上下文信息可帮助模型尽力实现预期行为。

### 示例 1：补充额外细节

补充有关回复格式的额外细节可帮助模型达成预期结果。其他一些常用的回复格式包括 CSV、Markdown、YAML 和 XML

#### Prompt without style constraints

User:

```
Outline the different types of rooms available in a typical 4-star hotel, including their features.
```

Assistant:

```
In a typical 4-star hotel, you'll find a range of room types, each designed to cater to different guest needs and preferences. Here's an outline of the different types of rooms available along with their features:
```

```
### 1. Standard Room
```

- **Size:** Usually around 250-350 square feet.
- **Features:**
  - Queen or double bed.
  - Complimentary Wi-Fi.
  - Flat-screen TV with cable or satellite channels.
  - Mini-fridge.
  - Coffee maker with complimentary coffee and tea.
  - Desk and chair.
  - En-suite bathroom with shower, toilet, and toiletries.
  - Air conditioning/heating.

```
### 2. Deluxe Room
```

- **Size:** Typically larger than standard rooms, around 350-450 square feet.

- **Features:**
- King bed or two queen beds.
- Larger bathroom with a bathtub.
- Additional seating area.
- Enhanced amenities like bathrobes and slippers.
- Upgraded toiletries and sometimes a welcome gift.
- Access to a lounge or executive floor perks.

### ### 3. **Suite**

- **Size:** Varies significantly, often between 500-1,200 square feet.
- **Features:**
- Separate living area with a sofa bed or additional seating.
- Kitchenette or full kitchen in some cases.
- Larger, more luxurious bathroom with a bathtub and separate shower.
- Multiple TVs.
- Access to executive lounge.
- High-end toiletries and often complimentary breakfast.
- Often includes concierge services and in-room dining options.

### ### 4. **Family Room**

- **Size:** Larger, usually around 400-500 square feet.
- **Features:**
- One king bed and one sofa bed or two queen beds.
- Designed for families with children.
- Extra space for luggage.
- Child-friendly amenities like a small kitchenette or high chair.
- Family bathroom with amenities suitable for kids.

### ### 5. **Executive Room**

- **Size:** Typically around 350-450 square feet.
- **Features:**
- Business-oriented with a larger workspace.
- Executive desk and chair.
- Access to executive lounge with complimentary snacks and refreshments.
- King bed or two queen beds.
- Enhanced amenities like high-speed internet and sometimes a Nespresso machine.

### ### 6. **Junior Suite**

- **Size:** Usually around 450-600 square feet.
- **Features:**
- Separate bedroom and living area.
- King bed or two queen beds.
- Kitchenette or mini-fridge.
- Luxurious bathroom with a bathtub.

- Lounge access.
- High-end toiletries.

### ### 7. **Premium Room**

- **Size:** Larger than standard rooms, around 400-500 square feet.
- **Features:**
  - King bed or two queen beds.
  - Larger bathroom with a bathtub.
  - Enhanced amenities like a welcome drink and premium toiletries.
  - Access to premium services and sometimes a lounge.

### ### 8. **Connecting Rooms**

- **Size:** Two rooms that connect via an internal door.
- **Features:**
  - Suitable for families or groups.
  - Often includes one standard room and one suite.
  - Shared bathroom or en-suite bathrooms in each room.
  - Extra space for luggage and guests.

### ### 9. **Accessible Rooms**

- **Size:** Varies, typically similar to standard rooms.
- **Features:**
  - Designed for guests with disabilities.
  - Wider doorways and bathroom space.
  - Adapted fixtures and fittings.
  - Additional support services.

### ### 10. **Business Room**

- **Size:** Usually around 300-350 square feet.
- **Features:**
  - Focused on business travelers.
  - High-speed internet and large workspace.
  - Compact yet fully equipped with essentials.
  - Often includes a king bed.

Each room type in a 4-star hotel is designed to provide comfort, convenience, and luxury tailored to different guest needs, ensuring a pleasant stay for all types of travelers.

## Prompt with style constraints

User:

Outline the different types of rooms available in a typical 4-star hotel, including their features. **All responses MUST be in YAML format.**

Assistant:

```
```yaml
hotel_rooms:
- room_type: Standard Room
  features:
  - description: A comfortable room with essential amenities.
  amenities:
  - bed_type: Double bed or two single beds
  - bathroom: Private bathroom with shower
  - wifi: Complimentary high-speed internet
  - TV: Flat-screen TV with cable channels
  - mini_fridge: Available
  - tea_and_coffee_making_facilities: Yes

- room_type: Deluxe Room
  features:
  - description: An upgraded room with more space and enhanced amenities.
  amenities:
  - bed_type: King-size bed or two single beds
  - bathroom: Private bathroom with both shower and bathtub
  - wifi: Complimentary high-speed internet
  - TV: Larger flat-screen TV with cable channels
  - mini_fridge: Available
  - tea_and_coffee_making_facilities: Yes
  - free_breakfast: Included

- room_type: Suite
  features:
  - description: A luxurious room with separate living area and premium amenities.
  amenities:
  - bed_type: King-size bed or two single beds
  - bathroom: Private bathroom with both shower and bathtub
  - separate_living_area: Yes
  - wifi: Complimentary high-speed internet
  - TV: Large flat-screen TV with cable channels
  - mini_fridge: Available
  - tea_and_coffee_making_facilities: Yes
  - free_breakfast: Included
  - concierge_service: Available
```
```

```
- room_type: Family Room
features:
- description: A spacious room designed for families with additional bedding
options.
amenities:
- bed_type: Queen bed and rollaway bed or two single beds and rollaway bed
- bathroom: Private bathroom with shower
- wifi: Complimentary high-speed internet
- TV: Flat-screen TV with cable channels
- mini_fridge: Available
- tea_and_coffee_making_facilities: Yes

- room_type: Business Room
features:
- description: A room tailored for business travelers with enhanced workspace.
amenities:
- bed_type: Double bed or two single beds
- bathroom: Private bathroom with shower
- wifi: Complimentary high-speed internet
- TV: Flat-screen TV with cable channels
- workspace: Desk and chair
- mini_fridge: Available
- tea_and_coffee_making_facilities: Yes
- business_center_access: Available
...
```

## 示例 2：设定样式限制

在此实例中，带样式限制的提示提供了有关要遵循的限制条件及所需回复格式的全面信息。正如回复所示，优化后的提示更符合所要求的简洁性与项目符号列表结构。

### Prompt without style constraints

User:

Explain the process of checking in a guest at a luxury resort.

Assistant:

Checking in a guest at a luxury resort involves a meticulous and personalized process designed to ensure a seamless and memorable start to their stay. Here's a detailed overview of the steps involved:

### ### Pre-Arrival Preparation

#### 1. **Pre-Arrival Communication:**

- **Confirmation:** Verify guest reservations and confirm arrival details.
- **Special Requests:** Collect any special requests (dietary restrictions, room preferences, etc.) and ensure they are noted in the guest profile.
- **Documentation:** Ensure all necessary documentation is ready, such as welcome letters, room keys, and any special amenities.

#### 2. **Room Preparation:**

- **Cleaning:** Conduct a thorough cleaning and preparation of the guest room.
- **Amenities:** Stock the room with high-end toiletries, fresh linens, and any requested amenities.
- **Personal Touches:** Add personalized touches such as a welcome note, fresh flowers, or locally sourced treats.

### ### Arrival and Initial Interaction

#### 3. **Guest Arrival:**

- **Greeting:** Have staff ready to greet guests as they arrive. This may include a warm welcome at the entrance or a meet-and-greet in the lobby.
- **Transport:** Provide luxury transport options like a chauffeured car service to escort guests to the check-in desk.

#### 4. **Check-In Desk:**

- **Professionalism:** Ensure check-in staff are well-trained, professional, and exude hospitality.
- **Technology:** Utilize sophisticated check-in technology (like tablet kiosks) to streamline the process.
- **Personalized Welcome:** Greet guests with a warm welcome, acknowledge their special requests, and present them with a personalized welcome kit.

### ### Documentation and Formalities

#### 5. **Document Verification:**

- **Identification:** Verify identification and payment methods.
- **Membership Cards:** If applicable, verify any loyalty or membership cards for additional benefits.

## 6. **Registration:**

- **Form Filling:** Assist guests with any required paperwork.
- **Special Benefits:** Clearly explain any special resort benefits, upgrades, or exclusive access to amenities that the guest is entitled to.

### ### Room Assignment

## 7. **Room Presentation:**

- **Walk-Through:** Escort guests to their room and conduct a walk-through to ensure everything is in order and to address any immediate concerns.
- **Upgrades:** Present any room upgrades and explain the features and services available in the upgraded room.

## 8. **Key Delivery:**

- **Room Key:** Provide the room key, often a high-end key card or a luxury key fob.
- **Access Information:** Explain how to use the key card, including access to other amenities like the spa, pool, and gym.

### ### Additional Services

## 9. **Welcome Amenities:**

- **Amenity Kits:** Deliver welcome amenity kits (like a bottle of champagne, local delicacies, etc.).
- **Itinerary:** Provide a detailed resort map and itinerary, highlighting all available services and activities.

## 10. **Personal Concierge:**

- **Concierge Services:** Introduce the guest to the concierge service and explain how to request any special services, such as dining reservations, transportation, or local excursions.
- **Contact Information:** Provide contact information for key staff members.

### ### Final Touches

## 11. **Feedback:**

- **Satisfaction Survey:** At the end of the check-in process, ask for feedback to ensure the guest's expectations were met and to gather any immediate suggestions.

## 12. **Follow-Up:**

- **Immediate Issues:** Ensure any immediate issues or requests are addressed promptly.
- **Comfort Check:** Make follow-up calls or send messages to ensure the guest is settling in comfortably.

By following these steps, a luxury resort aims to provide an exceptional check-in experience that sets the tone for an unforgettable stay.

### Prompt with style constraints

User:

Explain the process of checking in a guest at a luxury resort. **Provide all answers in a numbered list format, with each point containing no more than 15 words, and no more than 6 bullet points.**

Assistant:

1. Greet guest warmly with a smile and offer assistance.
2. Verify reservation details and check guest's identification.
3. Confirm room assignment and amenities provided.
4. Explain resort services and amenities available.
5. Present room key and offer directions to guest's suite.
6. Wish guest a pleasant stay and be available for queries.

### 示例 3：限制回复格式

在此示例中，带回复格式限制的提示会强制命令模型仅回复 1-10 之间的类别编号，避免冗长回答。

### Prompt without response format

User:

In this task, you are given a text which is the body of a document. Your job is to classify the topic of the document into these categories: 1)Company, 2)Educational Institution, 3)Artist, 4)Athlete, 5)Office Holder, 6)Mean of transportation, 7)Building, 8)Natural place, 9)Village, 10)Animal.  
Text: Johangsan is a mountain of South Korea. It has an elevation of 954 metres.

Assistant:

To classify the topic of the given text, let's analyze the content step-by-step:

1. The text mentions "Johangsan," which appears to be a specific name.

2. The text states that "Johangsan is a mountain," which provides a clear indication of its nature.
3. It is further specified that it is located in "South Korea" and has an elevation of "954 metres."

Based on the information provided, the text describes a geographical feature, specifically a mountain. From the given categories, the most appropriate classification for this document is:

8) Natural place

The text does not mention any company, educational institution, artist, athlete, office holder, means of transportation, building, village, or animal. Therefore, the classification as a "Natural place" is the most fitting choice.

## Prompt with response format

User:

In this task, you are given a text which is the body of a document. Your job is to classify the topic of the document into these categories: 1)Company, 2)Educational Institution, 3)Artist, 4)Athlete, 5)Office Holder, 6)Mean of transportation, 7)Building, 8)Natural place, 9)Village, 10)Animal. **Your output should be the category number. DO NOT generate anything apart from numbers 1-10.**

Text: Johangsan is a mountain of South Korea. It has an elevation of 954 metres.

Assistant:

8

## 明确划分提示部分

除初始上下文外，结构化查询格式对优化与语言模型的交互至关重要。通过分隔符划分不同的输入部分可提升清晰度，增强 Amazon Nova 2 模型对查询结构的理解能力。另一种有效方法是使用编号列表或项目符号，这可将任务拆解为易于管理的步骤，明确需要采取的行动。这种结构化方法有助于减少歧义，提高所生成回复的准确性。

示例：创建分节提示

请注意，在“分节提示”列中，提示针对不同实体（如任务、上下文、要求）划分了不同部分，实现了更清晰的表达；相比之下，“未分节提示”列的指令则显得模糊。

## Unsectioned Prompt

User:

Write an article about climate change for high school students explaining basic concepts, current impacts, and individual actions to mitigate it. Make it informative and engaging with simple language, including statistics and real-world examples.

Assistant:

### Understanding Climate Change: What It Is, What It's Doing, and What You Can Do About It

#### Introduction

Climate change might sound like a big, complicated topic, but it's really about the way our planet's weather is changing over time. This change is mostly happening because humans are doing things that are making our Earth warmer. Let's break it down so you can understand what's going on and what you can do to help.

#### What Is Climate Change?

Climate change refers to long-term shifts in temperatures and weather patterns. These shifts may be natural, but since the 1800s, human activities have been the main driver of rapid climate change.

**\*\*Key Concept: Greenhouse Gases\*\***

To understand this, we need to talk about greenhouse gases. These gases, like carbon dioxide (CO<sub>2</sub>), methane (CH<sub>4</sub>), and nitrous oxide (N<sub>2</sub>O), trap heat in our atmosphere. Without these gases, Earth would be too cold for most life forms. But when we burn fossil fuels (like coal, oil, and natural gas) and cut down forests, we release too much CO<sub>2</sub> and other greenhouse gases, trapping more heat and making the planet warmer.

#### Current Impacts of Climate Change

Let's look at some real-world examples to see how climate change is affecting our world today.

**\*\*1. Rising Temperatures\*\***

Since the late 19th century, Earth's average surface temperature has risen about 2 degrees Fahrenheit (1 degree Celsius). Scientists predict that if we don't change our ways, it could rise by 3.6 to 4.3 degrees Fahrenheit (2 to 2.4 degrees Celsius) by the end of this century.

**\*\*2. More Extreme Weather\*\***

Climate change is making extreme weather events more frequent and severe. For example:

- **Hurricanes:** In the U.S., hurricane seasons are getting longer and more destructive. In 2020, there were 30 named storms, the most ever recorded.
- **Droughts:** Parts of California have experienced some of the worst droughts in history, affecting agriculture and water supplies.

**\*\*3. Melting Ice Caps and Rising Sea Levels\*\***

Glaciers and ice sheets are melting at alarming rates. This not only affects wildlife but also leads to rising sea levels. Since 1901, global sea levels have risen by about 8 inches (20 centimeters). By 2100, they could rise by another 1 to 4 feet (0.3 to 1.2 meters).

**\*\*4. Ocean Acidification\*\***

As CO2 levels rise, more of it is absorbed by the oceans, making them more acidic. This is particularly harmful to marine life, including coral reefs. Coral reefs are crucial for marine biodiversity but are now dying at an alarming rate.

**#### Individual Actions to Mitigate Climate Change**

You might think that since climate change is a huge problem, one person can't make a difference. But even small actions can add up to make a big impact.

**\*\*1. Reduce, Reuse, Recycle\*\***

Try to minimize waste by:

- **Reducing:** Buy fewer items and choose products with less packaging.
- **Reusing:** Use reusable bags, bottles, and containers.
- **Recycling:** Make sure to recycle paper, plastic, and glass properly.

**\*\*2. Conserve Energy\*\***

Small changes can lead to big savings:

- Turn off lights when you leave a room.
- Unplug devices that you're not using.
- Use energy-efficient bulbs and appliances.

**\*\*3. Use Less Meat\*\***

Livestock farming is a major contributor to greenhouse gas emissions. By eating less meat, especially beef, you can reduce your carbon footprint.

**\*\*4. Support Renewable Energy\*\***

If possible, support or switch to renewable energy sources like solar or wind power. Even advocating for these options in your community can make a difference.

**\*\*5. Educate Yourself and Others\*\***

Knowledge is power. The more you know about climate change, the more you can help others understand it. Share facts and encourage your friends and family to take action too.

#### #### Conclusion

Climate change is a pressing issue that affects everyone on the planet. By understanding its basics, recognizing its current impacts, and taking individual actions, you can play a crucial role in mitigating its effects. Remember, every small step counts, and together, we can make a big difference for our planet.

Let's start today and build a sustainable future!

## Sectioned Prompt

User:

**\*\*Task:\*\***

Write an article

**\*\*Context Section:\*\***

Topic: Climate Change

Target Audience: High School Students

Purpose: Educational Article

**\*\*Content Requirements:\*\***

- Explanation of basic concepts related to climate change.
- Examples of current impacts on ecosystems and human populations.
- Suggestions for individual actions to mitigate climate change.

**\*\*Tone and Style:\*\***

- Informative and engaging
- Use simple language
- Include statistics and real-world examples

Assistant:

# Understanding Climate Change: A Guide for High School Students

## ## Introduction

Climate change is one of the most pressing issues of our time. It affects our planet, our ecosystems, and our daily lives. Understanding what climate change is and how it impacts us can help us take action to make a difference. This article

will explain the basic concepts of climate change, provide examples of its current impacts, and suggest actions you can take to help mitigate it.

## ## What is Climate Change?

**Climate change** refers to long-term changes in global temperatures and weather patterns. These changes are primarily caused by human activities, especially the burning of fossil fuels like coal, oil, and natural gas. When we burn these fuels, we release large amounts of greenhouse gases, such as carbon dioxide (CO<sub>2</sub>) and methane (CH<sub>4</sub>), into the atmosphere. These gases trap heat from the sun, causing the Earth's temperature to rise. This is often referred to as the **greenhouse effect**.

### ### Basic Concepts:

- Greenhouse Gases**: These gases include carbon dioxide, methane, nitrous oxide, and fluorinated gases. They trap heat in the Earth's atmosphere.
- Global Warming**: This is the increase in Earth's average surface temperature due to rising levels of greenhouse gases.
- Climate Change**: Refers to the broader set of changes in the Earth's climate system, including global warming.

## ## Current Impacts of Climate Change

### ### On Ecosystems:

- Melting Ice Caps and Glaciers**: The polar ice caps and glaciers are melting at an alarming rate. For example, the Arctic sea ice is shrinking, which affects polar bears and other Arctic animals.
- Rising Sea Levels**: As ice melts, sea levels rise. This can lead to the loss of coastal habitats and increased flooding in coastal areas.
- Changing Weather Patterns**: Climate change can lead to more frequent and severe weather events such as hurricanes, droughts, and heatwaves. For example, the 2020 Australian bushfires were more intense and lasted longer than usual due to climate change.

### ### On Human Populations:

- Health Issues**: Rising temperatures can lead to more heat-related illnesses and the spread of diseases like malaria and dengue fever.

2. **Food Security**: Changes in climate can affect crop yields, leading to food shortages. For example, droughts in Sub-Saharan Africa have caused significant food insecurity.

3. **Displacement**: Rising sea levels and extreme weather events can force people to leave their homes, becoming climate refugees. For instance, communities in Bangladesh are being displaced by rising sea levels.

## ## Actions You Can Take

Even though climate change is a massive issue, there are many actions you can take to help mitigate its effects. Here are some simple steps you can start with today:

1. **Reduce, Reuse, Recycle**: By reducing waste, reusing items, and recycling, you can decrease the amount of trash that ends up in landfills and reduce greenhouse gas emissions.

2. **Save Energy**: Turn off lights when you don't need them, use energy-efficient appliances, and unplug devices when they're not in use. These small actions can make a big difference.

3. **Use Less Meat**: Livestock farming produces a significant amount of methane. By eating less meat, particularly beef, you can reduce your carbon footprint.

4. **Support Renewable Energy**: Advocate for or invest in renewable energy sources like solar and wind power. These sources produce energy without greenhouse gas emissions.

5. **Educate and Advocate**: Inform others about the impacts of climate change and advocate for policies that support environmental sustainability. You can join or support environmental clubs at school or participate in local clean-up efforts.

## ## Conclusion

Climate change is a complex issue with far-reaching impacts. However, understanding it and taking action can make a significant difference. By learning about the basic concepts, recognizing the current impacts, and implementing simple changes in our daily lives, we can all contribute to mitigating climate change. Remember, every small action counts, and together, we can create a more sustainable future for our planet.

## 聚焦提示的各个部分

Amazon Nova 2 模型可以通过将指令分成若干部分并引用这些特定部分，从而密切关注提示词中的特定内容。若提示使用 Markdown、XML 或其他结构进行了清晰的分节界定，模型就能留意到相关内容。例如，您可以使用 `##Section Name##` 定义相应部分的名称，并通过 `##Section Name##` 在提示中引用该部分。

您还可以利用此策略，限制模型在所生成的回复中显示输入提示中的部分内容。例如，若在输入提示中包含少量样本示例或指令，可使用 `##Instructions##` 或 `##Examples##` 等分隔符并添加行分隔符，同时提供明确指令（比如 DO NOT mention anything inside the `##Instructions##` or `##Examples##` in the response），以防模型在输出中重复这些部分的输入提示内容。

示例：分节界定

用户：

```
You're an expert Prompts creator. Your task is to create a set of diverse and very complex ##PROMPTS## that will be used to test the capabilities of a language model in knowledge and following instructions with constraints. Please create 10 ##PROMPTS##. You must strictly follow ##GUIDELINES##:
```

```
##GUIDELINES##
```

- Generate ##PROMPTS## similar to the structure and style of the given ##EXAMPLE PROMPTS##. Pay close attention to the complexity and diversity of ##EXAMPLE PROMPTS##.
- Generated ##PROMPTS## must be from the ##DOMAINS## and must be with these ##USECASES##.
- Each of the ##PROMPTS## needs to be unique and very complex.
- Each of the ##PROMPTS## must have more than 4 sentences and 1 constraint.
- Each of the ##PROMPTS## should have at least 70 words.
- Each of the ##PROMPTS## should have an answer that can be written in text.
- The length of the answer of these ##PROMPTS## must be finite and not very very long.
- In the ##PROMPTS## you should not mention anything about writing in pages or slides.
- Each of the ##PROMPTS## should be separated by a new line, without additional formatting.

```
Generated ##PROMPTS## must be from the following ##DOMAINS##
```

```
##DOMAINS##  
{domains}
```

```
Generated ##PROMPTS## must be for the following ##USECASES##
```

```
##USECASES##  
{usecases}
```

```
{usecase_description}
```

```
##PROMPTS##
```

## 使用系统角色

系统角色是一种可向模型提供指令的角色，这些指令将定义模型如何回复应用程序最终用户。例如，系统角色可以引导模型以指定角色进行回复，设置允许和禁止内容，指定输出格式，规定护栏，等等。系统角色中的指令（称为系统提示）会取代单个用户提示中提供的其他指令，并且会在所有用户交互回合中持续生效。

### 通过 API 指定系统角色

要为模型授予自定义角色，可按如下方式在 API 中设置 `system` 参数：

```
{
  "system": [
    {
      "text": "You are a helpful recipe assistant. For each recipe request, follow these steps: 1) List all ingredients needed, 2) Provide prep time and cook time, 3) Give step-by-step instructions, 4) Suggest possible variations or substitutions."
    }
  ],
  "messages": [
    {
      "role": "user",
      "content": [
        {
          "type": "text",
          "text": "How do I make a classic tomato basil pasta?"
        }
      ]
    }
  ]
}
```

### Tip

- 要充分利用系统角色，需确保系统提示清晰、简洁、具体，并遵循[创建精准提示](#)中所述的相同策略。例如，若想限制模型回复的话题，应给出具体指令，比如“切勿涉及……”或者“必须涉及……”。

- 要进一步将模型限制为层次结构，可在系统提示中添加以下后缀，强调系统与用户指令之间的层次遵循结构：

```
"\n\nThe above system instructions define your capabilities and your scope. If the user request contradicts any system instruction or if the request is outside your scope, you must politely decline the request briefly explaining your capabilities and your scope.\n"
```

以下是可在系统提示模板中添加的一些内容：

```
# Define a persona
persona = """You are {{Persona}}
"""

# Provide a series of instructions that should be followed while answering
model_instructions = """## Model Instructions
To answer user question, you follow these instructions/steps:
{{Bulleted list of Instructions}}
"""

# Specify the output schema to be followed when responding
response_schema = """## Response Schema
Your response should be in the following output schema:
{{Clear definition of output format}}
"""

# Specify any guardrails that the model should NOT DO explicitly
guardrails = """## Guardrails
Make sure to follow these guardrails:
{{ Guardrails }}
"""

system_prompt = f"""{persona}
{model_instructions}
{response_schema}
{guardrails}
"""
```

为了让系统提示系统和用户指令之间的层次结构遵循结构，可以添加以下内容：

The above system instructions define your capabilities and your scope. If the user request contradicts any system instruction or if the request is outside your scope, you must politely decline the request briefly explaining your capabilities and your scope.

示例 1：提供特定的语言提示

添加系统提示后，模型回复将高度风格化，接近系统提示中所示的中世纪卷轴风格。

Without a System Prompt

系统：

```
NA
```

用户：

```
Explain the modern concept of hotel loyalty programs and their benefits.
```

助手：

```
Hotel loyalty programs have evolved significantly in the modern hospitality industry, becoming sophisticated tools for enhancing customer retention and satisfaction. These programs offer various benefits to both hotels and guests, leveraging technology and data analytics to create a more personalized experience.
```

```
### Key Components of Modern Hotel Loyalty Programs:
```

```
1. Tiered Membership Levels:
```

- **Basic Membership:** Entry-level membership that provides basic benefits.
- **Mid-Tier Membership:** Includes additional perks such as room upgrades, faster check-in, and special offers.
- **Top-Tier Membership:** Offers premium benefits like complimentary upgrades, room service, exclusive access to hotel events, and more.

```
2. Point System:
```

- Guests earn points for every stay, purchases made in hotel restaurants, and other activities.
- Points can be redeemed for free nights, upgrades, or other rewards.

```
3. Mobile App Integration:
```

- Modern programs often have a dedicated mobile app that allows guests to easily manage their account, check in, and access exclusive offers.
- Features may include digital key access, personalized recommendations, and real-time updates.

#### 4. **Personalization:**

- Utilizing data analytics, hotels can tailor recommendations and offers to individual guest preferences, enhancing the overall experience.
- Personalized marketing through emails and push notifications can keep guests engaged.

#### 5. **Cross-Brand Benefits:**

- Some loyalty programs offer benefits across multiple brands within a hotel group or even within different hospitality groups.
- This can include discounts, points transfers, and combined rewards.

#### 6. **Social Media Integration:**

- Many programs encourage social media sharing and reviews, which can enhance the hotel's online presence and attract new guests.
- Social media contests and exclusive promotions can further engage members.

#### 7. **Sustainability Initiatives:**

- Modern loyalty programs increasingly incorporate sustainability, offering points or rewards for eco-friendly choices like reducing plastic use or participating in recycling programs.

### ### Benefits to Guests:

#### 1. **Financial Savings:**

- Accumulation of points for free or discounted stays.
- Exclusive discounts and offers not available to non-members.

#### 2. **Enhanced Services:**

- Faster check-in, preferred room assignments, and personalized services.
- Access to exclusive events and experiences.

#### 3. **Convenience:**

- Easy booking and management through mobile apps.
- Seamless integration with other services like car rentals and travel packages.

#### 4. **Recognition and Prestige:**

- Status symbols like elite membership levels.
- Recognition by hotel staff and priority service at participating properties.

### ### Benefits to Hotels:

#### 1. **Increased Guest Loyalty:**

- Encourages repeat visits and long-term relationships with guests.
- Helps retain customers even when faced with competition.

#### 2. **Data Collection and Analysis:**

- Provides valuable insights into guest preferences and behaviors.
- Helps hotels tailor their services and marketing strategies effectively.

#### 3. **Revenue Growth:**

- Drives up room occupancy and average daily rate (ADR) through repeat business.
- Cross-selling opportunities within the hotel (e.g., dining, spa services).

#### 4. **Market Differentiation:**

- Helps hotels stand out in a competitive market.
- Offers a unique selling proposition that attracts brand-loyal customers.

### ### Conclusion:

Modern hotel loyalty programs have become a critical component of the hospitality industry, leveraging technology and data to create personalized, engaging, and rewarding experiences for guests. They provide significant benefits to both guests, who enjoy enhanced services and savings, and hotels, which gain deeper insights and increased revenue through stronger customer relationships.

## With a System Prompt

系统 :

Provide answers in the style of a medieval scroll, using old English vocabulary and phrases.

用户 :

Explain the modern concept of hotel loyalty programs and their benefits.

助手 :

Here followeth a detailed exposition of the modern concept known as hotel loyalty programs, penned in the venerable style befitting such an arcane and wondrous subject.

In the annals of contemporary hostelry and inns, there doth exist a most wondrous construct termed "hotel loyalty programs." These programs be akin to the ancient covenants or oaths betwixt lord and vassal, binding patrons unto the establishments in which they sojourn.

Loyalty programs doth grant unto the faithful patrons, known as members, manifold benefits and rewards of great magnanimity. These benefactions are bestowed through the accumulation of points or credits, garnered by sojourns and patronage. Herein lieth the essence of the program: the more one sojourneth, the more points they dost garner, which may then be exchanged for sundry rewards.

The benefits of these programs are manifold and most enticing:

1. **Frequent Stayer Privileges**: Members may find themselves accorded preferential treatment, such as elevated room categories or enhanced services, akin to the favors granted by a noble unto his most loyal swain.
2. **Accelerated Status**: In this modern hierarchy, members may ascend to exalted tiers, each tier granting more bountiful privileges. Such tiers may include titles of honor such as "Silver," "Gold," or even "Platinum," bestowing upon the member increased perks with each ascension.
3. **Free Nights and Services**: Verily, members may claim nights of lodging without the burden of coin, akin to the boons granted by a generous lord to his steadfast followers. Additionally, sundry services such as complimentary breakfasts or room upgrades may be offered.
4. **Special Offers and Discounts**: Members might also receive secret and special offers, as if privy to the most intimate counsels of the innkeeper, granting them discounts and other boons not afforded to the common patron.
5. **Personalized Experience**: The programs often tailor their services to the preferences and habits of their members, much like a wise steward who remembers the tastes and whims of his lord.

Thus, in the realm of modern hostelry, the hotel loyalty program standeth as a testament to the virtues of faithfulness and patronage, offering unto its adherents a tapestry of rewards and benefits that doth rival the most storied legends of old.

## 示例 2：提供护栏

通过添加系统提示，用户可以为应用程序添加护栏，防止应用程序被用于非预期用途。以下示例说明了护栏如何引导模型专注于目标任务。

### Without a System Prompt

系统：

NA

用户：

Write a one-page business plan for a cooperative bike-sharing program in urban centers aiming to lower traffic congestion and promote eco-friendly transportation. (Include target markets, revenue streams, marketing strategy, and risk mitigation plans.)

助手：

**\*\*Urban EcoCycle: A Cooperative Bike-Sharing Program\*\***

**\*\*Executive Summary:\*\***

Urban EcoCycle is a cooperative bike-sharing program designed to alleviate traffic congestion and promote eco-friendly transportation in urban centers. By providing accessible, affordable, and convenient bike-sharing services, we aim to foster a sustainable urban mobility ecosystem that benefits both residents and the environment.

**\*\*Target Markets:\*\***

Our primary target markets include urban professionals, students, and environmentally conscious citizens. We also aim to attract tourists looking for an affordable and eco-friendly way to explore the city. Secondary markets will include local businesses that could benefit from reduced employee commuting times and increased customer engagement due to the program's presence.

**\*\*Revenue Streams:\*\***

- \*\*Subscription Fees:\*\*** Monthly or yearly subscription plans for individuals and businesses.
- \*\*Ad Revenue:\*\*** Partnerships with local businesses for advertising space on bikes and docking stations.
- \*\*Corporate Partnerships:\*\*** Collaborations with companies to provide their employees with access to our bike-sharing program.

4. **Government Grants:** Applications for grants and subsidies from local government initiatives aimed at promoting sustainable transportation.

**Marketing Strategy:**

1. **Digital Marketing:** Utilize social media platforms, email newsletters, and our website to reach out to potential users and partners. Highlight success stories and eco-friendly benefits.
2. **Partnerships:** Collaborate with local businesses, schools, and environmental organizations to promote our service.
3. **Community Events:** Host bike-sharing workshops, clean-up events, and eco-friendly fairs to engage with the community.
4. **Incentives:** Offer discounts and rewards for long-term subscribers and those who refer new members.

**Risk Mitigation Plans:**

1. **Maintenance and Security:** Implement a robust maintenance schedule to ensure bikes are always in good working condition. Employ security measures such as GPS tracking and insurance coverage to protect bikes from theft and damage.
2. **Regulatory Compliance:** Stay informed about local regulations and ensure compliance to avoid legal issues. Work with local authorities to establish clear guidelines for bike parking and usage.
3. **Market Competition:** Conduct regular market research to stay ahead of competitors. Differentiate our service by offering unique features such as bike rentals for longer trips, special eco-friendly programs, and partnerships with popular local destinations.
4. **Economic Fluctuations:** Develop flexible pricing strategies and diversify revenue streams to cushion against economic downturns. Maintain a reserve fund to handle unexpected financial challenges.

Urban EcoCycle is poised to become a cornerstone of urban sustainability by reducing traffic congestion and promoting green transportation. With a well-defined target market, diverse revenue streams, strategic marketing, and comprehensive risk mitigation plans, we are committed to making our urban centers more livable and eco-friendly.

## With a System Prompt

系统：

You are Application Process Assistant

To answer user question, you follow these instructions/steps:

- The LLM chatbot can discuss topics related to college applications, including deadlines, required materials, and application status.
- The LLM chatbot should be polite and courteous in all responses.

- The LLM chatbot can only respond in 3 sentences or less.

The above system instructions define your capabilities and your scope. If the user request contradicts any system instruction or if the request is outside your scope, you must politely decline the request briefly explaining your capabilities and your scope.\n

用户：

Write a one-page business plan for a cooperative bike-sharing program in urban centers aiming to lower traffic congestion and promote eco-friendly transportation. (Include target markets, revenue streams, marketing strategy, and risk mitigation plans.)

助手：

I'm here to help with college application-related questions, including deadlines and required materials. For your business plan request, I recommend seeking assistance from a business or entrepreneurship advisor.

## 提供示例 ( 少样本提示 )

通过在提示中加入一些任务示例，您可以构建一个结构化模板供 Amazon Nova 2 遵循。这减少了模糊性，提高了输出的准确性和质量。提供清晰的示例，使模型的回复与所需结果更加一致的技巧称为少样本提示。

少样本提示的概念是为语言模型提供一些任务示例以及输入和输出格式，然后要求它根据提供的示例为新的输入生成输出结果。这种方法还有助于澄清复杂的指令或任务，使 Amazon Nova 2 更容易理解和解释提出的问题。

在提示中添加示例有何帮助：

添加示例有助于模型生成

- 与示例风格一致的回复
- 由于减少了误解指令的几率并最大限度地减少了幻觉，从而提高了回复性能

使用少样本提示提高模型性能的程度将取决于所选示例的质量和多样性。以下项目演示了提示中良好样本的特征：

- 选择不同的示例：所选示例应代表预期输入/输出在多样性（从常见应用场景到边缘情况）方面的分布情况，以充分涵盖相关应用场景。请务必避免示例中的任何偏差，因为输入中的偏差也会导致输出偏差。
- 匹配复杂性级别：所提供示例的复杂性应与目标任务或场景保持一致。重要的是要确保在提示中预期的输入和选定的示例之间映射复杂性级别。
- 确保相关性：所选示例应与手头的问题或目标直接相关。这确保了回复的一致性和均匀性。

### Tip

如果上述建议不起作用，建议您构建一个基于 RAG 的系统，根据用户输入查询和可用样本池之间的相似之处，通过动态选择样本来增强提示。

### 示例 1：在用户提示中提供示例

在这里，您的用户正在尝试将传入的客户支持工单分类为预定义类别。您可以为 Amazon Nova 2 模型提供一些正确分类的工单示例，以提高其对工单中使用的类别和语言的理解。样本示例应简明扼要，并代表应用程序中针对的所有任务。如果有对任务有意义的特定字段名称（例如本示例中的 Ticket 或 Category），则您的用户可以直接使用这些字段，后跟冒号 ":"。

### 0-shot prompt

用户：

```
Please classify the following customer support tickets into one of the categories:
"Technical Issue," "Billing Inquiry," or "General Inquiry."
```

```
Ticket: "Can you explain why my latest bill is higher than usual? I haven't changed
my plan."
```

助手：

```
The customer support ticket should be classified as a "Billing Inquiry."
```

```
Here's the reasoning:
```

- The customer is asking about the reason for a change in their latest bill amount.
- The context revolves around billing and payment, which falls under the category of "Billing Inquiry."

## 3-shot prompt

用户：

Please classify the following customer support tickets into one of the categories: "Technical Issue," "Billing Inquiry," or "General Inquiry."

Use these examples to guide your classification:

**Ticket:** "I can't log in to my account. The system keeps giving me an error message."

**Category:** Technical Issue

**Ticket:** "I received an email saying my account will be suspended unless I make a payment. Is this a scam?"

**Category:** Billing Inquiry

**Ticket:** "What are the hours of operation for your customer support team?"

**Category:** General Inquiry

Now, classify the following ticket:

**Ticket:** "Can you explain why my latest bill is higher than usual? I haven't changed my plan."

助手：

Category: Billing Inquiry

Explanation: The ticket is asking for an explanation regarding a change in the billing amount, which falls under the billing category as it pertains to questions about charges or payments.

## 示例 2：提供示例作为回合提示

在此示例中，样本以回合形式提供，每个样本一回合，其中 User 角色包含输入，Assistant 角色包含该样本的预期输出。

## 0-shot Prompt

系统：

Your task is to Classify the following texts into the appropriate categories. The categories to classify are:

Categories:

- Food
- Entertainment
- Health
- Wealth
- Other

用户 :

I have 20\$ in my pocket.

助手 :

Category: Wealth

Explanation: The text mentions a specific amount of money, which falls under the category of wealth. The presence of a monetary value directly relates to financial status or economic aspect.

### 3-shot Prompt (as turns)

系统 :

Your task is to Classify the following texts into the appropriate categories. The categories to classify are:

Categories:

- Food
- Entertainment
- Health
- Wealth
- Other

用户 :

I love to eat pizza.

助手 :

Food

用户：

I enjoy watching movies.

助手：

Entertainment

用户：

I am going to the gym after this.

助手：

Health

用户：

I have 20\$ in my pocket.

助手：

Wealth

### 示例 3：在系统提示中提供示例

在此示例中，您可以指定“示例”部分，然后在系统提示中列出一系列示例。如果您的样本长且复杂，则这种在提示中提供样本的方式特别有用，并且可以帮助模型找到示例样本。

#### 0-shot Prompt

系统：

You are an AI assistant helping to draft professional reports for enterprise stakeholders. Ensure the report is formatted in rich text, with headings, bullet points, and highlighted important tasks.

用户：

Draft a detailed project update report for our stakeholders, summarizing the project milestones achieved, tasks completed, and future plans.

助手：

```
# Project Update Report

## Executive Summary
This report provides a comprehensive update on the project milestones achieved, tasks completed, and future plans. Our team has made significant progress, and we are on track to meet the project goals. The following sections outline the details of our accomplishments and upcoming activities.

---

## Project Milestones Achieved

### **1. Project Initiation**
- **Milestone Date:** July 15, 2023
- **Description:** Completed initial project planning and secured stakeholder approval.

### **2. System Design Completion**
- **Milestone Date:** September 30, 2023
- **Description:** Finished the detailed system design and received positive feedback from technical review meetings.

### **3. Prototype Development**
- **Milestone Date:** November 15, 2023
- **Description:** Developed and tested a working prototype of the system.

---

## Tasks Completed

### **1. Requirement Gathering**
- **Description:** Conducted comprehensive stakeholder interviews and collected detailed requirements.
- **Highlighted Important Tasks:**
  - **User Story Mapping:** Mapped out user stories to ensure all user needs are addressed.
  - **Requirement Documentation:** Created detailed requirement specifications.
```

```
### **2. System Architecture Design**
- **Description:** Designed the system architecture, focusing on scalability and
  maintainability.
- **Highlighted Important Tasks:**
  - **Technology Stack Selection:** Chose the appropriate technology stack based on
    project requirements.
  - **Architecture Diagrams:** Developed detailed architecture diagrams for easy
    understanding.

### **3. Prototype Development and Testing**
- **Description:** Developed and rigorously tested the prototype.
- **Highlighted Important Tasks:**
  - **Coding:** Implemented core functionalities of the prototype.
  - **Testing:** Conducted unit and integration testing to ensure prototype
    reliability.

---

## Future Plans

### **1. Full System Development**
- **Description:** Begin the full-scale development of the system based on the
  approved design and prototype.
- **Highlighted Important Tasks:**
  - **Module Development:** Develop each module as per the design specifications.
  - **Code Reviews:** Implement regular code reviews to maintain code quality.

### **2. User Acceptance Testing (UAT)**
- **Description:** Conduct UAT to ensure the system meets user requirements and is
  ready for deployment.
- **Highlighted Important Tasks:**
  - **Test Plan Development:** Create detailed UAT plans and test cases.
  - **UAT Execution:** Execute UAT and gather feedback for final adjustments.

### **3. Deployment and Training**
- **Description:** Deploy the system and provide necessary training to end-users.
- **Highlighted Important Tasks:**
  - **Deployment:** Roll out the system to the production environment.
  - **Training Sessions:** Organize training sessions for users to familiarize them
    with the new system.

---

## Conclusion
```

Our project has successfully achieved key milestones and completed significant tasks. With the current momentum, we are well-positioned to meet our future objectives. We remain committed to delivering a high-quality system that meets all stakeholder expectations.

---

**\*\*Prepared by:\*\***

[Your Name]

[Your Position]

[Date]

**\*\*Contact Information:\*\***

[Your Email]

[Your Phone Number]

---

This detailed update aims to keep stakeholders informed and reassured about the project's progress and future direction. For any queries or further information, please do not hesitate to contact me directly.

## 2-shot Prompt

系统：

You are an AI assistant helping to draft professional reports for enterprise stakeholders. Below are a few examples of well-formatted project update reports to guide your response.

<Example 1>

<b>Project Update Report</b><br>

<i>Date:</i> January 15, 2023<br>

<hr>

<b>Project Milestones Achieved:</b><br>

- Successfully launched the new CRM system.<br>
- Integrated the marketing automation tool.<br>
- Completed the employee training sessions.

<b>Tasks Completed:</b><br>

- Developed the user manual.<br>

```
- Conducted system testing.<br>
- Prepared the training materials.

<b>Future Plans:</b><br>
- Roll out the CRM system to all departments.<br>
- Schedule follow-up training sessions.<br>
- Monitor system performance and make necessary adjustments.

<hr>
<b>Prepared by:</b><br>
[Your Name]<br>
[Your Position]<br>
[Your Contact Information]

<Example 2>

<b>Project Update Report</b><br>
<i>Date:</i> February 20, 2023<br>
<hr>

<b>Project Milestones Achieved:</b><br>
- Launched the new website.<br>
- Completed the SEO optimization.<br>
- Organized the stakeholder meeting.

<b>Tasks Completed:</b><br>
- Designed the website layout.<br>
- Wrote the content for the website.<br>
- Tested the website functionality

<b>Future Plans:</b><br>
- Implement the analytics tracking.<br>
- Launch the social media campaign.<br>
- Gather feedback from stakeholders.

<hr>
<b>Prepared by:</b><br>
[Your Name]<br>
[Your Position]<br>
[Your Contact Information]
```

用户：

Draft a detailed project update report for our stakeholders, summarizing the project milestones achieved, tasks completed, and future plans.

助手：

```
<b>Project Update Report</b><br>
<i>Date:</i> October 5, 2023<br>
<hr>

<b>Project Milestones Achieved:</b><br>
- Successfully completed the implementation of the new enterprise resource planning (ERP) system.<br>
- Conducted and completed the first phase of the data migration process.<br>
- Successfully launched the new inventory management module.

<b>Tasks Completed:</b><br>
- Developed and finalized the comprehensive user training program, including detailed user manuals and video tutorials.<br>
- Conducted extensive system integration testing with all existing systems and ensured seamless interoperability.<br>
- Executed the initial data migration from the legacy system to the new ERP system, resulting in a 95% accuracy rate.

<b>Future Plans:</b><br>
- Roll out the new ERP system to all departments in the next two phases.<br>
- Schedule and conduct follow-up training sessions for all employees.<br>
- Implement the remaining modules of the ERP system, including financial management and human resources.<br>
- Establish a robust monitoring and support framework to ensure smooth system operation and address any issues promptly.

<hr>
<b>Prepared by:</b><br>
[Your Name]<br>
[Your Position]<br>
[Your Contact Information]

---
```

This detailed report provides a clear and concise overview of the project's progress, highlighting the milestones achieved, tasks completed, and future plans.

It ensures stakeholders are well-informed and can make strategic decisions based on the current status of the project.

## 工具调用系统

通过在请求中传递工具配置架构，即可在 Amazon Nova 模型上进行工具调用。模型提示可通过此工具配置得到增强，因此这也是开始优化工具调用系统的极具影响力的一步。

考虑以下关键原则：

- 工具定义应该清晰简洁。定义应易于理解，意图必须非常明显。
- 使用关键区分因素和边界条件来定义何时应使用某一个工具而不是其他工具。
- 输入参数类型很重要。需思考：它们有意义吗？通常会以这种方式使用它们吗？

使用“工具选择”来控制何时调用工具

“工具选择”参数允许您自定义模型调用工具的行为。我们建议使用此参数来精细控制调用哪些工具以及何时调用。

例如，对于结构化输出等应用场景，您可能需要在每次调用 Amazon Nova 时系统调用特定的工具。您可以将输出架构定义为工具，然后将“工具选择”设置为该工具的名称。

```
{
  "toolChoice": {
    "tool": {
      "name": "name_of_tool"
    }
  }
}
```

对于许多座席应用场景，您可能需要确保模型始终选择其中一个可用工具。为此，您可以将“工具选择”设置为 any，这样每次调用模型时，系统只会调用一个工具。

```
{
  "toolChoice": {
    "any": {}
  }
}
```

最后，如果在应用场景中是否调用工具很大程度上取决于对话上下文，您可以将“工具选择”设置为 auto。这是默认行为，并且“工具选择”完全由模型决定。

```
{
  "toolChoice": {
    "auto": {}
  }
}
```

## 进阶提示技巧

以下章节提供了进阶指导，帮助您了解如何提高提示质量以及利用扩展思考等关键功能。

### 使用推理模式

Amazon Nova 2 模型提供了一个可选的推理模式，该模式允许模型在响应之前系统地处理问题，从而增强模型解决复杂问题的能力。利用模型的推理模式是提高提示词准确性的有效方法。

何时使用：建议在复杂任务中使用推理模式，例如以下使用案例：

- 多步推理：数学证明、算法设计、系统架构
- 信息交叉引用：文档分析、选项比较、权衡评估
- 易错计算：金融建模、数据分析、复杂调试
- 约束规划：资源优化、依赖管理、风险评估
- 复杂分类：多标签分类、层次化分类体系、细微决策边界
- 工具调用场景：多步 API 工作流、数据库查询优化、协调系统集成

#### Note

有关推理模式的更多信息，请参阅[使用推理](#)。

### 采用自上而下的方法

当模型需要评测多种解题方法时，可指示其采用自上而下的方法。

- Amazon Nova 2 模型若先从整体框架入手，再将其拆解为更细致的子问题或执行步骤，效果会更佳。

- 明确明确引导模型先确定核心目标，再将其分解为可处理的模块，最后逐一细化各部分内容。
- 这种结构化方法有助于模型梳理思路，形成更连贯的推理链。

示例：

```
{{User query}}. Start with the big picture and break it down into progressively smaller, more detailed subproblems or steps.
```

## 引导思维链

虽然推理模式通过系统化解题提升了准确性，但在部分特定场景中，在非推理模式下采用思维链 ( CoT ) 提示会更贴合需求。

何时使用：

- 透明度与可审计性：如需查看、验证或审计模型的推理过程，CoT 可完整呈现每一步细节。这对受监管行业、高风险决策场景或需要记录答案逻辑的场景尤为关键。
- 自定义推理结构：通过 CoT 设定特定的推理模式与方法论。您可引导模型遵循组织内部决策框架、采用领域专属解题思路，或按指定顺序考量相关因素。
- 提示词开发与调试：在提示工程阶段，CoT 可帮助您理解模型的解题逻辑，定位推理失效点，并更高效地迭代优化提示词。
- 混合方案：可在开发阶段使用 CoT 完善提示词，待确认模型在特定使用案例下的表现符合预期后，再切换至推理模式进行生产部署。

### Note

并非所有任务都需要使用 CoT。对于较简单的任务，可直接使用模型自身的推理逻辑。

引导模型的 CoT 方向：

```
{{User query}} Please follow these steps:
```

1. {{Step 1}}
2. {{Step 2}}
- ...

## 利用长上下文窗口

Amazon Nova 2 模型支持 100 万个词元的上下文长度，并且在代码理解和长文档问答方面表现出色。随着上下文长度增加，模型性能（包括对系统提示的遵从度与工具使用能力）可能略有下降。

使用方法：

- 将长格式数据放在开头：将长文档和输入放在提示的开头附近。将它们放在查询、指令和示例之前。
- 将指令放在末尾：将指令放在提示的末尾。当首先提供上下文并在末尾提供指令时，模型表现最佳。
- 构建文档内容开始和结束标记：使用开始和结束标记（例如 DOCUMENT {idx} START 和 DOCUMENT {idx} END）来标识长文档的开始和结束，其中 {idx} 代表特定文档的索引。

示例模板：

```
// Provide your long inputs at the top of your prompt
BEGIN INPUT DOCUMENTS

DOCUMENT 1 START
{{Your document}}
DOCUMENT 1 END

END INPUT DOCUMENTS

// Then specify your query and instructions
BEGIN QUESTION
{{User query}}
END QUESTION

BEGIN INSTRUCTIONS
{{Instructions}}
END INSTRUCTIONS
```

## 基于参考文本生成有据回答

建议向模型提供与当前查询相关的可信信息。通常，此类信息与输入查询共同构成检索增强生成（RAG）系统的一部分。

- 在此过程中，相关的上下文文档或信息将添加到实际用户提示中，使模型基于可信内容生成相关且精准的回复。
- 通过指令要求 Amazon Nova 2 使用来自可信来源的参考文本进行回答，可以引导其根据所提供的材料撰写回复，并确保回复以准确的相关信息为依据，从而提升所生成内容的可靠性与可信度。

- 使用参考文本可有效规避幻觉，从而提升回复的整体质量与可信度。为最大限度减少幻觉，建议在模型指令中明确注明：DO NOT USE INFORMATION THAT IS NOT IN REFERENCE TEXTS!。

## 提示模板

System:

In this session, the model has access to search results and a user's question, your job is to answer the user's question using only information from the search results.

Model Instructions:

- DO NOT USE INFORMATION THAT IS NOT IN SEARCH RESULTS!

User: {Query}

Resource: Search Results: {Reference texts}

示例：提供依据上下文避免模型产生幻觉

此示例说明了依据上下文如何避免模型产生幻觉。

系统：

In this session, the model has access to search results and a user's question, your job is to answer the user's question using only information from the search results. Model Instructions:- DO NOT USE INFORMATION THAT IS NOT IN SEARCH RESULTS!

用户：

What were the economic impacts of the COVID-19 pandemic on the United States in 2020?

Resource: Search Results: In 2020, the United States experienced significant economic impacts due to the COVID-19 pandemic. The U.S. economy contracted by 3.5% in 2020, according to the Bureau of Economic Analysis. Unemployment rates surged to 14.7% in April 2020, the highest since the Great Depression, before gradually declining. Small businesses faced severe challenges, with millions of firms closing permanently. Additionally, consumer spending dropped sharply as people reduced non-essential expenditures and saved more. Government intervention played a critical role in mitigating these impacts through stimulus packages and support programs, such as the Paycheck Protection Program (PPP) for small businesses and direct payments to individuals. Despite these measures, the economic recovery remained uneven across different sectors and regions.

## 基于引文标记生成有据回答

针对长篇文档任务，建议先指示 Amazon Nova 2 模型依据文档对应章节引文生成回复，再执行后续任务。

这种方法有助于模型关注最相关的信息，避免被无关内容分散注意力。要求模型基于依据生成回复时，可引用的章节应预先进行编号。例如：Passage %[1]%、Passage %[2]%，依此类推。

示例：使用引文标记

### Example引文标记提示

```
You are an AI financial assistant. Your task is to find patterns and insights from
multi-year financial documents

Passage %[1]%
{{Your document}}

Passage %[2]%
{{Your document}}

## Task:
Analyze my LLC's reports across multiple years to identify significant performance
trends, segment growth patterns and strategic shifts.

## Context information:
- You have access to my LLC's annual financial reports (10-K) for multiple fiscal years
in PDF format
- These reports contain comprehensive financial data including income statements,
balance sheets, cash flow statements and management discussions
- The analysis should focus on year-over-year comparisons to identify meaningful trends
- I operate two business segments, one in Massachusetts and one in New York

Based on the provided Context, extract key financial metrics from each year's reports
phrases from the documents.
Place citations as inline markers (e.g., %[1]%, %[2]%, etc.) directly within the
relevant parts of the response
text. Do not include a separate citation section after the response.
## Response Schema:
%% (Extracted Financial Metrics)
%% (Extracted Financial Metrics)
%% (Extracted Financial Metrics)
...
```

在基于用户任务提取关键信息后，您可以使用提取的财务指标来回答相关问题，如下所示：

### Example使用提取的指标进行后续分析

```
## Task
Analyze my LLC's financial reports across multiple years to identify significant performance trends, segment growth patterns and strategic shifts.

{{extracted financial metrics}}

## Model Instructions:
- Organize data chronologically to identify meaningful trends
- DO compare segment performance across the five-year period
- DO identify significant strategic shifts or investments mentioned in management discussions
- DO NOT make speculative predictions beyond what is supported by the data
- ALWAYS note any changes in accounting practices or reporting methodologies that might affect year-over-year comparisons

## Response style and format requirements:
- Respond in markdown
- Structure the analysis with clear headings and subheadings
- Present key financial metrics in tabular format showing all five years side-by-side
- Include percentage changes year-over-year for all major metrics
- Create a section dedicated to visualizing the most significant trends (with descriptions of what would be shown in charts)
- Limit the executive summary to 250 words maximum
- Format segment analysis as separate sections with consistent metrics across all segments
- MUST include a Key Insights bullet-pointed list at the end of each major section
```

### 使用 Nova Web Grounding

Amazon Nova 2 模型无需通过提示词直接要求引文作为依据文本，而是内置了 Web Grounding 工具。启用后，Amazon Nova 2 模型将直接查询 Web 和 Amazon 的知识图谱，并在最终响应中附带引文依据。

要详细了解如何利用 Amazon Nova Web Grounding，请参阅《[Amazon Nova Web Grounding 用户指南](#)》。

## 生成结构化输出

为确保输出格式的一致性和结构化，您可以使用结构化输出，包括 XML、JSON、Markdown 等格式，或使用工具使用功能。

- 这种方法可使下游系统更有效地理解和解析模型生成的输出。
- 向模型提供明确指令，生成的回复将以符合预定义架构的方式呈现。

例如，若下游解析器希望 JSON 对象的键名遵循特定命名约定，您应该在提示末尾指定响应架构。此外，如果希望回复以 JSON 格式呈现，且不带任何前言文本，则须相应地向模型提供指令。即明确写明：“请仅输出 JSON 内容，切勿添加任何前置说明”，保证输出格式纯净。

### Tip

- 我们发现，将数据格式要求直接定义在架构本身中，模型的遵守效果最佳，优于通过示例进行指定（例如将日期格式设为 YYYY/MM/DD）。
- 对于最多包含 10 个键的简单 JSON 输出，可直接使用下方提供的架构。对于更复杂的架构，建议通过工具来定义架构。工具使用采用了一种名为“约束解码”的技术，可提升模型对这类复杂架构的遵从度。

## 常见格式架构

以下是常见格式架构的示例。

### JSON

```
JSON_format = """Write your response following the JSON format below:

```json
{
  "key1": "value1",
  "key2": "value2",
  key3: [{
    "key3_1": "value_3_1 written in YYYY/MM/DD format",
    "key3_2": "value_3_2 day of the week written in full form",
    ...}]```
"""
```

## XML

```
XML_format = """Write your response following the XML format below:
```

```
<output>
  <task>"task1"</task>
  <subtask>
    <task1_result> ( task 1 result )</task1_result>
    <task2_result> ( task 2 result )</task2_result>
    <task3_result> ( task 3 result )</task3_result>
  </subtask>
  <task>"task2"</task>
  <subtask>
    <task1_result> ( task 1 result )</task1_result>
    <task2_result> ( task 2 result )</task2_result>
    <task3_result> ( task 3 result )</task3_result>
  </subtask>
</output>
```

```
"""
```

## Markdown

```
markdown_schema = """Write your response following the markdown format below:
```

```
## Introduction
( 2-3 line intro)

## Design Guidance
(Bulleted list of design guidance)

## Step by Step Instructions on Execution
( Bulleted list of instructions with each with bold title.

## Conclusion
( conclusion )
```

```
"""
```

## 预填充助手内容

如果您在非推理模式下生成结构化输出，则可以通过预填充助手内容来引导模型的响应。

预填充可以提高非推理模式下输出格式的一致性。这有助于引导模型的行为，省略前置说明，并强制输出 JSON、XML 等指定格式。例如，若使用 { 或 ```json 预填充助手内容，则该输入可引导模型直接生成不带额外信息的 JSON 对象。

### Tip

若需明确提取 JSON 内容，常用方式为使用 ```json 进行预填充，并在 ``` 位置设置停止序列。此操作可确保模型输出能够以编程方式解析的 JSON 对象。

## Example 预填充助手内容

以下代码演示了如何使用 API 进行预填充：

```
import boto3
import json

# Create a Bedrock Runtime client.
client = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1"
)

request_body = {
    "system": [
        {"text": "You write JSON objects based on the given instructions"}
    ],
    "messages": [
        {
            "role": "user",
            "content": [{"text": "Provide details about the best selling full-frame cameras in past three years. Answer in JSON format with keys like name, brand, price and a summary."}]
        },
        {
            "role": "assistant",
            "content": [{"text": " Here is the JSON response: ```json"}]
        }
    ]
}
```

```
    ],
    "inferenceConfig": {
        "maxTokens": 1000,
    }
}

# Invoke the model and extract the response body.
response = client.invoke_model(
    modelId="amazon.nova-2-lite-v1:0",
    body=json.dumps(request_body)
)

model_response = json.loads(response["body"].read())
```

## 指定用于复杂架构的工具

另一种方法是在工具使用的架构部分中加入 pydantic JSON 架构，强制模型响应遵循指定架构。可为所提供架构指定工具选择，这样 Amazon Nova 就会根据所选工具的架构生成回复。如需了解关于如何使用工具的详情，请参阅[工具使用](#)。

## 示例

### 示例 1：添加输出架构

向“输出架构”部分添加恰当的架构可引导模型遵循该架构生成内容。然而，模型输出具有不确定性，可能与输出架构存在差异。

## Unspecific Prompt

用户：

```
Provide details about the best selling full-frame cameras in past three years.
Answer in JSON format with keys like name, brand, price and a summary.
```

助手：

```
```json
[
  {
    "name": "Sony Alpha A7R IV",
    "brand": "Sony",
    "price": "Approximately $3,500",
```

```

"summary": "The Sony Alpha A7R IV is renowned for its exceptional image quality,
thanks to a 61-megapixel full-frame sensor. It also features 4K video recording
and a robust build, making it a favorite among professional photographers and
videographers."
},
{
"name": "Canon EOS R5",
"brand": "Canon",
"price": "Approximately $3,900",
"summary": "The Canon EOS R5 combines high-resolution imaging with advanced video
capabilities. It houses a 45-megapixel sensor, 8K video recording, and impressive
autofocus performance, appealing to both photography and filmmaking professionals."
},
{
"name": "Nikon Z7 II",
"brand": "Nikon",
"price": "Approximately $2,800",
"summary": "The Nikon Z7 II offers a 45.7-megapixel full-frame sensor and is
praised for its high resolution and excellent dynamic range. It also supports 4K
video recording and features a versatile Z-mount lens system, making it a strong
contender in the full-frame market."
},
{
"name": "Fujifilm GFX 100S",
"brand": "Fujifilm",
"price": "Approximately $7,000",
"summary": "While not a traditional full-frame camera, the Fujifilm GFX 100S stands
out with its 102-megapixel medium format sensor. It is lauded for its stunning
detail and color reproduction, though it comes at a higher price point and is aimed
at specialized users."
}
]
` ``

```

## Specific Prompt

用户：

Provide details about the best selling full-frame cameras in past three years. You MUST answer in JSON format only. Write your response following the the format below:

```

[ {
  "name": "name goes here",
  "brand": "brand goes here",

```

```
"price": "price goes here",  
"summary": "summary goes here"  
}]
```

助手：

```
[  
  {  
    "name": "Nikon D850",  
    "brand": "Nikon",  
    "price": "$3,296.95",  
    "summary": "The Nikon D850 is a professional full-frame DSLR camera known for its exceptional image quality, 45.7-megapixel resolution, 7 fps continuous shooting, and 4K video capabilities."  
  },  
  {  
    "name": "Canon EOS 5D Mark IV",  
    "brand": "Canon",  
    "price": "$3,499.00",  
    "summary": "The Canon EOS 5D Mark IV is a popular full-frame DSLR appreciated for its 30.4-megapixel sensor, 4K video recording, robust build quality, and reliable autofocus system."  
  },  
  {  
    "name": "Sony A7R III",  
    "brand": "Sony",  
    "price": "$3,198.00",  
    "summary": "The Sony A7R III is a high-resolution full-frame mirrorless camera renowned for its 42.4-megapixel sensor, excellent low-light performance, and versatile lens compatibility."  
  },  
  {  
    "name": "Fujifilm GFX 100S",  
    "brand": "Fujifilm",  
    "price": "$5,999.99",  
    "summary": "The Fujifilm GFX 100S is a medium format mirrorless camera that stands out for its 102-megapixel sensor, high dynamic range, and exceptional color reproduction."  
  },  
  {  
    "name": "Sony A7 III",  
    "brand": "Sony",  
    "price": "$1,998.00",
```

```
"summary": "The Sony A7 III is a full-frame mirrorless camera popular among
enthusiasts and professionals for its 24.2-megapixel sensor, fast autofocus, and 4K
video capabilities."
}
]
```

## 示例 2：预填充模型回复

第二种方法是在助手消息中包含所需的初始文本，从而预填充模型的回复。在这种情况下，Amazon Nova 模型将从助手消息中断处继续回复。

提示：仅当未启用推理功能时，预填充才有效。

用户：

```
Provide details about the best selling full-frame cameras in past three years.
Your response should be in JSON format, with the following keys: name, brand, year,
price, summary.
```

助手 ( 预填充 )：

```
```json
```

助手：

```
{
  "cameras": [
    {
      "name": "Sony Alpha A7 III",
      "brand": "Sony",
      "year": 2018,
      "price": "$1999",
      "summary": "The Sony Alpha A7 III is a highly acclaimed full-frame mirrorless camera
known for its excellent image quality, advanced autofocus system, and 4K video
capabilities."
    },
    {
      "name": "Canon EOS R5",
      "brand": "Canon",
      "year": 2020,
      "price": "$3899",
```

```
"summary": "The Canon EOS R5 is a top-tier full-frame mirrorless camera that offers
stunning 45MP images, 8K video recording, and exceptional performance in various
lighting conditions."
},
{
  "name": "Nikon Z7 II",
  "brand": "Nikon",
  "year": 2020,
  "price": "$2999",
  "summary": "The Nikon Z7 II is a flagship full-frame mirrorless camera with a 45.7MP
sensor, superb image quality, and advanced video recording features."
},
{
  "name": "Fujifilm GFX 100S",
  "brand": "Fujifilm",
  "year": 2020,
  "price": "$7999",
  "summary": "The Fujifilm GFX 100S is a medium format mirrorless camera known for its
impressive 102MP sensor, beautiful film simulations, and versatile lens options."
},
{
  "name": "Sony Alpha A7R IV",
  "brand": "Sony",
  "year": 2019,
  "price": "$3499",
  "summary": "The Sony Alpha A7R IV features a 61MP sensor, excellent low-light
performance, and advanced video capabilities, making it a favorite among professional
photographers."
}
]
}
```

### 示例 3：指定使用工具

作为使用预填充技术进行结构化输出的替代方案，您可以将 Nova 的工具功能与 toolChoice 结合使用。在本示例中，模型将生成符合所提供 JSON 架构的工具调用，并按照定义的完整 JSON 格式调用指定工具。有关其他指导，请参阅“高级系统”。

用户：

```
From the below provided Query, extract the relevent entities
```

**Query:**

John works in BUILDING-0987 and have been incharge of product id 23564#. His performance have been excellent in past year and is up for a raise.

Use the `extract_ner` tool.

**ToolConfig :**

```
{
  "tools": [
    {
      "toolSpec": {
        "name": "extract_ner",
        "description": "Extract all the named entities. based on provided
input",
        "inputSchema": {
          "json": {
            "type": "object",
            "properties": {
              "entities": {
                "type": "array",
                "items": {
                  "type": "object",
                  "properties": {
                    "name": {
                      "type": "string",
                      "description": "The extracted entity name. This
should be a name of a person, place, animal or thing"
                    },
                    "location": {
                      "type": "string",
                      "description": "The extracted location name. This
is a site name or a building name like SITE-001 or BUILDING-003"
                    },
                    "product": {
                      "type": "string",
                      "description": "The extrcted product code, this is
generally a 6 digit alphanumeric code such as 45623#, 234567"
                    }
                  },
                  "required": [
                    "name",
                    "location",
```

```
        "product"
        ]
      }
    },
    "required": [
      "entities"
    ]
  }
}
}],
"toolChoice": {
  "tool": {
    "name": "extract_ner"
  }
}
}
```

## 多语言使用案例

Amazon Nova 2 模型已在超过 200 种语言上进行了训练，并为 15 种语言进行了优化。

### 主题

- [精准翻译提示词](#)
- [强制执行一致的书写规范](#)

### 精准翻译提示词

如需使用该功能进行短文本翻译（若干句子），可直接指示模型将文本译为指定目标语言。

#### Example 翻译提示

```
Translate the following text into {target language}. Please output only the translated text with no prefix or introduction: {text}
```

```
Translate the following sentence from {source_language} to {target language}: {text}
```

```
{text} How do you say this sentence in {target_language}
```

## 强制执行一致的书写规范

在基于字符的语言中，Amazon Nova 2 模型可能会使用源语言的字符集。您可采用如下提示来强制执行一致的输出。

### Example强制执行书写规范

```
When translating, ensure to use the correct orthography / script / writing convention of the target language, not the source language's characters
```

## 工具调用

### 代理系统

#### 主题

- [设置正确的推理参数](#)
- [考虑延迟要求](#)
- [工具调用指令应措辞精准](#)
- [使用“思考”指令](#)
- [工具调用顺序](#)
- [构建高质量工具架构](#)
- [创建子代理](#)
- [多模态输入的工具使用](#)
- [后续步骤](#)

## 设置正确的推理参数

工具调用要求模型输出高度特定的结构化内容，通过以下推理参数可提升效果：

- 非推理模式：Temperature：0.7；TopP：0.9
- 推理模式：Temperature：1；TopP：0.9

## 考虑延迟要求

### Tip

Amazon Nova 2 模型在启用或关闭推理的情况下都能进行工具调用。不过，推理模式对延迟有显著影响。

对于延迟敏感的应用程序，建议采用关闭推理模式进行优化，并尽可能简化所需的工具调用。将多步工作流程拆分为离散步骤，以减少模型对机械输出不必要参数的依赖。

### 工具调用指令应措辞精准

工具名称：。在工具调用系统中，通常会在系统提示中引用工具，以指示模型何时调用工具。在提示中引用工具时，建议直接使用工具名称，而非采用 XML 或 Python 风格的引用方式或示例。

### 优质工具引用示例

```
Use the 'run_shell_command' tool for running shell commands
```

### 劣质工具引用示例

```
Call run_shell_command() to run shell commands
```

### 使用“思考”指令

在所有思考有助于工具调用的场景中，建议使用推理模式，而非提示模型“在标签内思考”或使用“思考”类工具。

Amazon Nova 2 模型针对推理模式进行了大量训练，在推理模式下执行思维链任务可获得最优效果。

### 工具调用顺序

在需要同时使用内置工具与原生工具调用的场景中，模型会优先调用内置工具。

请勿在提示词中指示模型改变行为，而应在 workflow 设计中对此加以考量。

例如，若不希望模型使用内置工具，只需将其排除在 workflow 之外，即可避免模型倾向于调用此类工具。

### 构建高质量工具架构

工具架构是通过提示工程实现高效工具调用系统的核心环节之一。设计时需重点关注：架构本身应包含哪些内容、各元素的语义描述方式，以及系统提示在指令中如何引用工具与架构元素。

Amazon Nova 2 模型针对工具架构的简洁描述进行了专项优化。请保持描述简短精炼。

工具架构与系统提示指南：

工具架构应包含以下内容：

- 核心功能：工具实现的功能（建议 20 – 50 个字）
- 参数规范：各参数清晰描述（每个参数约 10 个字）
- 格式要求：数据类型（如枚举、整数、浮点数）、必填字段及有效值范围

系统提示应包含以下内容：

- 专门设立 #Tool Usage 章节，说明编排逻辑（何时及为何使用特定工具）与业务规则（条件逻辑、执行顺序及依赖关系）。
- 错误处理策略：添加 #Error Handling and Troubleshooting 章节，说明如何应对调用失败或意外输出
- 输出格式：详细说明结果的呈现方式

示例

```
You are a software engineering issue root cause analysis agent. You are tasked with reviewing a customer issue and examining the repository to identify a plan to resolve the issue.

# Core Mandates
- DO NOT update the original issue that was posted by the user. You only add additional comments to the reported issue if necessary

# Primary Workflows
1. Understand: Analyze the user's request and explore the codebase thoroughly using get_file_contents to grasp file structures and conventions.
2. Plan: Create a coherent, evidence-based plan for resolving the task and share it with the user following the format below

# Tool Usage
- Read the Issue: Always start by using the read_issue tool to get the details about the requested issue
- File Paths: Always end the file path with "/" if you are searching a directory using the get_file_contents tools
- Parallelism: Execute multiple independent tool calls in parallel when feasible
```

```
# Error Handling and Troubleshooting
- **File Exploration:** If you get an error that a file doesn't exist, try searching at
  the directory level first to validate the file path

# Output Formatting
Return your plan in markdown in the following format

## Issue
<Your root cause analysis of the issue>

## Resolution Plan
<your step by step plan of how to solve the issue>
```

## 创建子代理

若出现以下情形，建议搭建专用子代理，而非使用搭载大量工具的单一代理：

- 工具数量超过 20 个：工具集过大会增加管理难度，同时提升工具选择错误率
- 功能领域差异显著：工具可自然划分为不同类别（如数据检索、数据处理、报表生成）
- 架构复杂：参数层级深度超 3 – 4 层，或工具间存在复杂关联依赖
- 对话轮次过长：工作流常规超过 15 – 20 轮对话，专用子代理效果更优
- 性能下降：工具选择准确率降低、响应延迟升高

### Tip

MCP 服务器自带部分无法自主管控的工具与架构，仅需纳入工作流执行任务所需的必要工具即可。

## 多模态输入的工具使用

在多模态任务中，使用工具处理结构化任务（如信息提取、时间戳生成）并未提升准确率。

建议查阅“多模态输入提示”章节中的相关内容，参考提供的模板编写有效提示词。

## 后续步骤

- 有关多模态提示的信息，请参阅[多模态输入提示](#)。

## 多模态输入提示

以下章节为理解图像与视频提供了指导。有关音频相关提示，请参阅[the section called “语音对话提示”](#)一节。

### 多模态通用准则

#### 用户提示与系统提示

在多模态理解使用案例中，每个请求都应包含用户提示文本。系统提示为可选项，且仅支持文本内容。

系统提示可用于为模型指定角色，定义通用的个性与回复风格，但不建议用于详细的任务定义或输出格式说明。

在多模态使用案例中，将任务定义、指令及格式要求写入用户提示，效果会优于系统提示。

#### 内容顺序

发送到 Amazon Nova 的多模态理解请求，应包含一个或多个文件以及用户提示。用户文本提示需作为消息中的最后一项内容，始终置于图像、文档或视频内容之后。

```
message = {
  "role": "user",
  "content": [
    { "document|image|video|audio": {...} },
    { "document|image|video|audio": {...} },
    ...
    { "text": "<user prompt>" }
  ]
}
```

如需在用户提示中引用特定文件，可使用 text 元素为每个文件块前置定义标签。

```
message = {
  "role": "user",
  "content": [
    { "text": "<label for item 1>" },
    { "document|image|video|audio": {...} },
    { "text": "<label for item 2>" },
    { "document|image|video|audio": {...} },
    ...
    { "text": "<user prompt>" }
  ]
}
```

```
]
}
```

## 理解文档与图像

以下章节将指导如何为需要理解或分析图像和文档的任务编写提示。

### 从图像中提取文本

Amazon Nova 模型支持从图像中提取文本，该能力称为光学字符识别 (OCR)。为获得最佳效果，请确保输入模型的图像分辨率足够高，保证文字清晰可辨。

在文本提取使用案例中，建议采用如下推理配置：

- temperature：默认值 (0.7)
- TopP：默认值 (0.9)
- 请勿启用推理功能

Amazon Nova 模型可将提取的文本输出为 Markdown、HTML 或 LaTeX 格式。建议采用如下用户提示模板：

```
## Instructions
Extract all information from this page using only {text_formatting} formatting. Retain
the original layout and structure including lists, tables, charts and math formulae.

## Rules
1. For math formulae, always use LaTeX syntax.
2. Describe images using only text.
3. NEVER use HTML image tags `` in the output.
4. NEVER use Markdown image tags `![]()` in the output.
5. Always wrap the entire output in ``` tags.
```

输出内容会被完整或部分包裹在 Markdown 代码围栏 (```) 内。可使用类似如下代码去除代码围栏：

```
def strip_outer_code_fences(text):
    lines = text.split("\n")
    # Remove only the outer code fences if present
    if lines and lines[0].startswith("```"):
        lines = lines[1:]
        if lines and lines[-1].startswith("```"):
```

```
lines = lines[:-1]
return "\n".join(lines).strip()
```

## 从图像或文本中提取结构化信息

Amazon Nova 模型可从图像中提取信息并输出为机器可解析的 JSON 格式，该过程称为关键信息提取 (KIE)。要执行 KIE，需提供以下内容：

- JSON 架构：遵循 JSON 架构规范的正式架构定义。
- 以下任意一项或多项：文档文件、图像或文档文本

请求中的文档或图像必须始终置于用户提示之前。

在 KIE 使用案例中，建议采用如下推理配置：

- temperature : 0
- 推理：并非必需，但在仅输入图像或使用复杂架构时，启用推理可提升效果。

### 提示模板

#### 纯文档或纯图像输入

```
Given the image representation of a document, extract information in JSON format
according to the given schema.
```

```
Follow these guidelines:
```

- Ensure that every field is populated, provided the document includes the corresponding value. Only use null when the value is absent from the document.
- When instructed to read tables or lists, read each row from every page. Ensure every field in each row is populated if the document contains the field.

```
JSON Schema:
```

```
{json_schema}
```

#### 纯文本输入

```
Given the OCR representation of a document, extract information in JSON format
according to the given schema.
```

```
Follow these guidelines:
```

- Ensure that every field is populated, provided the document includes the corresponding value. Only use null when the value is absent from the document.
- When instructed to read tables or lists, read each row from every page. Ensure every field in each row is populated if the document contains the field.

JSON Schema:

```
{json_schema}
```

OCR:

```
{document_text}
```

## 文档或图像与文本输入

Given the image and OCR representations of a document, extract information in JSON format according to the given schema.

Follow these guidelines:

- Ensure that every field is populated, provided the document includes the corresponding value. Only use null when the value is absent from the document.
- When instructed to read tables or lists, read each row from every page. Ensure every field in each row is populated if the document contains the field.

JSON Schema:

```
{json_schema}
```

OCR:

```
{document_text}
```

## 检测图像中的物体及其位置

Amazon Nova 2 模型能够识别图像中的物体及其位置，这项任务有时也称为“图像定位”或“物体定位”。实际应用场景包括图像分析与标注、界面自动化、图像编辑等。

无论图像输入分辨率和宽高比如何，模型都使用坐标空间，将图像横向分为 1,000 个单位，纵向分为 1,000 个单位，其中 x:0 y:0 位于图像的左上角。

使用分别表示左、上、右和底部的 [x1, y1, x2, y2] 格式来描述边界框。二维坐标使用 [x, y] 格式表示。

在物体检测使用案例中，建议采用如下推理参数值：

- temperature : 0

- 请勿启用推理功能

提示模板：通用物体检测

建议采用如下用户提示模板。

使用边界框检测多个实例：

```
Please identify {target_description} in the image and provide the bounding box coordinates for each one you detect. Represent the bounding box as the [x1, y1, x2, y2] format, where the coordinates are scaled between 0 and 1000 to the image width and height, respectively.
```

使用边界框检测单个区域：

```
Please generate the bounding box coordinates corresponding to the region described in this sentence: {target_description}. Represent the bounding box as the [x1, y1, x2, y2] format, where the coordinates are scaled between 0 and 1000 to the image width and height, respectively.
```

使用中心点检测多个实例：

```
Please identify {target_description} in the image and provide the center point coordinates for each one you detect. Represent the point as the [x, y] format, where the coordinates are scaled between 0 and 1000 to the image width and height, respectively.
```

使用中心点检测单个区域：

```
Please generate the center point coordinates corresponding to the region described in this sentence: {target_description}. Represent the center point as the [x, y] format, where the coordinates are scaled between 0 and 1000 to the image width and height, respectively.
```

解析模型输出：

上面推荐的每个提示都将生成一个以逗号分隔的字符串，其中包含一项或多项边界框描述，形式类似如下内容。字符串结尾是否包含“.”可能略有不同。例如，[356, 770, 393, 872], [626, 770, 659, 878]。

可以使用正则表达式解析模型生成的坐标信息，如以下 Python 代码示例所示。

## 代码示例

```
def parse_coord_text(text):
    """Parses a model response which uses array formatting ([x, y, ...])
    to describe points and bounding boxes. Returns an array of tuples."""
    pattern = r"\(([^\[\]]*?)\)"
    return [
        tuple(int(x.strip()) for x in match.split(","))
        for match in re.findall(pattern, text)
    ]
```

## 代码示例

要将边界框的归一化坐标重新映射到输入图像的坐标空间，可使用类似如下 Python 示例的函数。

```
def remap_bbox_to_image(bounding_box, image_width, image_height):
    return [
        bounding_box[0] * image_width / 1000,
        bounding_box[1] * image_height / 1000,
        bounding_box[2] * image_width / 1000,
        bounding_box[3] * image_height / 1000,
    ]
```

## 提示模板：带位置信息的多类别物体检测

若需在图像中识别多类目标，可通过以下任一格式在提示中加入类别列表。

对于模型易于理解的通用类别，直接在方括号内列出类别名称（无需加引号）：

```
[car, traffic light, road sign, pedestrian]
```

对于含义细微、不常见或来自专业领域、模型可能不熟悉的类别，请在括号内为每个类别补充定义。由于该任务难度较高，模型效果可能会有所下降。

```
[taraxacum officinale (Dandelion - bright yellow flowers, jagged basal leaves, white
puffball seed heads), digitaria spp (Crabgrass - low spreading grass with coarse
blades and finger-like seed heads), trifolium repens (White Clover - three round
leaflets and small white pom-pom flowers), plantago major (Broadleaf Plantain - wide
oval rosette leaves with tall narrow seed stalks), stellaria media (Chickweed - low
mat-forming plant with tiny star-shaped white flowers)]
```

您可根据偏好的 JSON 输出格式，选用以下任一用户提示模板。

## 提示选项 1

Detect all objects with their bounding boxes in the image from the provided class list. Normalize the bounding box coordinates to be scaled between 0 and 1000 to the image width and height, respectively.

Classes: {candidate\_class\_list}

Include separate entries for each detected object as an element of a list.

Formulate your output as JSON format:

```
[
  {
    "class 1": [x1, y1, x2, y2]
  },
  ...
]
```

## 提示选项 2

Detect all objects with their bounding boxes in the image from the provided class list. Normalize the bounding box coordinates to be scaled between 0 and 1000 to the image width and height, respectively.

Classes: **{candidate\_class\_list}**

Include separate entries for each detected object as an element of a list.

Formulate your output as JSON format:

```
[
  {
    "class": class 1,
    "bbox": [x1, y1, x2, y2]
  },
  ...
]
```

## 提示选项 3

Detect all objects with their bounding boxes in the image from the provided class list. Normalize the bounding box coordinates to be scaled between 0 and 1000 to the image width and height, respectively.

```
Classes: {candidate_class_list}
```

Group all detected bounding boxes by class.

Formulate your output as JSON format:

```
{
  "class 1": [[x1, y1, x2, y2], [x1, x2, y1, y2], ...],
  ...
}
```

#### 提示选项 4

Detect all objects with their bounding boxes in the image from the provided class list. Normalize the bounding box coordinates to be scaled between 0 and 1000 to the image width and height, respectively.

```
Classes: {candidate_class_list}
```

Group all detected bounding boxes by class.

Formulate your output as JSON format:

```
[
  {
    "class": class 1,
    "bbox": [[x1, y1, x2, y2], [x1, x2, y1, y2], ...]
  },
  ...
]
```

#### 解析模型输出

输出将以 JSON 格式编码，可使用任意 JSON 解析库进行解析。

提示模板：截图 UI 边界检测

建议采用如下用户提示模板。

根据目标检测 UI 元素位置：

```
In this UI screenshot, what is the location of the element if I want to {goal}? Express the location coordinates using the [x1, y1, x2, y2] format, scaled between 0 and 1000.
```

根据文本检测 UI 元素位置：

```
In this UI screenshot, what is the location of the element if I want to click on
"{text}"? Express the location coordinates using the [x1, y1, x2, y2] format, scaled
between 0 and 1000.
```

解析模型输出：

对于上述所有界面边界检测提示，均可通过正则表达式解析模型生成的坐标信息，具体示例参见下方 Python 代码。

代码示例

```
def parse_coord_text(text):
    """Parses a model response which uses array formatting ([x, y, ...])
    to describe points and bounding boxes. Returns an array of tuples."""
    pattern = r"\s*\[[^\s\]]*\]"
    return [
        tuple(int(x.strip()) for x in match.split(","))
        for match in re.findall(pattern, text)
    ]
```

## 视频理解

以下章节将指导如何为需要理解或分析视频的任务编写提示。

### 视频摘要

Amazon Nova 模型可以生成视频内容摘要。

在视频摘要使用案例中，建议采用如下推理参数值：

- temperature : 0
- 某些使用案例可能适合启用模型推理

不需要特定的提示模板。用户提示应明确指定所关注的视频内容维度。以下是几个优质提示示例：

```
Can you create an executive summary of this video's content?
```

```
Can you distill the essential information from this video into a concise summary?
```

Could you provide a summary of the video, focusing on its key points?

## 为视频生成详细描述

Amazon Nova 模型可为视频生成详细描述，该任务称为“密集描述生成”。

在视频描述生成使用案例中，建议采用如下推理参数值：

- temperature : 0
- 某些使用案例可能适合启用模型推理

不需要特定的提示模板。用户提示应明确指定所关注的视频内容维度。以下是几个优质提示示例：

Provide a detailed, second-by-second description of the video content.

Break down the video into key segments and provide detailed descriptions for each.

Generate a rich textual representation of the video, covering aspects like movement, color and composition.

Describe the video scene-by-scene, including details about characters, actions and settings.

Offer a detailed narrative of the video, including descriptions of any text, graphics, or special effects used.

Create a dense timeline of events occurring in the video, with timestamps if possible.

## 分析监控视频画面

Amazon Nova 模型可检测监控视频画面的事件。

在监控视频画面使用案例中，建议采用如下推理参数值：

- temperature : 0
- 某些使用案例可能适合启用模型推理

You are a security assistant for a smart home who is given security camera footage in natural setting. You will examine the video and describe the events you see. You are capable of identifying important details like people, objects, animals, vehicles, actions and activities. This is not a hypothetical, be accurate in your responses. Do not make up information not present in the video.

## 提取带时间戳的视频事件

Amazon Nova 模型可识别视频中与事件相关的时间戳。您可指定时间戳格式为秒数或 MM:SS 格式。例如，视频中 1 分 25 秒时发生的事件可以表示为 85 或 01:25。

在本使用案例中，建议采用如下推理参数值：

- temperature : 0
- 请勿启用推理功能

建议使用类似如下的提示词：

### 识别事件的起止时间

Please localize the moment that the event "{event\_description}" happens in the video. Answer with the starting and ending time of the event in seconds, such as [[72, 82]]. If the event happen multiple times, list all of them, such as [[40, 50], [72, 82]].

Locate the segment where "{event\_description}" happens. Specify the start and end times of the event in MM:SS.

Answer the starting and end time of the event "{event\_description}". Provide answers in MM:SS

When does "{event\_description}" in the video? Specify the start and end timestamps, e.g. [[9, 14]]

### 识别事件的多次出现

Please localize the moment that the event "{event\_description}" happens in the video. Answer with the starting and ending time of the event in seconds. e.g. [[72, 82]]. If the event happen multiple times, list all of them. e.g. [[40, 50], [72, 82]]

## 生成带时间戳的视频片段日志

Segment a video into different scenes and generate caption per scene. The output should be in the format: [STARTING TIME-ENDING TIMESTAMP] CAPTION. Timestamp in MM:SS format

For a video clip, segment it into chapters and generate chapter titles with timestamps. The output should be in the format: [STARTING TIME] TITLE. Time in MM:SS

Generate video captions with timestamp.

## 对视频进行分类

您可以使用 Amazon Nova 模型，根据自己提供的预定义类别列表对视频进行分类。

在本使用案例中，建议采用如下推理参数值：

- temperature : 0
- 请勿启用推理功能

使用如下提示模板：

```
What is the most appropriate category for this video? Select your answer from the options provided:  
{class1}  
{class2}  
{...}
```

示例：

```
What is the most appropriate category for this video? Select your answer from the options provided:  
Arts  
Technology  
Sports  
Education
```

# 代码和故障排除

本节将提供代码示例和故障排除指导，帮助您高效使用 Amazon Nova 模型。

## 代码库

本节将提供使用 Converse API 或 InvokeModel API 执行常见 Amazon Nova 操作的代码示例。

### Converse API 示例

#### 基本请求

使用 Converse API 向 Amazon Nova 模型发送基本文本请求。

#### Non-streaming

```
import boto3
from botocore.config import Config

# Create the Bedrock Runtime client, using an extended timeout configuration
# to support long-running requests.
bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(read_timeout=3600),
)

# Invoke the model
response = bedrock.converse(
    modelId="us.amazon.nova-2-lite-v1:0",
    messages=[
        {
            "role": "user",
            "content": [{"text": "Write a short story. End the story with 'THE
END'."}],
        }
    ],
    system=[{"text": "You are a children's book author."}], # Optional
    inferenceConfig={ # These parameters are optional
        "maxTokens": 1500,
        "temperature": 0.7,
```

```

        "topP": 0.9,
        "stopSequences": ["THE END"],
    },
    additionalModelRequestFields={ # These parameters are optional
        "inferenceConfig": {
            "topK": 50,
        }
    },
)

# Extract the text response
content_list = response["output"]["message"]["content"]
for content in content_list:
    if "text" in content:
        print(content["text"])

```

## Streaming

```

import boto3
from botocore.config import Config

# Create the Bedrock Runtime client, using an extended timeout configuration
# to support long-running requests.
bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(connect_timeout=3600, read_timeout=3600),
)

# Invoke the model
response = bedrock.converse_stream(
    modelId="us.amazon.nova-2-lite-v1:0",
    messages=[
        {
            "role": "user",
            "content": [{"text": "Write a short story. End the story with 'THE
END'."}],
        }
    ],
    system=[{"text": "You are a children's book author."}], # Optional
    inferenceConfig={ # These parameters are optional
        "maxTokens": 1500,
        "temperature": 0.7,
    }
)

```

```

        "topP": 0.9,
        "stopSequences": ["THE END"],
    },
    additionalModelRequestFields={ # These parameters are optional
        "inferenceConfig": {
            "topK": 50,
        }
    },
)

# Handle streaming events
for event in response["stream"]:
    if "contentBlockDelta" in event:
        delta = event["contentBlockDelta"]["delta"]
        if "text" in delta:
            print(delta["text"], end="", flush=True)

```

## 使用嵌入资产进行多模态输入

通过在请求中直接嵌入文档、图像、视频或音频数据来处理多模态内容。此示例使用图像数据。有关其他模态内容结构的详细信息，请参阅 Amazon Bedrock API 文档中的 [ContentBlock](#) 详细信息。

### Non-streaming

```

import boto3
from botocore.config import Config

# Read a document, image, video, or audio file
with open("sample_image.png", "rb") as image_file:
    binary_data = image_file.read()
    data_format = "png"

# Define message with image
messages = [
    {
        "role": "user",
        "content": [
            {
                "image": {
                    "format": data_format,
                    "source": {

```

```

        "bytes": binary_data # For Invoke API, encode as Base64
    string
        },
    },
    {"text": "Provide a brief caption for this asset."},
],
}
]

# Create the Bedrock Runtime client, using an extended timeout configuration
# to support long-running requests.
bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(read_timeout=3600),
)

# Invoke model
response = bedrock.converse(
    modelId="us.amazon.nova-2-lite-v1:0",
    messages=messages,
)

# Extract the text response
content_list = response["output"]["message"]["content"]
for content in content_list:
    if "text" in content:
        print(content["text"])

```

## Streaming

```

import boto3
from botocore.config import Config

# Read a document, image, video, or audio file
with open("sample_image.png", "rb") as image_file:
    binary_data = image_file.read()
    data_format = "png"

# Define message with image
messages = [
    {

```

```

    "role": "user",
    "content": [
        {
            "image": {
                "format": data_format,
                "source": {
                    "bytes": binary_data # For Invoke API, encode as Base64
string
                },
            },
            {"text": "Provide a brief caption for this asset."},
        ],
    ]
}
]

# Create the Bedrock Runtime client, using an extended timeout configuration
# to support long-running requests.
bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(connect_timeout=3600, read_timeout=3600),
)

# Invoke model with streaming
response = bedrock.converse_stream(
    modelId="us.amazon.nova-2-lite-v1:0",
    messages=messages,
)

# Handle streaming events
for event in response["stream"]:
    if "contentBlockDelta" in event:
        delta = event["contentBlockDelta"]["delta"]
        if "text" in delta:
            print(delta["text"], end="", flush=True)

```

## 使用 S3 URI 进行多模态输入

通过引用存储在 S3 中的文档、图像、视频或音频文件来处理多模态内容。此示例使用图像引用。有关其他模态内容结构的详细信息，请参阅 Amazon Bedrock API 文档中的 [ContentBlock](#) 详细信息。

## Non-streaming

```
import boto3
from botocore.config import Config

# Define message with image
messages = [
    {
        "role": "user",
        "content": [
            {
                "image": {
                    "format": "png",
                    "source": {
                        "s3Location": {
                            "uri": "s3://path/to/your/asset",
                            # "bucketOwner": "<account_id>" # Optional
                        }
                    },
                },
            },
            {"text": "Provide a brief caption for this asset."},
        ],
    }
]

# Create the Bedrock Runtime client, using an extended timeout configuration
# to support long-running requests.
bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(read_timeout=3600),
)

# Invoke model
response = bedrock.converse(
    modelId="us.amazon.nova-2-lite-v1:0",
    messages=messages,
)

# Extract the text response
content_list = response["output"]["message"]["content"]
for content in content_list:
    if "text" in content:
```

```
print(content["text"])
```

## Streaming

```
import boto3
from botocore.config import Config

# Define message with image
messages = [
    {
        "role": "user",
        "content": [
            {
                "image": {
                    "format": "png",
                    "source": {
                        "s3Location": {
                            "uri": "s3://path/to/your/asset",
                            # "bucketOwner": "<account_id>" # Optional
                        }
                    },
                },
            },
            {"text": "Provide a brief caption for this asset."},
        ],
    }
]

# Create the Bedrock Runtime client, using an extended timeout configuration
# to support long-running requests.
bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(connect_timeout=3600, read_timeout=3600),
)

# Invoke model with streaming
response = bedrock.converse_stream(
    modelId="us.amazon.nova-2-lite-v1:0",
    messages=messages,
)

# Handle streaming events
```

```
for event in response["stream"]:  
    if "contentBlockDelta" in event:  
        delta = event["contentBlockDelta"]["delta"]  
        if "text" in delta:  
            print(delta["text"], end="", flush=True)
```

## 扩展思考 ( 推理 )

为复杂问题求解任务启用扩展思考。

## Non-streaming

```
import boto3  
from botocore.config import Config  
  
# Create the Bedrock Runtime client, using an extended timeout configuration  
# to support long-running requests.  
bedrock = boto3.client(  
    "bedrock-runtime",  
    region_name="us-east-1",  
    config=Config(read_timeout=3600),  
)  
  
# Invoke the model  
response = bedrock.converse(  
    modelId="us.amazon.nova-2-lite-v1:0",  
    messages=[  
        {  
            "role": "user",  
            "content": [  
                {  
                    "text": 'How many capital letters appear in the following  
passage. Your response must include only the number: "Wilfred ordered an anvil from  
ACME. Shipping was expensive."' }  
            ],  
        }  
    ],  
    additionalModelRequestFields={  
        "reasoningConfig": {  
            "type": "enabled",  
            "maxReasoningEffort": "low", # "low" | "medium" | "high"        }  
    })
```

```

    }
  },
)

# Extract response content
content_list = response["output"]["message"]["content"]
for content in content_list:
    # Extract the reasoning response
    if "reasoningContent" in content:
        print("\n== Reasoning ==")
        print(content["reasoningContent"]["reasoningText"]["text"])
    # Extract the text response
    if "text" in content:
        print("\n== Text ==")
        print(content["text"])

```

## Streaming

```

import boto3
from botocore.config import Config

# Create the Bedrock Runtime client, using an extended timeout configuration
# to support long-running requests.
bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(connect_timeout=3600, read_timeout=3600),
)

# Invoke the model
response = bedrock.converse_stream(
    modelId="us.amazon.nova-2-lite-v1:0",
    messages=[
        {
            "role": "user",
            "content": [
                {
                    "text": 'How many capital letters appear in the following
passage. Your response must include only the number: "Wilfred ordered an anvil from
ACME. Shipping was expensive."'
                }
            ],
        }
    ]
)

```

```
    ],
    additionalModelRequestFields={
        "reasoningConfig": {
            "type": "enabled",
            "maxReasoningEffort": "low", # "low" | "medium" | "high"
        },
    },
)

# Process the streaming response
reasoning_output = ""
text_output = ""
for event in response["stream"]:
    if "contentBlockDelta" in event:
        delta = event["contentBlockDelta"]["delta"]

        if "reasoningContent" in delta:
            if len(reasoning_output) == 0:
                print("\n\n== Reasoning ==")
            reasoning_text_chunk = delta["reasoningContent"]["text"]
            print(reasoning_text_chunk, end="", flush=True)
            reasoning_output += reasoning_text_chunk

        elif "text" in delta:
            if len(text_output) == 0:
                print("\n\n== Text ==")
            text_chunk = delta["text"]
            print(text_chunk, end="", flush=True)
            text_output += text_chunk
```

## 内置工具：带引文的 Nova Grounding

使用 Nova Grounding 功能，从 Web 获取实时信息并附带引文。

### Non-streaming

```
import boto3
from botocore.config import Config

# Define the list of tools the model may use
tool_config = {"tools": [{"systemTool": {"name": "nova_grounding"}}]}
```

```
# Create the Bedrock Runtime client, using an extended timeout configuration
# to support long-running requests.
bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(read_timeout=3600),
)

messages = [
    {
        "role": "user",
        "content": [
            {"text": "What is the latest news about renewable energy sources?"}
        ],
    }
]

# Invoke the model
response = bedrock.converse(
    modelId="us.amazon.nova-2-lite-v1:0", messages=messages, toolConfig=tool_config
)

# Extract the text with interleaved citations
output_with_citations = ""
content_list = response["output"]["message"]["content"]
for content in content_list:
    if "text" in content:
        output_with_citations += content["text"]

    elif "citationsContent" in content:
        citations = content["citationsContent"]["citations"]
        for citation in citations:
            url = citation["location"]["web"]["url"]
            output_with_citations += f"[{url}]"

print(output_with_citations)
```

## Streaming

```
import boto3
from botocore.config import Config

# Define the list of tools the model may use
```

```
tool_config = {"tools": [{"systemTool": {"name": "nova_grounding"}}]}

# Create the Bedrock Runtime client, using an extended timeout configuration
# to support long-running requests.
bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(read_timeout=3600),
)

messages = [
    {
        "role": "user",
        "content": [
            {"text": "What is the latest news about renewable energy sources?"}
        ],
    }
]

# Invoke the model with streaming
response = bedrock.converse_stream(
    modelId="us.amazon.nova-2-lite-v1:0", messages=messages, toolConfig=tool_config
)

# Process the streaming response with interleaved citations
for event in response["stream"]:
    if "contentBlockDelta" in event:
        delta = event["contentBlockDelta"]["delta"]

        if "text" in delta:
            print(delta["text"], end="", flush=True)

        elif "citation" in delta:
            url = delta["citation"]["location"]["web"]["url"]
            print(f"[{url}]", end="", flush=True)
```

## 内置工具：代码解释器

使用代码解释器工具执行 Python 代码以进行计算和数据分析。

## Non-streaming

```
import boto3
from botocore.config import Config

# Define the list of tools the model may use
tool_config = {"tools": [{"systemTool": {"name": "nova_code_interpreter"}}]}

# Create the Bedrock Runtime client, using an extended timeout configuration
# to support long-running requests.
bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(read_timeout=3600),
)

messages = [
    {
        "role": "user",
        "content": [
            {
                "text": "What is the average of 10, 24, 2, 3, 43, 52, 13, 68, 6, 7,
902, 82?"
            }
        ],
    }
]

# Invoke the model
response = bedrock.converse(
    modelId="us.amazon.nova-2-lite-v1:0", messages=messages, toolConfig=tool_config
)

# Extract the text and the code the was executed
content_list = response["output"]["message"]["content"]
for content in content_list:
    if "text" in content:
        print("\n== Text ==")
        print(content["text"])

    elif "toolUse" in content and content["toolUse"]["name"] ==
"nova_code_interpreter":
        print("\n== Code Interpreter: input.snippet ==")
```

```
print(content["toolUse"]["input"]["snippet"])
```

## Streaming

```
import boto3
from botocore.config import Config
import json

# Define the list of tools the model may use
tool_config = {"tools": [{"systemTool": {"name": "nova_code_interpreter"}}]}

messages = [
    {
        "role": "user",
        "content": [
            {
                "text": "What is the average of 10, 24, 2, 3, 43, 52, 13, 68, 6, 7,
902, 82?"
            }
        ],
    }
]

# Create the Bedrock Runtime client, using an extended timeout configuration
# to support long-running requests.
bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(connect_timeout=3600, read_timeout=3600),
)

# Invoke the model with streaming
response = bedrock.converse_stream(
    modelId="us.amazon.nova-2-lite-v1:0", messages=messages, toolConfig=tool_config
)

# Process the streaming response
current_block_start = None
response_text = ""
for event in response["stream"]:
    if "contentBlockStart" in event:
        current_block_start = event["contentBlockStart"]["start"]
```

```
elif "contentBlockStop" in event:
    current_block_start = None

elif "contentBlockDelta" in event:
    delta = event["contentBlockDelta"]["delta"]

    if (
        current_block_start
        and "toolUse" in current_block_start
        and current_block_start["toolUse"]["name"] == "nova_code_interpreter"
    ):
        # This is code interpreter content
        tool_input = json.loads(delta["toolUse"]["input"])
        print("\n== Executed Code Snippet ==")
        print(tool_input["snippet"], end="", flush=True)

    elif "text" in delta:
        # This is text response content
        if len(response_text) == 0:
            print("\n== Text ==")
            text = delta["text"]
            response_text += text
            print(text, end="", flush=True)
```

## 工具使用

定义模型在对话期间可以使用的自定义工具。

### Non-streaming

```
import boto3
from botocore.config import Config

def get_weather(city):
    # Mock function to simulate weather API
    return {"temperatureF": 48, "conditions": "light rain"}

# Define the toolSpec for the weather tool
weather_tool = {
    "toolSpec": {
```

```
"name": "get_weather",
"description": "Get the current weather conditions in a given location",
"inputSchema": {
  "json": {
    "type": "object",
    "properties": {
      "city": {
        "type": "string",
        "description": "The city and state, e.g. San Francisco, CA",
      }
    },
    "required": ["city"],
  }
},
}

# Define the list of tools the model may use
tool_config = {"tools": [weather_tool]}

# Create the Bedrock Runtime client, using an extended timeout configuration
# to support long-running requests.
bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(read_timeout=3600),
)

# Start tracking message history
messages = []

messages.append(
    {
        "role": "user",
        "content": [
            {
                "text": "Suggest some activities to do in Seattle based on the
current weather."
            }
        ],
    }
)

# Invoke the model
```

```
response = bedrock.converse(
    modelId="us.amazon.nova-2-lite-v1:0", messages=messages, toolConfig=tool_config
)

assistant_message = response["output"]["message"]

# Add the assistant response to the message history
messages.append(assistant_message)

content_list = assistant_message["content"]
stop_reason = response["stopReason"]

if stop_reason == "tool_use":
    # Extract the toolUse details
    tool_use = next(
        content["toolUse"] for content in content_list if "toolUse" in content
    )
    tool_name = tool_use["name"]
    tool_use_id = tool_use["toolUseId"]

    if tool_name == "get_weather":
        # Call the tool
        weather = get_weather(tool_use["input"]["city"])

        # Send the result back to the model
        messages.append(
            {
                "role": "user",
                "content": [
                    {
                        "toolResult": {
                            "toolUseId": tool_use_id,
                            "content": [{"json": weather}],
                        }
                    }
                ],
            }
        )

    # Submit the tool result back to the model
    response = bedrock.converse(
        modelId="us.amazon.nova-2-lite-v1:0",
        messages=messages,
        toolConfig=tool_config,
```

```

    )

    content_list = response["output"]["message"]["content"]
    for content in content_list:
        # Extract the text response
        if "text" in content:
            print("\n== Text ==")
            print(content["text"])
else:
    # A tool call was not needed
    for content in content_list:
        # Extract the text response
        if "text" in content:
            print("\n== Text ==")
            print(content["text"])

```

## Streaming

```

import boto3
from botocore.config import Config
import json

def get_weather(city):
    # Mock function to simulate weather API
    return {"temperatureF": 48, "conditions": "light rain"}

# Define the toolSpec for the weather tool
weather_tool = {
    "toolSpec": {
        "name": "get_weather",
        "description": "Get the current weather conditions in a given location",
        "inputSchema": {
            "json": {
                "type": "object",
                "properties": {
                    "city": {
                        "type": "string",
                        "description": "The city and state, e.g. San Francisco, CA",
                    }
                }
            },
            "required": ["city"],

```

```
        }
    },
}

# Define the list of tools the model may use
tool_config = {"tools": [weather_tool]}

# Create the Bedrock Runtime client, using an extended timeout configuration
# to support long-running requests.
bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(read_timeout=3600),
)

# Start tracking message history
messages = []

messages.append(
    {
        "role": "user",
        "content": [
            {
                "text": "Suggest some activities to do in Seattle based on the
current weather."
            }
        ],
    }
)

# Invoke the model with streaming
response = bedrock.converse_stream(
    modelId="us.amazon.nova-2-lite-v1:0", messages=messages, toolConfig=tool_config
)

# Process the streaming response
assistant_message = {"role": "assistant", "content": []}
current_tool_use = None
stop_reason = None

for event in response["stream"]:
    if "contentBlockStart" in event:
        start = event["contentBlockStart"]["start"]
```

```
    if "toolUse" in start:
        current_tool_use = start["toolUse"]
        current_tool_use["input"] = ""

    elif "contentBlockDelta" in event:
        delta = event["contentBlockDelta"]["delta"]
        if "toolUse" in delta:
            current_tool_use["input"] += delta["toolUse"]["input"]
        elif "text" in delta:
            print(delta["text"], end="", flush=True)

    elif "contentBlockStop" in event:
        if current_tool_use:
            # Parse the accumulated tool input
            current_tool_use["input"] = json.loads(current_tool_use["input"])
            assistant_message["content"].append({"toolUse": current_tool_use})
            current_tool_use = None

    elif "messageStop" in event:
        stop_reason = event["messageStop"]["stopReason"]
        if stop_reason == "end_turn":
            exit

# Add the assistant response to the message history
messages.append(assistant_message)

if stop_reason == "tool_use":
    # Extract the toolUse details
    tool_use = next(
        content["toolUse"]
        for content in assistant_message["content"]
        if "toolUse" in content
    )
    tool_name = tool_use["name"]
    tool_use_id = tool_use["toolUseId"]

    if tool_name == "get_weather":
        # Call the tool
        weather = get_weather(tool_use["input"]["city"])

        # Send the result back to the model
        messages.append(
            {
                "role": "user",
```

```
        "content": [
            {
                "toolResult": {
                    "toolUseId": tool_use_id,
                    "content": [{"json": weather}],
                }
            }
        ],
    }
)

# Submit the tool result back to the model with streaming
response = bedrock.converse_stream(
    modelId="us.amazon.nova-2-lite-v1:0",
    messages=messages,
    toolConfig=tool_config,
)

# Handle the final streaming response
print("\n== Text ==")
for event in response["stream"]:
    if "contentBlockDelta" in event:
        delta = event["contentBlockDelta"]["delta"]
        if "text" in delta:
            print(delta["text"], end="", flush=True)
```

## InvokeModel API 示例

以下示例重点说明 Invoke API 与 Converse API 在请求和响应结构上略有差异的几个关键部分。在其他大多数方面，两个 API 均兼容，因此您可以轻松将上述 Converse API 示例适配至 InvokeModel API 使用。

### 基本请求

使用 InvokeModel API 向 Amazon Nova 2 模型发送基本文本请求。

### Non-streaming

```
import json

import boto3
from botocore.config import Config
```

```
# Configure the request
request_body = {
    "messages": [
        {
            "role": "user",
            "content": [{"text": "Write a short story. End the story with 'THE
END'."}],
        }
    ],
    "system": [{"text": "You are a children's book author."}], # Optional
    "inferenceConfig": { # These parameters are optional
        "maxTokens": 1500,
        "temperature": 0.7,
        "topP": 0.9,
        "topK": 50,
        "stopSequences": ["THE END"],
    },
}

bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(read_timeout=3600),
)

# Invoke the model
response = bedrock.invoke_model(
    modelId="us.amazon.nova-2-lite-v1:0", body=json.dumps(request_body)
)
response_body = json.loads(response["body"].read())

# Extract the text response
content_list = response_body["output"]["message"]["content"]
for content in content_list:
    if "text" in content:
        print(content["text"])
```

## Streaming

```
import json

import boto3
```

```
from botocore.config import Config

# Configure the request
request_body = {
    "messages": [
        {
            "role": "user",
            "content": [{"text": "Write a short story. End the story with 'THE
END'."}],
        }
    ],
    "system": [{"text": "You are a children's book author."}], # Optional
    "inferenceConfig": { # These parameters are optional
        "maxTokens": 1500,
        "temperature": 0.7,
        "topP": 0.9,
        "topK": 50,
        "stopSequences": ["THE END"],
    },
}

bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(connect_timeout=3600, read_timeout=3600),
)

# Invoke the model with streaming
response = bedrock.invoke_model_with_response_stream(
    modelId="us.amazon.nova-2-lite-v1:0", body=json.dumps(request_body)
)

# Process the streaming response
for event in response["body"]:
    chunk = json.loads(event["chunk"]["bytes"])
    if "contentBlockDelta" in chunk:
        delta = chunk["contentBlockDelta"]["delta"]
        if "text" in delta:
            print(delta["text"], end="", flush=True)
```

## 启用推理功能的 InvokeModel API

在启用推理功能的情况下使用 InvokeModel API 来解决复杂问题。

### Non-streaming

```
import json

import boto3
from botocore.config import Config

# Configure the request
request_body = {
    "messages": [
        {
            "role": "user",
            "content": [
                {
                    "text": 'How many capital letters appear in the following
passage. Your response must include only the number: "Wilfred ordered an anvil from
ACME. Shipping was expensive."'
                }
            ],
        }
    ],
    "reasoningConfig": {
        "type": "enabled",
        "maxReasoningEffort": "low", # "low" | "medium" | "high"
    },
}

bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(read_timeout=3600),
)

# Invoke the model
response = bedrock.invoke_model(
    modelId="us.amazon.nova-2-lite-v1:0", body=json.dumps(request_body)
)
response_body = json.loads(response["body"].read())

# Extract response content
```

```
content_list = response_body["output"]["message"]["content"]
for content in content_list:
    # Extract the reasoning response
    if "reasoningContent" in content:
        print("\n== Reasoning ==")
        print(content["reasoningContent"]["reasoningText"]["text"])
    # Extract the text response
    if "text" in content:
        print("\n== Text ==")
        print(content["text"])
```

## Streaming

```
import json

import boto3
from botocore.config import Config

# Configure the request
request_body = {
    "messages": [
        {
            "role": "user",
            "content": [
                {
                    "text": 'How many capital letters appear in the following
passage. Your response must include only the number: "Wilfred ordered an anvil from
ACME. Shipping was expensive."'
                }
            ],
        }
    ],
    "reasoningConfig": {
        "type": "enabled",
        "maxReasoningEffort": "low", # "low" | "medium" | "high"
    },
}

bedrock = boto3.client(
    "bedrock-runtime",
    region_name="us-east-1",
    config=Config(connect_timeout=3600, read_timeout=3600),
)
```

```
# Invoke the model with streaming
response = bedrock.invoke_model_with_response_stream(
    modelId="us.amazon.nova-2-lite-v1:0", body=json.dumps(request_body)
)

# Process the streaming response
for event in response["body"]:
    chunk = json.loads(event["chunk"]["bytes"])

    if "contentBlockDelta" in chunk:
        delta = chunk["contentBlockDelta"]["delta"]

        # Extract the reasoning response
        if "reasoningContent" in delta:
            print("\n== Reasoning ==")
            print(delta["reasoningContent"]["reasoningText"]["text"], end="",
                flush=True)

        # Extract the text response
        if "text" in delta:
            print("\n== Text ==")
            print(delta["text"], end="", flush=True)
```

## 问题排查

本节提供了在使用 Amazon Nova 模型时遇到的常见问题的解决方案。

### 身份验证和设置

#### 缺少权限

症状：无法访问 Nova 模型或功能

解决方案：

- 确保 IAM 角色具有 AmazonBedrockFullAccess 或相应的权限
- 通过 Amazon Bedrock 控制台请求特定的模型访问权限
- 验证模型访问权限和工具使用权限

## 访问模型被拒

症状：模型访问请求失败

解决方案：

- 通过 Amazon Bedrock 控制台请求特定的模型访问权限
- 验证是否已向账户授予对所请求模型的访问权限
- 检查模型的区域可用性

## 区域可用性问题

症状：所选区域中功能不可用

解决方案：

- Web Grounding 功能仅在使用美国 CRIS 配置文件的美国区域可用
- 验证所选区域中模型和功能是否可用
- 如有必要，切换到受支持的区域

## 超时配置


症状：请求未完成即超时

原因：默认超时时间过短，无法满足复杂操作

解决方案：配置更长的超时设置

```
from botocore.config import Config

bedrock = boto3.client(
    'bedrock-runtime',
    region_name='us-east-1',
    config=Config(
        read_timeout=3600      # 60 minutes
    )
)
```

 Note

Amazon Nova 推理请求最多可能需要 60 分钟才能执行复杂操作。

## API 响应问题

终止原因说明：

`end_turn`

正常完成。无需任何操作。

`max_tokens`

已达到词元限制。

解决方案：在 `inferenceConfig` 中增加 `maxTokens` 参数值。

`content_filtered`

内容违反 AWS 负责的人工智能政策。

解决方案：检查并修改输入内容，使其符合内容政策。

`malformed_model_output`

输出格式无效。

解决方案：检查输出架构和约束条件；确认 JSON 架构格式正确无误。

`malformed_tool_use`

工具调用格式无效。

解决方案：确认工具定义符合预期架构；检查工具输入参数格式是否正确。

`service_unavailable`

内置工具服务不可用。

解决方案：稍候重试请求；查看 AWS 服务运行状况控制面板。

`invalid_query`

对内置工具的查询无效。

解决方案：检查查询格式和参数；确保查询符合工具要求。

## max\_tool\_invocations

工具重试次数已用尽。

解决方案：简化任务或将其拆分为更小步骤；查看工具错误信息以定位具体问题。

## 推理模式错误

### 推理强度过高导致响应被截断

解决方案：进行高强度推理时，请取消以下参数设置：temperature、topP、maxToken。这允许模型为复杂的推理任务使用最佳设置。

### 推理所需词元不足

错误：maxTokens is insufficient

解决方案：提高上限后自动重试

```
token_limits = {
    "low": 15000,
    "medium": 30000,
    "high": 50000
}

try:
    response = client.converse(
        modelId="us.amazon.nova-2-lite-v1:0",
        messages=messages,
        inferenceConfig={
            "maxTokens": token_limits[max_effort]
        },
        additionalModelRequestFields={
            "reasoningConfig": {
                "type": "enabled",
                "maxReasoningEffort": max_effort
            }
        }
    )
except Exception as e:
    if "maxTokens is insufficient" in str(e):
        higher_limit = int(token_limits[max_effort] * 1.5)
        # Retry with higher limit
```

## 工具使用问题

### 模式验证失败

#### 工具架构验证错误

解决方案：

- 为获得最佳性能，将 JSON 架构限制为两层嵌套
- 确保所有必填字段均已正确定义
- 根据 JSON 架构规范验证架构

#### 模型未正确使用工具

解决方案：

- 确保工具名称清晰描述其用途
- 详细说明工具功能
- 明确定义输入架构并清晰描述参数
- 必要时在描述中附上示例

#### 工具调用行为不一致

解决方案：将工具调用的温度值设为 0：

```
inferenceConfig={
  "temperature": 0,
  "maxTokens": 10000
}
```

这将启用贪婪解码，提升工具使用的可靠性。

#### 工具选择冲突

问题：将自定义工具与 Web 搜索或代码解释器结合使用时出错

解决方案：切勿创建名称为 `nova_grounding` 的自定义 `toolSpec`，该名称会与系统工具冲突。请改用系统工具配置：

```
# Correct - use system tool
tool_config = {
  "tools": [{
    "systemTool": {"name": "nova_grounding"}
```

```
    ]]  
  }  
  
  # Incorrect - don't create custom tool with this name  
  # tool_config = {  
  #   "tools": [{  
  #     "toolSpec": {"name": "nova_grounding", ...}  
  #   }]  
  # }
```

## Web Grounding 问题

### 访问控制问题

问题：Web Grounding 和代码解释器无法正常工作

解决方案：确保 IAM 策略包含以下内容：

```
{  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": ["bedrock:InvokeTool"],  
      "Resource": ["arn:aws:bedrock::{YOUR_ACCOUNT_ID}:system-tool/  
amazon.nova_grounding"]  
    }  
  ]  
}
```

### 服务控制策略问题

问题：Web Grounding 被 SCP 阻止

解决方案：若服务控制策略中配置了 `aws:requestedRegion` 条件，请更新策略，为 Web Grounding 功能允许“未指定”区域。

## 媒体处理限制

### 对图像/视频中多语言内容理解不佳

限制：Nova 模型对视觉媒体中的多语言内容理解能力有限

解决办法：

- 在提供图像的同时附上文本翻译
- 尽可能使用基于文本的输入来处理多语言内容

## 人物识别

问题：模型拒绝识别图像中的人物

预期行为：出于隐私与安全考虑，模型会拒绝识别或标注图像、文档或视频中的个人身份

解决办法：询问整体特征或场景信息而非具体身份

## 空间推理限制

问题：定位或布局分析结果不准确

局限：精确空间推理能力有限

解决办法：

- 使用边界框检测进行物体定位
- 在提示中提供清晰的参考点
- 将复杂的空间查询拆分为更简单的部分

## 图像/视频中的小字

问题：无法读取媒体中的小字

解决方案：

- 裁剪图像，聚焦相关文字区域
- 提高源媒体分辨率
- 如有文字，请单独提供

## 文档与文件处理

### 不支持的内容

问题：PDF 处理失败

原因：

- 使用 CMYK 颜色配置文件的 PDF

- 包含 SVG 图像的 PDF

解决方案：

- 将 PDF 转换为 RGB 颜色配置文件
- 在置入 PDF 前先将 SVG 图像栅格化

词元估算

问题：PDF 产生预期外的词元消耗

参考标准：标准 8.5x11 英寸 PDF 页面，每页约按 2560 个词元估算

解决方案：根据文档长度相应调整 `maxTokens`

# API 和 SDK 参考

本节旨在提供适用于 Amazon Nova API 与 SDK 的完整技术参考文档。

主题

- [请求和响应架构](#)
- [SDK 参考](#)

## 请求和响应架构

Invoke API 和 Converse API 之间的请求架构几乎相同。主要区别在于二进制数据（图像、视频、音频）的编码方式：Converse API 使用二进制数组，而 Invoke API 使用 Base64 编码的字符串。

### 完整请求结构

以下是 Amazon Nova 模型的完整请求结构。除非标注为必填项，否则所有字段均为选填：

```
{
  "system": [
    {
      "text": "string"
    }
  ],
  "messages": [ // Required
    {
      "role": "user", // Required - first turn must be user
      "content": [ // Required
        {
          "text": "string"
        },
        {
          "image": {
            "format": "jpeg" | "png" | "gif" | "webp", // Required
            "source": { // Required
              "bytes": image // Binary array (Converse) or Base64 string (Invoke)
            }
          }
        }
      ],
    },
    {
      "video": {
```

```

    "format": "mkv" | "mov" | "mp4" | "webm" | "three_gp" | "flv" | "mpeg" |
"mpg" | "wmv",
    "source": {
      // Option 1: S3 location
      "s3Location": {
        "uri": "string", // e.g., s3://my-bucket/object-key
        "bucketOwner": "string" // Optional, e.g., "123456789012"
      },
      // Option 2: File bytes
      "bytes": video // Binary array (Converse) or Base64 string (Invoke)
    }
  },
  {
    "audio": { // Nova 2 Omni and Sonic only
      "format": "mp3" | "opus" | "wav" | "aac" | "flac" | "mp4" | "ogg" | "mkv",
      "source": {
        // Option 1: S3 location
        "s3Location": {
          "uri": "string",
          "bucketOwner": "string" // Optional
        },
        // Option 2: File bytes
        "bytes": audio // Binary array (Converse) or Base64 string (Invoke)
      }
    }
  }
],
{
  "role": "assistant",
  "content": [
    {
      "text": "string" // For prefilling assistant response
    }
  ]
}
],
"inferenceConfig": { // All optional
  "maxTokens": int, // 1-5000, default: dynamic
  "temperature": float, // 0.00001-1, default: 0.7
  "topP": float, // 0-1, default: 0.9
  "topK": int, // 0-128, default: not used
  "stopSequences": ["string"],

```

```
"reasoningConfig": { // Nova 2 Lite and Sonic only
  "type": "enabled" | "disabled", // default: "disabled"
  "maxReasoningEffort": "low" | "medium" | "high"
},
"toolConfig": { // Optional
  "tools": [
    {
      "toolSpec": {
        "name": "string", // Max 64 characters
        "description": "string",
        "inputSchema": {
          "json": {
            "type": "object",
            "properties": {
              "arg1": {
                "type": "string",
                "description": "string"
              }
            }
          },
          "required": ["string"]
        }
      }
    }
  ]
},
"toolChoice": { // Choose one option
  "auto": {},
  "any": {},
  "tool": {
    "name": "string"
  }
}
}
```

#### 关键请求参数：

- `system`：提供上下文信息和操作指令的系统提示词
- `messages`：包含角色（用户或助手）及对话内容的对话轮次数组
- `inferenceConfig`：控制模型输出行为（温度值、词元数量等）
- `toolConfig`：用于函数调用的工具规范

**Note**

使用 Converse API 时，`topK` 和 `reasoningConfig` 参数必须放置在 `additionalModelRequestFields` 中，不能放置在 `inferenceConfig` 中。

以下章节详细说明了每个请求参数：

## 系统

`system` - ( 可选 ) 请求的系统提示。系统提示词为 Amazon Nova 提供上下文信息和操作指令，例如指定特定的目标或角色。

## 消息

`messages` : ( 必需 ) ( 必填 ) 包含对话轮次的输入消息数组。

- `role` : ( 必需 ) 对话轮次的角色。有效值为 `user` 和 `assistant`。第一条消息必须始终使用 `user` 角色。
- `content` : ( 必填 ) 内容块数组。每个块指定一种内容类型 ( `text`、`image`、`video` 或 `audio` ) :
  - `text` : 对话轮次的文本内容。如果与图像或视频结合使用，则解释为随附文本。
  - `image` : ( 不适用于 Nova 2 Lite ) 图像内容，包含 :
    - `format` : ( 必填 ) 图像格式 : `jpeg`、`png`、`webp` 或 `gif`
    - `source.bytes` : ( 必填 ) 图像数据，以二进制数组形式 ( Converse API ) 或 Base64 字符串形式 ( Invoke API ) 传入
  - `video` : ( 不适用于 Nova 2 Lite ) 视频内容，包含 :
    - `format` : ( 必填 ) 视频格式 : `mkv`、`mov`、`mp4`、`webm`、`three_gp`、`flv`、`mpeg`、`mpg` 或 `wmv`
    - `source` : ( 必填 ) 视频源，通过 S3 URI ( `s3Location.uri` 和可选的 `bucketOwner` ) 或文件字节流 ( `bytes` ) 传入
  - `audio` : ( 仅适用于 Amazon Nova Sonic ) 音频内容，包含 :
    - `format` : ( 必填 ) 音频格式 : `mp3`、`opus`、`wav`、`aac`、`flac`、`mp4`、`ogg` 或 `mkv`
    - `source` : ( 必填 ) 音频源，通过 S3 URI 或文件字节流传入

## inferenceConfig

inferenceConfig : ( 选填 ) 控制模型输出生成的配置参数。

- maxTokens : ( 选填 ) 停止前要生成的最大词元数量。Amazon Nova 模型可能会在达到此限制之前停止。最大值为 5,000。未指定时, 将根据请求上下文使用动态默认值。
- temperature : ( 选填 ) 响应的随机性。有效范围 : 0.00001 – 1 ( 默认值 : 0.7 )。数值越低, 输出结果越具确定性。
- topP : ( 选填 ) 核采样阈值。Amazon Nova 从累积概率达到 topP 的词元中进行采样。有效范围 : 0 – 1 ( 默认值 : 0.9 )。建议仅调整 temperature 或 topP 二者之一, 切勿同时调整。
- topK : ( 选填 ) 仅从排名前 K 的词元中采样。此举可剔除低概率响应。有效范围 : 0 – 128 ( 默认值 : 未启用 )。

### Note

对于 Converse API, 请在 additionalModelRequestFields 中传递 topK。

- stopSequences : ( 选填 ) 字符串数组, 模型遇到指定内容时将停止生成。
- reasoningConfig : ( 仅限 Amazon Nova Sonic ) 推理配置 :
  - type : ( 选填 ) enabled 或 disabled ( 默认值 : disabled )
  - maxReasoningEffort : 计算强度 – low、medium 或 high。设置为 low 和 medium 时, 推理内容将逐段流式输出; 设置为 high 时, 推理结果将在最后一次性完整输出。

### Note

对于 Converse API, 请在 additionalModelRequestFields 中传递 reasoningConfig。

## toolConfig

toolConfig : ( 选填 ) 遵循 [ToolConfiguration 架构](#) 的工具配置。

- tools : 工具规范数组, 包含 name、description 和 inputSchema
- toolChoice : ( 选填 ) 控制工具选择策略 :
  - auto : 由模型决定是否使用工具及使用何种工具
  - any : 模型必须至少使用一个工具

- tool : 模型必须使用指定名称的工具

## 完整响应结构

以下为 Amazon Nova 模型的完整响应结构：

```
{
  "ResponseMetadata": {
    "RequestId": "string",
    "HTTPStatusCode": int,
    "HTTPHeaders": {
      "date": "string",
      "content-type": "application/json",
      "content-length": "string",
      "connection": "keep-alive",
      "x-amzn-requestid": "string"
    },
    "RetryAttempts": 0
  },
  "output": {
    "message": {
      "role": "assistant",
      "content": [
        {
          "reasoningContent": { // Optional - if reasoning enabled
            "reasoningText": {
              "text": "[REDACTED]"
            }
          }
        }
      ],
      "toolUse": { // Optional - if tool called
        "toolUseId": "string",
        "name": "string",
        "input": {} // Tool-specific arguments
      }
    },
    "text": "string" // Optional - text response
  },
  "image": { // Optional - Nova 2 Omni only
```

```

        "format": "png",
        "source": {
            "bytes": image // Binary array (Converse) or Base64 string (Invoke)
        }
    }
}
]
}
},
"stopReason": "string", // See stop reasons below
"usage": {
    "inputTokens": int,
    "outputTokens": int,
    "totalTokens": int
},
"metrics": {
    "latencyMs": int
}
}
}

```

停止原因：

- `end_turn`：响应自然结束
- `max_tokens`：已达到 `maxTokens` 限制
- `content_filtered`：违反内容使用政策
- `malformed_model_output`：模型输出格式无效
- `malformed_tool_use`：工具使用输出格式无效
- `service_unavailable`：内置工具服务不可访问
- `invalid_query`：对内置工具的查询无效
- `max_tool_invocations`：工具重试次数已用尽

以下章节详细说明了每个响应字段：

## output

`output`：(必填) 包含模型的响应消息。

- `message`：(必填) 助手的响应消息，包含角色与内容数组。
- `content`：(必填) 内容块数组，可能包含以下内容：

- `reasoningContent` : ( 选填 ) 启用推理功能时返回的内容。包含推理文本，该内容在响应中始终为 [REDACTED]。
- `toolUse` : ( 选填 ) 调用工具时返回的内容。包含工具使用 ID、名称和输入参数。
- `text` : ( 选填 ) 模型以文本内容响应时返回的内容。
- `image` : ( 选填 , 仅限 ) 模型生成图像时返回的内容。格式始终为 PNG。

## stopReason

`stopReason` : ( 必填 ) 指示模型停止生成输出的原因 :

- `end_turn` : 响应自然结束
- `max_tokens` : 已达到 `maxTokens` 限制或模型最大输出限制
- `content_filtered` : 输出内容违反 AWS 负责任的人工智能政策
- `malformed_model_output` : 模型生成无效输出
- `malformed_tool_use` : 模型生成无效的工具使用输出
- `service_unavailable` : 无法访问内置工具服务
- `invalid_query` : 对内置工具的查询无效
- `max_tool_invocations` : 内置工具重试后仍未生成有效结果

## usage

`usage` : ( 必填 ) 词元使用信息 :

- `inputTokens` : 模型处理的输入词元总数
- `outputTokens` : 生成的词元数量
- `totalTokens` : 输入词元数与输出词元数之和

## metrics

`metrics` : ( 必填 ) 性能指标 :

- `latencyMs` : 推理总耗时 ( 毫秒 )

## SDK 参考

AWS 软件开发工具包 ( SDK ) 适用于许多常用编程语言。每个 SDK 都提供 API、代码示例和文档，帮助开发人员更轻松地以其首选语言构建应用程序。

| SDK 文档                                         | 代码示例                                                |
|------------------------------------------------|-----------------------------------------------------|
| <a href="#">AWS SDK for C++</a>                | <a href="#">适用于 C++ 的 AWS SDK 代码示例</a>              |
| <a href="#">AWS CLI</a>                        | <a href="#">AWS CLI 代码示例</a>                        |
| <a href="#">适用于 Go 的 AWS SDK</a>               | <a href="#">适用于 Go 的 AWS SDK 代码示例</a>               |
| <a href="#">适用于 Java 的 AWS 开发工具包</a>           | <a href="#">适用于 Java 的 AWS SDK 代码示例</a>             |
| <a href="#">适用于 JavaScript 的 AWS SDK</a>       | <a href="#">适用于 JavaScript 的 AWS SDK 代码示例</a>       |
| <a href="#">AWS SDK for Kotlin</a>             | <a href="#">适用于 Kotlin 的 AWS SDK 代码示例</a>           |
| <a href="#">AWS SDK for .NET</a>               | <a href="#">适用于 .NET 的 AWS SDK 代码示例</a>             |
| <a href="#">适用于 PHP 的 AWS SDK</a>              | <a href="#">适用于 PHP 的 AWS SDK 代码示例</a>              |
| <a href="#">AWS Tools for PowerShell</a>       | <a href="#">适用于 PowerShell 的 AWS 工具代码示例</a>         |
| <a href="#">适用于 Python 的 AWS SDK ( Boto3 )</a> | <a href="#">适用于 Python ( Boto3 ) 的 AWS SDK 代码示例</a> |
| <a href="#">AWS SDK for Ruby</a>               | <a href="#">适用于 Ruby 的 AWS SDK 代码示例</a>             |
| <a href="#">适用于 Rust 的 AWS SDK</a>             | <a href="#">适用于 Rust 的 AWS SDK 代码示例</a>             |
| <a href="#">适用于 SAP ABAP 的 AWS SDK</a>         | <a href="#">适用于 SAP ABAP 的 AWS SDK 代码示例</a>         |
| <a href="#">适用于 Swift 的 AWS SDK</a>            | <a href="#">适用于 Swift 的 AWS SDK 代码示例</a>            |

# 在 SageMaker AI 上自定义 Amazon Nova 模型

您可以通过[配方](#)自定义 Amazon Nova 模型（包括增强版 Amazon Nova 2.0 模型），并在 SageMaker 上完成模型训练。这些配方支持监督式微调（SFT）和强化微调（RFT）之类的技术，同时提供全秩适配与低秩适配（LoRA）选项。

端到端自定义工作流程包括模型训练、模型评测和推理部署等阶段。SageMaker 上的这种模型自定义方法提供了更大的灵活性和控制力，可以微调其支持的 Amazon Nova 模型，精确地优化超参数，并实施 LoRA 参数高效微调（PEFT）、全秩 SFT、RFT 和持续预训练（CPT）等技术。

## 自定义方法

SageMaker 提供了两种用于自定义 Amazon Nova 模型的方法：

**基于 UI 的方法：**可通过简洁的引导式界面自定义 Amazon Nova 模型。该方法提供包含训练、评测与部署的端到端工作流，无需编写代码。基于 UI 的体验适合快速实验、概念验证开发以及偏好可视化工作流的用户。

**基于代码的方法：**使用 SageMaker Python SDK、Nova SDK 和训练配方，以编程方式自定义模型。该方法具备更高灵活性，支持配置高级超参数、集成 CI/CD 管道并实现训练工作流自动化。建议将基于代码的方法用于生产工作负载、复杂的自定义需求以及具备成熟 MLOps 实践的团队。

| 方法    | 适用于            | 主要优势           |
|-------|----------------|----------------|
| 基于 UI | 实验验证、原型开发、快速迭代 | 配置简便，流程引导，无需编码 |
| 基于代码  | 生产环境、自动化、高级配置  | 高度灵活，管道集成，版本控制 |

## 自定义平台

AWS 提供三个用于自定义 Amazon Nova 模型的平台，每个平台针对不同的使用案例和要求而设计：

**Amazon Bedrock：**提供最简单、最快速的模型自定义路径，设置最少。Bedrock 自动处理所有基础设施管理，让您专注于数据和使用案例。当您需要最快获得价值且偏好全托管体验时，该平台是理想选择。

**SageMaker 训练作业：**提供全托管环境，用于自定义 Amazon Nova 模型，无需创建或维护任何集群。该服务会自动处理所有基础设施预置、扩展和资源管理，使您能够专注于配置训练参数和提交作

业。该平台兼顾易用性与灵活性，支持参数高效微调 ( PEFT )、全秩微调和强化微调 ( RFT ) 等技术。

SageMaker HyperPod：专为大规模分布式训练提供专用环境，需创建并管理包含受限实例组 ( RIG ) 的 EKS 集群。该平台在配置训练环境方面提供最大的灵活性，支持专用 GPU 实例和集成的适用于 Lustre 的 Amazon FSx 存储，因此特别适合高级分布式训练场景、持续模型开发以及企业级自定义工作负载。

| 平台                 | 复杂度 | 弹性  | 适用于              |
|--------------------|-----|-----|------------------|
| Amazon Bedrock     | 最低  | 标准  | 最快自定义，最少设置       |
| SageMaker 训练作业     | 中   | 高   | 兼顾灵活性与易用性        |
| SageMaker HyperPod | 最高  | 最大值 | 大规模分布式训练，企业级工作负载 |

### Note

如果为 Amazon Nova 模型自定义训练作业提供 KMS 密钥，用于对 Amazon 拥有的输出 S3 存储桶进行加密：

- 在调用后续的迭代训练作业，或在调用 Amazon Bedrock 的 CreateCustomModel API 以使用该加密模型时，必须提供相同的 KMS 密钥。
- 调用 CreateTrainingJob API 的身份 ( 而非执行角色 ) 必须具备 KMS 密钥策略中定义的以下权限：CreateGrant、RetireGrant、Encrypt、GenerateDataKey。

## 一般先决条

自定义过程涉及几个关键阶段，其中包括模型训练、评估和推理部署等，每个阶段都需要特定的资源和配置。在 SageMaker 上开始自定义 Amazon Nova 模型之前，请确保满足以下一般先决条件。

- AWS 账户。如果您没有 AWS 账户，请按照[以下说明](#)注册一个。
- [安装 AWS CLI](#) 和 [设置配置](#)。
- 访问[基本 Amazon Nova 模型自定义配方](#)。

- 熟悉 [YAML](#) 配置文件。
- 熟悉如何在您的环境中运行笔记本。
- 熟悉如何创建 AWS 资源，例如 [Amazon S3 存储桶](#) 以及 [具有适当权限的 IAM 角色](#) 等。
- 熟悉如何 [使用 SageMaker 训练模型](#)。
- 熟悉 [SageMaker HyperPod 与 EKS 编排](#) 的协同工作机制。
- 熟悉 [SageMaker HyperPod CLI](#)。
- 熟悉 [Amazon Nova 基础模型](#)。
- 熟悉 [可用的 Amazon Nova 模型和自定义算法](#)。
- 熟悉 [Amazon Bedrock 推理](#)。

## Amazon Nova 配方

您可以从[配方](#)存储库中获取 Amazon Nova 配方。Amazon Nova 配方是指 [YAML](#) 配置文件，该文件向 SageMaker 提供有关如何运行模型自定义作业的详细信息。它提供基本模型名称、设置训练超参数、定义优化设置，并且包括成功微调或训练模型所需的任何其他选项。

您还可以通过 SageMaker AI Monarch 访问 Amazon Nova 配方：进入模型中心，选择 AWS，浏览 Amazon Nova 模型，即可找到其关联的配方。SageMaker AI Monarch 为每个配方都提供了示例笔记本，其中包括修改配方以及使用 SageMaker 训练作业或 SageMaker HyperPod 环境运行自定义作业的所有必要步骤。

要访问 Amazon SageMaker Studio 中的配方页面，执行角色必须具有以下权限。

### JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3::*model-customization-recipes*"
      ]
    }
  ]
}
```

```
    ]
}
```

要在 SageMaker 训练作业上执行示例笔记本，请使用以下 SageMaker 分发映像版本之一：2.7.1+、2.8.0+、3.2.1+、3.3.0+。这适用于 SageMaker AI Monarch。

### 主题

- [获取 Amazon Nova 配方](#)
- [获取 Amazon Nova Forge 配方](#)
- [可用的模型和算法](#)
- [Amazon Nova 2.0 Lite](#)

## 获取 Amazon Nova 配方

要获取基本 Amazon Nova 配方，请运行以下命令克隆[配方](#)存储库。

```
git clone https://github.com/aws/sagemaker-hyperpod-recipes.git
```

可通过 [recipes\\_collection/recipes/](#) 获得基本配方。

```
cd recipes_collection/recipes/
```

Amazon Nova 自定义配方位于以下文件夹中。

| 食谱类型             | Folder                           |
|------------------|----------------------------------|
| SFT ( 全秩和 PEFT ) | <a href="#">fine-tuning/nova</a> |
| 评估               | <a href="#">evaluation/nova</a>  |
| CPT              | <a href="#">training/nova</a>    |

## 获取 Amazon Nova Forge 配方

如需访问作业专用的 Amazon Nova Forge 配方，请先完成 [Amazon Nova Forge 访问权限设置](#)，然后按照[本讲习会](#)完成 Forge 版 CLI 的设置。对于 SageMaker 训练作业，AWS 管理控制台和 Nova Forge SDK 将自动访问 Forge 配方。

## 可用的模型和算法

下表汇总了 Amazon Nova 模型自定义功能的可用情况，以及使用 SageMaker 的支持算法。

| 模型名称                 | 模型 ID                        | 微调 | 备注                                                                                                                                                                              |
|----------------------|------------------------------|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Amazon Nova Lite 2.0 | amazon.nova-2-lite-v1:0:256k | 是  | <ul style="list-style-type: none"><li>• SFT：接受文本和/或图像或文本和/或视频作为输入，并生成文本作为输出。单个作业不能在同一次运行中合并文本、图像和视频。</li><li>• RFT：支持以文本和图像作为单轮对话的输入，并输出文本。具备更优的奖励优化能力。</li><li>• CPT</li></ul> |

## Amazon Nova 2.0 Lite

下表列出了 Amazon Nova 2.0 Lite 配方参考的详细信息。

| 模型            | 类别/子类别 | 方法    | 配方名称                                  | 映像 URI ( 训练作业 )                                                                      | 映像 URI ( SageMaker HyperPod )                                                        | 计算实例                              |
|---------------|--------|-------|---------------------------------------|--------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|-----------------------------------|
| Nova 2.0 Lite | 训练     | 持续预训练 | nova_lite_2_0_p5x8_gpu_pret_rain.yaml | 不适用                                                                                  | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-HP-CPT-V2-latest | m1.p5.48xlarge , m1.p5en.48xlarge |
| Nova 2.0 Lite | 参数高效微调 | 微调    | nova_lite_2_0_p5_gpu_lora_sft.yaml    | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-TJ-SFT-V2-latest | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-HP-SFT-V2-latest | m1.p5.48xlarge , m1.p5en.48xlarge |

| 模型            | 类别/子类别   | 方法 | 配方名称                                        | 映像 URI ( 训练作业 )                                                                      | 映像 URI ( SageMaker HyperPod )                                                              | 计算实例                             |
|---------------|----------|----|---------------------------------------------|--------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|----------------------------------|
| Nova 2.0 Lite | 全秩微调     | 微调 | nova_lite_v2_0_p5_gpu_sft.yaml              | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-TJ-SFT-V2-latest | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-HP-SFT-V2-latest       | m1.p5.48xlarge, m1.p5en.48xlarge |
| Nova 2.0 Lite | 参数高效 RFT | 微调 | nova_lite_v2_smtj_p5_p5en_gpu_lora_rft.yaml | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-TJ-RFT-V2-latest | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-HP-RFT-TRAIN-V2-latest | m1.p5.48xlarge, m1.p5en.48xlarge |

| 模型            | 类别/子类别 | 方法 | 配方名称                                | 映像 URI ( 训练作业 )                                                                      | 映像 URI ( SageMaker HyperPod )                                                              | 计算实例                             |
|---------------|--------|----|-------------------------------------|--------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|----------------------------------|
| Nova 2.0 Lite | 全秩 RFT | 微调 | nova_lite_v2_smtj_p5en_gpu_rft.yaml | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-TJ-RFT-V2-latest | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-HP-RFT-TRAIN-V2-latest | m1.p5.48xlarge, m1.p5en.48xlarge |

| 模型            | 类别/子类别 | 方法 | 配方名称                                                        | 映像 URI ( 训练作业 )                                                                        | 映像 URI ( SageMaker HyperPod )                                                          | 计算实例                              |
|---------------|--------|----|-------------------------------------------------------------|----------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|-----------------------------------|
| Nova 2.0 Lite | 自带数据   | 评估 | nova_lite_2_0_p5_48x1_gpu_bring_your_own_data_set_eval.yaml | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-evaluation-repo:SM-TJ-Eval-V2-1atest | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-evaluation-repo:SM-HP-Eval-V2-1atest | m1.p5.48xlarge , m1.p5en.48xlarge |
| Nova 2.0 Lite | 通用文本基准 | 评估 | nova_lite_2_0_p5_48x1_gpu_general_text_benchmark_eval.yaml  | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-evaluation-repo:SM-TJ-Eval-V2-1atest | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-evaluation-repo:SM-HP-Eval-V2-1atest | m1.p5.48xlarge , m1.p5en.48xlarge |

| 模型            | 类别/子类别           | 方法 | 配方名称                                                  | 映像 URI ( 训练作业 )                                                                        | 映像 URI ( SageMaker HyperPod )                                                          | 计算实例                              |
|---------------|------------------|----|-------------------------------------------------------|----------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|-----------------------------------|
| Nova 2.0 Lite | RFT 评测           | 评估 | nova_lite_2_0_p5_4_8x1_gpu_rft_eval.yaml              | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-evaluation-repo:SM-TJ-Eval-V2-1atest | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-evaluation-repo:SM-HP-Eval-V2-1atest | m1.p5.48xlarge , m1.p5en.48xlarge |
| Nova 2.0 Lite | LLM Rubric Judge | 评估 | nova_lite_2_0_p5_4_8x1_gpu_rubric_llm_judge_eval.yaml | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-evaluation-repo:SM-TJ-Eval-V2-1atest | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-evaluation-repo:SM-HP-Eval-V2-1atest | m1.p5.48xlarge , m1.p5en.48xlarge |

# 基于 SageMaker 训练作业自定义 Amazon Nova

SageMaker 训练作业是支持您大规模训练机器学习模型的环境。它可以自动预调配和扩展计算资源，从 Amazon S3 等来源加载训练数据，执行您的训练代码，并存储生成的模型构件。

训练目标是使用您的专有数据自定义基本 Amazon Nova 模型。训练过程通常涉及准备数据、选择[配方](#)、修改 YAML 文件中的配置参数以及提交训练作业等步骤。训练过程将在服务托管的 Amazon S3 存储桶中输出经过训练的模型检查点。您可以将此检查点位置用于评估作业。基于 SageMaker AI 训练作业的 Nova 自定义，将模型构件存储在服务托管的 Amazon S3 存储桶中。服务托管存储桶中的构件使用 SageMaker AI 托管的 KMS 密钥进行加密。服务托管的 Amazon S3 存储桶目前不支持使用客户托管的 KMS 密钥进行数据加密。

## Amazon Nova 自定义最佳实践

### 概述

本节概述了自定义技术，有助您根据自身需求与可用数据选择最优方案。

### LLM 训练的两个阶段

大型语言模型训练包括两个主要阶段：预训练和后训练。在预训练阶段，模型会处理原始文本的词元，并针对下一词元预测进行优化。此过程会产生一个模式补全器，可从网络和精选文本中学习语法、语义、事实知识和推理模式。然而，预训练模型无法理解指令、用户目标，也不具备适配上下文的行为能力。其会依据训练数据分布所涵盖的任意风格，持续生成文本。预训练模型执行的是自动补全而非遵循指令，输出格式不统一，还可能复刻训练数据中的不良偏见与不安全内容。预训练构建的是通用能力，而非任务完成能力。

后训练则可将模式补全器转化为实用助手。首先进行多轮监督式微调 ( SFT )，通过模仿高质量的演示，教会模型遵循指令、遵守架构和策略、调用工具并生成可靠的输出。这种齐训练能让模型将提示词视为待执行的任务，而非单纯的续写文本。接着引入强化微调 ( RFT )，利用可衡量的反馈 ( 如验证器或 LLM-as-a-judge ) 来优化模型行为，在准确性与简洁性、安全性与覆盖范围，或约束条件下的多步骤推理等权衡中实现平衡。在实践中，通常采用 SFT 与 RFT 交替迭代的策略，将预训练模型打磨为一个稳定可靠、策略对齐且能一致性处理复杂任务的智能系统。

### 选择合适的自定义方法

本节将介绍后训练自定义策略：RFT 和 SFT。

## 强化微调 ( RFT )

强化微调通过反馈信号 ( 即可衡量回复质量的分数或奖励 ) 提升模型性能，而非依靠精确标准答案进行直接监督。与从输入-输出对中学习的传统监督式微调不同，RFT 使用奖励函数评测模型响应，并迭代优化模型以最大化奖励。这种方法适用于难以定义精确输出但能可靠衡量响应质量的任务。RFT 使模型能够通过尝试和反馈来学习复杂行为与偏好，非常适合需要精细决策、创造性问题求解，或需符合可程序化评测的特定质量标准的场景。例如，回答复杂法律问题就是 RFT 的理想使用案例，因为您希望教会模型如何更好地进行推理，从而更准确地回答问题。

### 工作原理

在强化微调中，您从经过指令微调的基线模型出发，将每个提示视作一场小型竞赛。对于给定输入，从模型中采样若干候选回答，通过奖励函数为每个回答打分并在本组内排序。更新步骤会促使模型提升高分候选回答的生成概率，降低低分回答的概率；同时通过“贴近基线”约束，避免模型行为偏移、内容冗长或钻规则漏洞。您在大量提示上重复这一循环，不断补充困难样本，在发现漏洞利用行为时强化验证器或优化评判规则，并持续跟踪任务指标。

### 何时使用 RFT

最适合使用 RFT 的任务通常具备以下共同特征。即便难以定义唯一正确输出，仍存在可衡量的成功信号；支持部分得分或分级质量评估，因此可对同一提示下的回答进行优劣排序，或通过奖励函数完成排序；涉及多项需兼顾的目标 ( 例如在准确性与简洁性、清晰度、安全性或成本之间取得平衡 ) ；需要遵守可通过程序校验的明确约束；在工具介导或基于环境的环境中运行，且结果可观察 ( 成功或失败、延迟、资源使用情况 ) ；处于低标注数据场景，即获取标准答案成本高昂，但自动化或基于评分规则的反馈十分丰富。当您能将质量转化为可靠的标量分值或排序，并希望模型优先强化高分行为，而无需大量标注样本时，RFT 效果最佳。

在以下情况下，建议考虑其他方法：

- 拥有充足且可靠的标注输入-输出对：使用 SFT
- 主要差距在于知识或术语：使用检索增强生成 ( RAG )
- 奖励信号存在噪声或不可靠，且无法通过优化评分规则或校验工具解决：在采用 RFT 之前先稳定奖励信号

### 何时不使用 RFT

在以下情况下应避免使用 RFT：

- 可低成本生成可靠的标注输入-输出对 ( SFT 更简单、低成本且更稳定 )

- 差距在于知识或术语，而非行为（建议使用 RAG）
- 奖励信号存在噪声、稀疏、易被钻漏洞，或者计算成本高昂、速度缓慢（先优化评测器）
- 基线性能近乎为零（先通过 SFT 完成引导，再进行偏好优化）
- 任务具备确定性架构、严格格式要求或唯一正确答案（SFT 或基于规则的验证效果更佳）
- 严苛的延迟或成本预算，无法承担 RFT 所需的额外采样与探索开销
- 安全与策略约束定义不清晰，且无法在奖励函数中有效强制执行

如果能明确给出“正确答案”，请使用 SFT。如果需要补充新知识，请使用 RAG。仅当拥有稳健的基线模型，以及可靠、高效、难以被利用的奖励函数后，再使用 RFT。

## 监督式微调 (SFT)

监督式微调是在针对任务构建的人工标注输入-输出数据集上训练 LLM。您可以提供提示（问题、指令等）示例以及正确或所需响应，并基于这些样本继续训练模型。调整模型权重以最大限度地减少监督损失（通常是其预测与目标输出词元之间的交叉熵）。这与大多数监督式机器学习任务中使用的训练方式相同，旨在对 LLM 进行专业化处理。

SFT 改变的是行为，而非知识。其不会让模型学到预训练阶段未见过的新事实或专业术语，只教会模型如何回答，而非掌握什么知识。如果需要新增领域知识（如内部术语），可在推理阶段通过检索增强生成（RAG）提供相关上下文，由 SFT 赋予模型所需的指令遵循能力。

## 工作原理

SFT 通过最大限度地减少响应词元上的平均交叉熵损失来优化 LLM，将提示词元视为上下文并将其从损失计算中屏蔽。模型会将目标风格、结构和决策规则内化，学习为每个提示生成正确的补全。例如，要将文档分类为自定义类别，可以使用提示（文档文本）和带标签的补全（类别标签）来微调模型。您可以在这些数据对上训练模型，直到模型能够以高概率为每个提示输出正确的标签。

您可以使用少至几百个样本进行 SFT，也可以纵向扩展到几十万个样本。SFT 样本必须具备高质量，且与期望的模型行为直接对齐。

## 何时使用 SFT

当任务定义清晰、预期输出明确时，即可使用 SFT。如果您能明确给出“输入为 X 时，正确输出为 Y”的形式，并能收集此类映射样本，监督式微调便是理想选择。SFT 在以下场景中表现出色：

- 结构化或复杂的分类任务：将内部文档或合同分为多个自定义类别。通过 SFT，模型对这些特定类别的学习效果要远优于仅使用提示词。

- 具有已知答案的问答或转换任务：微调模型以回答公司知识库中的问题，或者在格式之间转换数据，其中每个输入都有对应的正确答案。
- 格式和风格一致性：通过根据正确格式或语气的样本进行微调，训练模型始终以特定格式或语气做出响应。例如，在展示特定品牌语气的提示-响应对上进行训练，可以使模型学会以该风格生成输出。同样，指令遵循行为通常也是通过在精选的良好助手行为样本上进行 SFT 来初步习得。

当您可以指定正确行为是什么样子时，SFT 是教授 LLM 新技能或行为最直接的方式。其利用模型现有的语言理解能力，并将其聚焦于任务上。若希望模型执行特定任务，且已有或可构建样本数据集，即可使用 SFT。

当您能够整理出高质量、高度贴合预期行为的提示-响应对时，也适合使用 SFT。该方法适用于目标明确或格式固定的任务，例如架构、函数或工具调用，以及适合以模仿作为训练信号的结构化回答等场景。其目标是行为塑造：让模型将提示视为任务、遵循指令、采用指定语气与拒答策略，并保持输出格式一致。建议至少准备数百个示范样本，数据质量、一致性与去重比样本数量更为重要。如需实现简单且高性价比的更新，可采用低秩适应 (LoRA) 等参数高效微调方法训练小型适配器，同时保持基座主体参数不变。

## 何时不使用 SFT

当差距在于知识而非行为时，不建议使用 SFT。其无法让模型学习新事实、专业术语或最新事件。此类场景应采用检索增强生成，在推理阶段引入外部知识。当您可以评估质量却无法标注唯一正确答案时，也应避免 SFT。请改用带可验证奖励或 LLM-as-a-judge 的强化微调，直接对奖励进行优化。如果需求或内容频繁变动，应依赖检索与工具使用，而非重新训练模型。

## 主题

- [Nova Forge SDK](#)
- [Amazon Nova 模型训练](#)
- [监控迭代进度](#)
- [评估经过 SageMaker AI 训练的模型](#)
- [迭代训练](#)

## Nova Forge SDK

Nova Forge SDK 是一套功能完整的 Python SDK，专用于 Amazon Nova 模型的自定义开发。该 SDK 为跨不同平台（包括 SageMaker AI 和 Amazon Bedrock）的 Amazon Nova 模型，提供了训练、评

估、监控、部署及推理的统一接口。无论是针对特定领域任务适配模型，还是根据自身使用案例优化性能，该 SDK 都能在单一统一接口中提供您所需的全部功能。

## 优势

- 一款 SDK 覆盖全流程模型自定义生命周期：从数据准备到部署与监控。
- 支持多种训练方法，包括持续预训练 ( CPT )、监督式微调 ( SFT )、直接偏好优化 ( DPO )、强化微调 ( RFT )；同时支持单轮/多轮对话场景，以及 LoRA 与全秩适配方式。
- 为 SageMaker 训练作业、SageMaker HyperPod 和 Amazon Bedrock 提供内置支持，且可实现自动资源管理。
- 无需再为训练方法查找适配的配方或容器 URI。
- 支持自备训练配方，也可使用 SDK 智能默认值并覆盖参数。
- 该 SDK 会依据支持的模型与实例组合对配置进行校验，并提供验证支持，从而在训练启动前规避错误。
- 集成 Amazon CloudWatch 监控功能，支持实时追踪训练进度。
- 集成 MLflow，可通过 SageMaker AI MLflow 跟踪服务器追踪训练实验。

## 要求

该 SDK 要求的最低 Python 版本为 3.12。

## 安装

如需安装该 SDK，请执行以下命令。

```
pip install amzn-nova-forge
```

## 支持的模型与技术

该 SDK 支持 Amazon Nova 系列中的以下模型与技术：

| 方法         | 支持的模型                                  |
|------------|----------------------------------------|
| 持续预训练      | <a href="#">所有 Nova 模型</a> ( 仅限 SMHP ) |
| 监督式微调 LoRA | <a href="#">所有 Nova 模型</a>             |

| 方法          | 支持的模型                                         |
|-------------|-----------------------------------------------|
| 全秩监督式微调     | <a href="#">所有 Nova 模型</a> ( 仅限 SMHP 和 SMTJ ) |
| 直接偏好优化 LoRA | Nova 1.0 模型 ( 仅限 SMHP 和 SMTJ )                |
| 全秩直接偏好优化    | Nova 1.0 模型 ( 仅限 SMHP 和 SMTJ )                |
| 强化微调 LoRA   | Nova Lite 2.0                                 |
| 全秩强化微调      | Nova Lite 2.0 ( 仅限 SMHP 和 SMTJ )              |
| 多轮强化微调 LoRA | Nova Lite 2.0 ( 仅限 SMHP )                     |
| 全秩多轮强化微调    | Nova Lite 2.0 ( 仅限 SMHP )                     |

## 开始使用

### 主题

- [1. 准备数据](#)
- [2. 配置基础设施](#)
- [3. 训练](#)
- [4. 监控](#)
- [5. 评估](#)
- [6. 部署](#)

### 1. 准备数据

从本地文件或 S3 加载数据集后，SDK 会自动将其转换为所选训练方法对应的标准格式。您也可直接提供已格式化的数据，快速启动训练流程。

```
from amzn_nova_forge.dataset.dataset_loader import JSONLDatasetLoader
from amzn_nova_forge.model.model_enums import Model, TrainingMethod

loader = JSONLDatasetLoader(question="input", answer="output")
loader.load("s3://your-bucket/training-data.jsonl")
loader.transform(method=TrainingMethod.SFT_LORA, model=Model.NOVA_LITE)
```

## 2. 配置基础设施

选定计算资源，SDK 自动验证配置，保障最优部署。

```
from amzn_nova_forge.manager.runtime_manager import BedrockRuntimeManager,
    SMTJRuntimeManager, SMHPRuntimeManager
# Bedrock
runtime = BedrockRuntimeManager(
    execution_role="arn:aws:iam::123456789012:role/ExampleRole"
)

# SageMaker Training Jobs
runtime = SMTJRuntimeManager(
    instance_type="ml.p5.48xlarge",
    instance_count=4
)

# SageMaker HyperPod
runtime = SMHPRuntimeManager(
    instance_type="ml.p5.48xlarge",
    instance_count=4,
    cluster_name="my-hyperpod-cluster",
    namespace="kubeflow"
)
```

## 3. 训练

仅需几行代码即可启动训练。

```
from amzn_nova_forge.model import NovaModelCustomizer
from amzn_nova_forge.model.model_enums import Model, TrainingMethod

customizer = NovaModelCustomizer(
    model=Model.NOVA_LITE_2,
    method=TrainingMethod.SFT_LORA,
    infra=runtime,
    data_s3_path="s3://your-bucket/prepared-data.jsonl"
)

result = customizer.train(job_name="my-training-job")
```

## 4. 监控

您可直接通过 SDK 追踪训练进度。

```
from amzn_nova_forge.monitor.log_monitor import CloudWatchLogMonitor

# Monitor training logs
customizer.get_logs()

# Or monitor directly via CloudWatchLogMonitor
monitor = CloudWatchLogMonitor.from_job_result(result)
monitor.show_logs(limit=10)

# Check job status
result.get_job_status() # InProgress, Completed, Failed
```

## 5. 评估

通过多种[内置基准](#)评估模型性能，也可自定义评估方案。

```
from amzn_nova_forge.recipe_config.eval_config import EvaluationTask

# Evaluate on benchmark tasks
eval_result = customizer.evaluate(
    job_name="model-eval",
    eval_task=EvaluationTask.MMLU,
    model_path=result.model_artifacts.checkpoint_s3_path
)
```

## 6. 部署

借助内置的 Amazon Bedrock 或 SageMaker 适配能力，可将自定义模型部署到生产环境。

```
from amzn_nova_forge.model.model_enums import DeployPlatform

# Bedrock provisioned throughput
deployment = customizer.deploy(
    model_artifact_path=result.model_artifacts.checkpoint_s3_path,
    deploy_platform=DeployPlatform.BEDROCK_PT,
    pt_units=10
)
```

```
# Bedrock On-Demand
deployment = customizer.deploy(
    model_artifact_path=result.model_artifacts.checkpoint_s3_path,
    deploy_platform=DeployPlatform.BEDROCK_OD,
    pt_units=10
)

# Sagemaker Real-time Inference
deployment = customizer.deploy(
    model_artifact_path=result.model_artifacts.checkpoint_s3_path,
    deploy_platform=DeployPlatform.SAGEMAKER,
    unit_count=10,
    sagemaker_instance_type="ml.p5.48xlarge",
    sagemaker_environment_variables={
        "CONTEXT_LENGTH": "12000",
        "MAX_CONCURRENCY": "16",
    }
)
```

## 关键功能

### 动态配方生成

该 SDK 无需您为特定训练方法查找适配的配方或容器 URI。

### 智能数据处理

该 SDK 会自动将数据转换为训练所需的标准格式。无论您使用 JSON、JSONL 或 CSV 文件，数据加载器均可无缝完成格式转换。数据加载器同时支持文本数据及多模态数据（图像和视频）。

### 企业级基础设施支持

该 SDK 可兼容 SageMaker 训练作业与 SageMaker HyperPod，并自动管理以下内容：

- 实例类型验证
- 配方验证
- 数据集验证
- 作业编排与监控

该 SDK 还支持 SageMaker 训练作业无服务器模式和 Bedrock 自定义功能。

## 全面评测

基于行业**标准基准**评测自定义模型，包括：

- MMLU ( 大规模多任务语言理解 )
- BBH ( 高级推理任务 )
- GPQA ( Google 无法解答的研究生级别问答基准 )

您可直接使用基准默认配置，或根据需求自行定义：

- BYOM ( 自定义指标 )
- BYOD ( 自定义数据集 )

## 生产环境部署

将模型部署到 Amazon Bedrock 或 SageMaker AI，支持以下部署方式：

- Bedrock 预置吞吐量：提供专属算力，保障性能稳定
- Bedrock 按需调用 ( 仅适用于基于 LoRA 的自定义 )：按需付费
- SageMaker AI 实时推理：提供专属算力，保障性能稳定

## 批量推理

高效运行大规模推理作业：

- 并行处理数千个请求
- 结果自动聚合
- 高性价比批量处理

## Nova Forge

对于 Nova Forge 订阅用户，该 SDK 支持数据混合配方。

## 了解更多

准备好开始使用 Nova Forge SDK 来自定义 Nova 模型了吗？前往我们的 GitHub 存储库，查看详细指南、API 参考及更多示例：：<https://github.com/aws/nova-forge-sdk>

# Amazon Nova 模型训练

通过 SageMaker 训练作业对 Amazon Nova 模型进行训练，支持监督式微调 ( SFT ) 与强化微调 ( RFT )。每种技术适用于不同的自定义需求，并可应用于不同版本的 Amazon Nova 模型。

## 主题

- [微调 Nova 2.0](#)
- [使用 Amazon Nova 模型进行强化微调 \( RFT \)](#)

## 微调 Nova 2.0

### 先决条件

在开始训练作业之前，请注意具备以下内容：

- Amazon S3 存储桶，用于存储您的输入数据和训练作业的输出。您可以为这两者使用一个存储桶，也可以为每种类型的数据使用不同的存储桶。确保您的存储桶位于您创建所有其他训练资源所用的 AWS 区域。有关更多信息，请参阅[创建通用存储桶](#)。
- 具有运行训练作业权限的 IAM 角色。请务必为 IAM 策略附加 AmazonSageMakerFullAccess。有关更多信息，请参阅[如何使用 SageMaker AI 执行角色](#)。
- 基本 Amazon Nova 配方，请参阅[???](#)。

### 什么是 SFT？

监督式微调 ( SFT ) 使用带标注的输入-输出对训练语言模型。模型从包含提示和响应的示范示例中学习，优化能力以适配特定任务、指令或预期行为。

### 数据准备

#### 概述

Nova 2.0 SFT 数据采用与 Nova 1.0 相同的 Converse API 格式，并新增了可选的推理内容字段。如需完整格式规范，请参阅：

- 推理内容：[ReasoningContentBlock](#)
- Converse API 架构：[Converse API](#)
- 数据集约束：[数据集约束](#)

## 支持的功能

- 输入类型：用户内容块中的文本、图像或视频
- 助手内容：纯文本响应和推理内容
- 数据集构成：必须为同构数据。选择如下选项之一：
  - 纯文本轮次
  - 文本 + 图像轮次
  - 文本 + 视频轮次（支持文档理解）

### Important

不得在同一数据集中或不同对话轮次间混合使用图像和视频。

## 目前的局限性

- 多模态推理内容：虽然 Converse 格式支持基于图像的推理内容，但 Nova 2.0 SFT 仅支持 reasoningText 字段中基于文本的推理内容。
- 验证集：Nova 2.0 的 SFT 训练不支持提供验证数据集。若传入验证数据集，训练过程中将被忽略。该限制同时适用于基于 UI 提交和程序化提交的训练作业。

## 支持的媒体格式

- 图像：PNG、JPEG、GIF
- 视频：MOV、MKV、MP4

## 数据格式示例

### Text-only (Nova 1.0 compatible)

```
{
  "schemaVersion": "bedrock-conversation-2024",
  "system": [
    {
      "text": "You are a digital assistant with a friendly personality"
    }
  ],
}
```

```
"messages": [
  {
    "role": "user",
    "content": [
      {
        "text": "What country is right next to Australia?"
      }
    ]
  },
  {
    "role": "assistant",
    "content": [
      {
        "text": "The closest country is New Zealand"
      }
    ]
  }
]
```

### Text with reasoning (Nova 2.0)

```
{
  "schemaVersion": "bedrock-conversation-2024",
  "system": [
    {
      "text": "You are a digital assistant with a friendly personality"
    }
  ],
  "messages": [
    {
      "role": "user",
      "content": [
        {
          "text": "What country is right next to Australia?"
        }
      ]
    },
    {
      "role": "assistant",
      "content": [
        {
          "reasoningContent": {
```

```

        "reasoningText": {
            "text": "I need to use my world knowledge of geography to answer this
question"
        }
    },
    {
        "text": "The closest country to Australia is New Zealand, located to the
southeast across the Tasman Sea."
    }
]
}

```

## Image + text input

```

{
  "schemaVersion": "bedrock-conversation-2024",
  "system": [
    {
      "text": "You are a helpful assistant."
    }
  ],
  "messages": [
    {
      "role": "user",
      "content": [
        {
          "image": {
            "format": "jpeg",
            "source": {
              "s3Location": {
                "uri": "s3://your-bucket/your-path/your-image.jpg",
                "bucketOwner": "your-aws-account-id"
              }
            }
          }
        },
        {
          "text": "Which country is highlighted in the image?"
        }
      ]
    }
  ]
}

```

```
},
{
  "role": "assistant",
  "content": [
    {
      "reasoningContent": {
        "reasoningText": {
          "text": "I will determine the highlighted country by examining its
location on the map and using my geographical knowledge"
        }
      }
    },
    {
      "text": "The highlighted country is New Zealand"
    }
  ]
}
]
```

## Video + text input

```
{
  "schemaVersion": "bedrock-conversation-2024",
  "system": [
    {
      "text": "You are a helpful assistant."
    }
  ],
  "messages": [
    {
      "role": "user",
      "content": [
        {
          "video": {
            "format": "mp4",
            "source": {
              "s3Location": {
                "uri": "s3://your-bucket/your-path/your-video.mp4",
                "bucketOwner": "your-aws-account-id"
              }
            }
          }
        }
      ]
    }
  ]
}
```

```
    },
    {
      "text": "What is shown in this video?"
    }
  ],
},
{
  "role": "assistant",
  "content": [
    {
      "reasoningContent": {
        "reasoningText": {
          "text": "I will analyze the video content to identify key elements"
        }
      }
    },
    {
      "text": "The video shows a map with New Zealand highlighted"
    }
  ]
}
]
```

## 工具调用

Nova 2.0 SFT 支持基于工具调用模式进行模型训练，使模型能够学习何时以及如何调用外部工具或函数。

### 工具调用的数据格式

工具调用训练数据包含用于定义可用工具的 `toolConfig` 部分，以及展示工具使用模式的对话轮次内容。

### 示例输入

```
{
  "schemaVersion": "bedrock-conversation-2024",
  "system": [
    {
      "text": "You are an expert in composing function calls."
    }
  ],
}
```

```
"toolConfig": {
  "tools": [
    {
      "toolSpec": {
        "name": "getItemCost",
        "description": "Retrieve the cost of an item from the catalog",
        "inputSchema": {
          "json": {
            "type": "object",
            "properties": {
              "item_name": {
                "type": "string",
                "description": "The name of the item to retrieve cost for"
              },
              "item_id": {
                "type": "string",
                "description": "The ASIN of item to retrieve cost for"
              }
            },
            "required": [
              "item_id"
            ]
          }
        }
      }
    },
    {
      "toolSpec": {
        "name": "getItemAvailability",
        "description": "Retrieve whether an item is available in a given location",
        "inputSchema": {
          "json": {
            "type": "object",
            "properties": {
              "zipcode": {
                "type": "string",
                "description": "The zipcode of the location to check in"
              },
              "quantity": {
                "type": "integer",
                "description": "The number of items to check availability for"
              },
              "item_id": {
                "type": "string",
```

```
        "description": "The ASIN of item to check availability for"
      }
    },
    "required": [
      "item_id", "zipcode"
    ]
  }
}
},
]
},
"messages": [
  {
    "role": "user",
    "content": [
      {
        "text": "I need to check whether there are twenty pieces of the following
item available. Here is the item ASIN on Amazon: id-123. Please check for the zipcode
94086"
      }
    ]
  },
  {
    "role": "assistant",
    "content": [
      {
        "reasoningContent": {
          "reasoningText": {
            "text": "The user wants to check how many pieces of the item with ASIN
id-123 are available in the zipcode 94086"
          }
        }
      }
    ],
    {
      "toolUse": {
        "toolUseId": "getItemAvailability_0",
        "name": "getItemAvailability",
        "input": {
          "zipcode": "94086",
          "quantity": 20,
          "item_id": "id-123"
        }
      }
    }
  }
]
```

```

    }
  ]
},
{
  "role": "user",
  "content": [
    {
      "toolResult": {
        "toolUseId": "getItemAvailability_0",
        "content": [
          {
            "text": "[{\"name\": \"getItemAvailability\", \"results\":
{\"availability\": true}]]\"
          }
        ]
      }
    }
  ]
},
{
  "role": "assistant",
  "content": [
    {
      "text": "Yes, there are twenty pieces of item id-123 available at 94086.
Would you like to place an order or know the total cost?"
    }
  ]
}
]
}

```

## 工具调用要求

创建工具调用训练数据时，请遵循以下要求：

| 要求            | 说明                                    |
|---------------|---------------------------------------|
| ToolUse 放置    | ToolUse 只能出现在助手轮次中                    |
| ToolResult 放置 | ToolResult 只能出现在用户轮次中                 |
| ToolResult 格式 | ToolResult 只能为文本或 JSON。Nova 模型不支持其他模态 |

| 要求             | 说明                                                                         |
|----------------|----------------------------------------------------------------------------|
| inputSchema 格式 | toolSpec 中的 inputSchema 必须是有效的 JSON 架构对象                                   |
| toolUseId 匹配   | 每个 ToolResult 必须引用前序助手轮次 ToolUse 中的有效 toolUseId，且每个 toolUseId 在单次对话中仅可使用一次 |

## 重要提示

- 确保工具定义在所有训练样本中保持一致
- 模型将从所提供的示例中学习工具调用模式
- 包含各类场景示例，明确各工具的适用与不适用情况

## 文档理解

Nova 2.0 SFT 支持基于文档任务的训练，使模型能够学习如何分析并回答与 PDF 文档相关的问题。

### 文档理解数据格式

文档理解训练数据在用户内容块中包含文档引用，模型将学习对文档内容进行提取与推理。

### 示例输入

```
{
  {
    "schemaVersion": "bedrock-conversation-2024",
    "messages": [
      {
        "role": "user",
        "content": [
          {
            "text": "What are the ways in which a customer can experience issues during checkout on Amazon?"
          },
          {
            "document": {
              "format": "pdf",
              "source": {
                "s3Location": {
                  "uri": "s3://my-bucket-name/path/to/documents/customer_service_debugging.pdf",

```

```

        "bucketOwner": "123456789012"
      }
    }
  }
]
},
{
  "role": "assistant",
  "content": [
    {
      "reasoningContent": {
        "reasoningText": {
          "text": "I need to find the relevant section in the document to answer
the question."
        }
      },
      {
        "text": "Customers can experience issues with 1. Data entry, 2. Payment
methods, 3. Connectivity while placing the order. Which one would you like to dive
into?"
      }
    ]
  }
]
}
}
}

```

## 文档理解限制

| 限制     | Details                           |
|--------|-----------------------------------|
| 支持的格式  | 仅支持 PDF 文件                        |
| 最大文档大小 | 10 MB                             |
| 模态混合   | 单个样本可包含文档与文本，但不可将文档与其他模态（图像、视频）混用 |

## 文档理解最佳实践

- 确保文档格式清晰，文本可正常提取
- 提供覆盖不同文档类型与问题格式的多样化样本
- 包含推理内容，帮助模型学习文档分析模式

## 视频理解

Nova 2.0 SFT 支持基于视频任务的训练，使模型能够学习如何分析并回答与视频内容相关的问题。

## 视频理解数据格式

视频理解训练数据在用户内容块中包含视频引用，模型将学习从视频内容中提取信息并进行推理。

## 示例输入

```
{
  "schemaVersion": "bedrock-conversation-2024",
  "messages": [
    {
      "role": "user",
      "content": [
        {
          "text": "What are the ways in which a customer can experience issues during checkout on Amazon?"
        },
        {
          "video": {
            "format": "mp4",
            "source": {
              "s3Location": {
                "uri": "s3://my-bucket-name/path/to/videos/customer_service_debugging.mp4",
                "bucketOwner": "123456789012"
              }
            }
          }
        }
      ]
    }
  ],
  {
```

```

    "role": "assistant",
    "content": [
      {
        "reasoningContent": {
          "reasoningText": {
            "text": "I need to find the relevant section in the video to answer the
question."
          }
        }
      },
      {
        "text": "Customers can experience issues with 1. Data entry, 2. Payment
methods, 3. Connectivity while placing the order. Which one would you like to dive
into?"
      }
    ]
  }
]
}

```

## 视频理解限制

| 限制       | Details                           |
|----------|-----------------------------------|
| 最大视频大小   | 50 MB                             |
| 视频最长时长   | 15 分钟                             |
| 单个样本视频数量 | 每个样本仅允许包含一个视频，不支持在同一样本中使用多个视频     |
| 模态混合     | 单个样本可包含视频与文本，但不可将视频与其他模态（图像、文档）混用 |

## 支持的视频格式

- MOV
- MKV
- MP4

## 视频理解最佳实践

- 保持视频简洁明了，聚焦与任务相关的内容
- 确保视频画质清晰，便于模型提取有效信息
- 提出的问题应明确指向视频中的特定内容
- 提供覆盖不同视频类型与问题格式的多样化样本

## 推理模式与非推理模式

### 理解推理内容

推理内容（亦称思维链）会记录模型在生成最终答案前的中间思考步骤。在 `assistant` 轮次中，可通过 `reasoningContent` 字段加入这些推理轨迹。

### 损失计算方式

- 包含推理内容：训练损失同时计入推理词元和最终输出词元
- 不含推理内容：训练损失仅基于最终输出词元计算

您可在多轮对话的多个助手轮次中添加 `reasoningContent`。

### 格式规范

- 推理内容使用纯文本
- 除非任务明确要求，否则避免使用 `<thinking>` 和 `</thinking>` 等标记标签
- 确保推理内容清晰，并与问题求解过程相关

### 何时启用推理模式

在以下场景中，在训练配置里设置 `reasoning_enabled: true`：

- 训练数据包含推理词元
- 希望模型在生成最终输出前先产生思考词元
- 需要在复杂推理任务上获得更优性能

允许在 `reasoning_enabled = true` 的情况下，使用非推理数据集训练 Nova。但是，这样做可能会导致模型丧失推理能力，因为 Nova 主要学习数据中呈现的应答方式，而非执行推理过

程。如果希望使用非推理数据集训练，同时在推理阶段保留推理能力，则可在训练时关闭推理 (`reasoning_enabled = false`)，并在推理时启用。这种方式虽能在推理时使用推理，但无法保证效果优于不启用推理的推理方式。一般建议：使用推理数据集时，训练与推理均启用推理；使用非推理数据集时，两者均关闭推理。

在以下场景中，设置 `reasoning_enabled: false`：

- 训练数据不包含推理词元
- 训练任务较为简单，无需显式推理步骤
- 希望优化速度并减少词元使用量

### 生成推理数据

如果数据集缺失推理轨迹，可借助 Nova Premier 等具备推理能力的模型来生成。将输入-输出对提供给模型，记录其推理过程，从而构建包含推理内容的增强型数据集。

### 使用推理词元进行训练

启用推理模式进行训练时，模型会学习将内部推理过程与最终答案分离开。训练过程：

- 将数据组织为三元组结构：输入、推理和答案
- 基于推理词元和答案词元的标准下一词元预测损失进行优化
- 引导模型在生成回复前先进行内部逻辑推理

### 优质推理内容的要素

高质量的推理内容应包含以下要素：

- 中间思考与分析
- 逻辑推导与推理步骤
- 分步解决问题的方法
- 推理步骤与结论之间的明确关联

这些要素有助于模型掌握“先思考，后作答”的能力。

## 数据集准备指南

### 规模与质量

- 推荐样本量：2000 – 10000 个样本
- 最小样本量：200
- 优先级：质量胜于数量。确保样本准确、标注规范
- 应用贴合度：数据集应尽可能贴近实际使用案例

### 多样性

样本需具备多样性，满足以下要求：

- 覆盖所有预期输入类型
- 包含不同难度级别的样本
- 纳入边界情况与各类变体
- 避免模型过拟合于单一模式

### 输出格式

在助手响应中明确指定所需输出格式：

- JSON 结构
- 表
- CSV 格式
- 应用程序专属的自定义格式

### 多回合对话

使用多轮对话数据集时请注意：

- 损失仅基于助手轮次计算，而非用户轮次
- 每个助手响应都应格式正确
- 保持各轮对话风格与格式一致

## 质量检查清单

- 数据集规模充足 ( 2000 – 10000 个样本 )
- 涵盖所有使用案例的多样化样本
- 输出格式清晰统一
- 标签与标注准确无误
- 贴合实际生产场景
- 无矛盾或歧义内容

## 上传数据

数据集需上传到 SageMaker 训练作业可访问的存储桶。有关设置相应权限的信息，请参阅[先决条件](#)。

## 启动训练作业

### 选择超参数并更新配方

Nova 2.0 的设置方式与 Nova 1.0 基本一致。将输入数据上传到 S3 后，使用 [SageMaker HyperPod Recipes](#) 中 fine-tuning 文件夹下的配方。对于 Nova 2.0，以下是一些可以根据使用案例更新的关键超参数。以下是 Nova 2.0 SFT PEFT 配方的样本。对于容器映像 URI，请使用 `708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-TJ-SFT-V2-latest` 来运行 SFT 微调作业。

请使用 v2.254.1 版本的 SageMaker AI PySDK，确保与 Nova 训练严格兼容。将 SDK 升级到 v3.0 版本会导致破坏性变更。对 SageMaker AI PySDK v3 的支持即将推出。

## 示例输入

```
!pip install sagemaker==2.254.1
```

```
run:
  name: {peft_recipe_job_name}
  model_type: amazon.nova-2-lite-v1:0:256k
  model_name_or_path: {peft_model_name_or_path}
  data_s3_path: {train_dataset_s3_path} # SageMaker HyperPod (SMHP) only and not
  compatible with SageMaker Training jobs. Note replace my-bucket-name with your real
  bucket name for SMHP job
  replicas: 4 # Number of compute instances for training, allowed
  values are 4, 8, 16, 32
```

```

output_s3_path: "" # Output artifact path (Hyperpod job-specific; not
compatible with standard SageMaker Training jobs). Note replace my-bucket-name with
your real bucket name for SMHP job

training_config:
  max_steps: 10 # Maximum training steps. Minimal is 4.
  save_steps: 10 # How many training steps the checkpoint will be
saved. Should be less than or equal to max_steps
  save_top_k: 1 # Keep top K best checkpoints. Note supported only
for SageMaker HyperPod jobs. Minimal is 1.
  max_length: 32768 # Sequence length (options: 8192, 16384, 32768
[default], 65536)
  global_batch_size: 32 # Global batch size (options: 32, 64, 128)
  reasoning_enabled: true # If data has reasoningContent, set to true;
otherwise False

  lr_scheduler:
    warmup_steps: 15 # Learning rate warmup steps. Recommend 15% of
max_steps
    min_lr: 1e-6 # Minimum learning rate, must be between 0.0 and 1.0

  optim_config: # Optimizer settings
    lr: 1e-5 # Learning rate, must be between 0.0 and 1.0
    weight_decay: 0.0 # L2 regularization strength, must be between 0.0
and 1.0
    adam_beta1: 0.9 # Exponential decay rate for first-moment estimates,
must be between 0.0 and 1.0
    adam_beta2: 0.95 # Exponential decay rate for second-moment
estimates, must be between 0.0 and 1.0

  peft: # Parameter-efficient fine-tuning (LoRA)
    peft_scheme: "lora" # Enable LoRA for PEFT
    lora_tuning:
      alpha: 64 # Scaling factor for LoRA weights ( options: 32, 64,
96, 128, 160, 192),
      lora_plus_lr_ratio: 64.0

```

该配方包含的超参数与 Nova 1.0 基本一致。核心超参数如下：

- `max_steps`：希望作业运行的步数。通常，1 轮 epoch（遍历完整数据集一次）的计算方式为：步数 = 数据样本数/全局批次大小。步数越大、全局批次越小，任务运行耗时越长。
- `reasoning_enabled`：控制数据集的推理模式。选项：

- `true` : 启用推理模式 ( 相当于高强度推理 )
- `false` : 禁用推理模式

注意 : 在 SFT 中 , 无法精细控制推理强度。设置 `reasoning_enabled: true` 即启用完整推理能力。

- `peft.peft_scheme` : 设置为“lora”即启用基于 PEFT 的微调。设置为 `null` ( 不带引号 ) 即启用全秩微调。

## 启动训练作业

```
from sagemaker.pytorch import PyTorch

# define OutputDataConfig path
if default_prefix:
    output_path = f"s3://{bucket_name}/{default_prefix}/{sm_training_job_name}"
else:
    output_path = f"s3://{bucket_name}/{sm_training_job_name}"

output_kms_key = "<KMS key arn to encrypt trained model in Amazon-owned S3 bucket>" #
optional, leave blank for Amazon managed encryption

recipe_overrides = {
    "run": {
        "replicas": instance_count, # Required
        "output_s3_path": output_path
    },
}

estimator = PyTorch(
    output_path=output_path,
    base_job_name=sm_training_job_name,
    role=role,
    disable_profiler=True,
    debugger_hook_config=False,
    instance_count=instance_count,
    instance_type=instance_type,
    training_recipe=training_recipe,
    recipe_overrides=recipe_overrides,
    max_run=432000,
    sagemaker_session=sagemaker_session,
    image_uri=image_uri,
```

```
output_kms_key=output_kms_key,  
tags=[  
    {'Key': 'model_name_or_path', 'Value': model_name_or_path},  
]  
)  
  
print(f"\nsm_training_job_name:\n{sm_training_job_name}\n")  
print(f"output_path:\n{output_path}")
```

```
from sagemaker.inputs import TrainingInput  
  
train_input = TrainingInput(  
    s3_data=train_dataset_s3_path,  
    distribution="FullyReplicated",  
    s3_data_type="Converse",  
)  
  
estimator.fit(inputs={"validation": val_input}, wait=False)
```

### Note

Nova 2.0 的监督式微调不支持传递验证数据集。

要启动作业，请执行以下操作：

- 更新配方中的数据路径与超参数
- 运行笔记本中指定的代码单元，提交训练作业

笔记本将自动处理作业提交，并提供状态跟踪功能。

## 使用 Amazon Nova 模型进行强化微调 ( RFT )

### 概述

#### 什么是 RFT ?

强化微调 ( RFT ) 通过基于反馈信号 ( 即指示模型表现优劣的可衡量分数或奖励 ) 的训练来提升模型性能，而不是基于精确的正确答案。与从输入-输出对中学习的监督式微调不同，RFT 使用奖励函数来评测模型响应，并迭代优化模型以获得最高奖励。这种方法适用于难以定义精确输出但能可靠衡量响应质量的任务。

## 何时使用 RFT

当您能够定义清晰、可衡量的成功标准，但难以提供精确的正确输出用于训练时，便可使用 RFT。该方法适用于：

- 质量具有主观性或多维度的任务，如创意写作、代码优化或复杂推理
- 存在多种可行方案，但部分方案明显更优的场景
- 需要迭代改进、个性化或遵循复杂业务规则的应用
- 收集高质量标注样本成本过高或不切实际的情况

## 最佳使用案例

在输出质量可客观衡量、但最优应答难以预先定义的领域，RFT 表现优异：

- 数学问题求解和代码生成
- 科学推理与结构化数据分析
- 需要逐步推理或多轮问题求解的任务
- 兼顾多个目标（准确率、效率、风格）的应用
- 可通过执行结果或性能指标以编程方式验证成功与否的场景

## 支持的模型

Nova Lite 2.0

## 数据格式概述

RFT 训练数据必须遵循 OpenAI 强化微调[格式](#)。每个训练样本都是一个 JSON 对象，其中包含：

- 包含 system 和 user 角色对话轮次的 messages 数组
- 包含用于奖励计算的期望输出或评测标准的 reference\_answer 字段

## 目前的局限性

- 仅文本

## 数据格式示例

每个样本需单独占一行，JSONL 文件中每行对应一个 JSON 对象。

## Chemistry problem

```
{
  "id": "chem-01",
  "messages": [
    {
      "role": "system",
      "content": "You are a helpful chemistry assistant"
    },
    {
      "role": "user",
      "content": "Calculate the molecular weight of caffeine (C8H10N4O2)"
    }
  ],
  "reference_answer": {
    "molecular_weight": 194.19,
    "unit": "g/mol",
    "calculation": "8(12.01) + 10(1.008) + 4(14.01) + 2(16.00) = 194.19"
  }
}
```

## Math problem

```
{
  "id": "sample-001", // Optional
  "messages": [
    {
      "role": "system",
      "content": "You are a math tutor"
    },
    {
      "role": "user",
      "content": "Solve:  $2x + 5 = 13$ "
    }
  ],
  "reference_answer": {
    "solution": " $x = 4$ ",
    "steps": [" $2x = 13 - 5$ ", " $2x = 8$ ", " $x = 4$ "]
  }
}
```

## Code problem

```
{
  "id": "code-002",
  "messages": [
    {
      "role": "system",
      "content": "You are a helpful programming assistant"
    },
    {
      "role": "user",
      "content": "Write a Python function that reverses a string without using
built-in reverse methods"
    }
  ],
  "reference_answer": {
    "code": "def reverse_string(s): \n    result = '' \n    for i in range(len(s)
- 1, -1, -1): \n        result += s[i] \n    return result",
    "test_cases": [
      {
        "input": "hello",
        "expected_output": "olleh"
      },
      {
        "input": "",
        "expected_output": ""
      },
      {
        "input": "a",
        "expected_output": "a"
      },
      {
        "input": "Python123",
        "expected_output": "321nohtyP"
      }
    ]
  },
  "all_tests_pass": true
}
```

`reference_answer` 字段包含期望输出或评测标准，供奖励函数对模型响应进行评分。其不限于结构化输出，可采用任何有助于奖励函数评测质量的格式。

## 数据集规模建议

### 起步要求

- 至少 100 个训练样本
- 至少 100 个评测样本

### 评测优先的方法

在投入大规模 RFT 训练之前，请先评测模型的基线性能：

- 性能优异（奖励大于 95%）：无需进行 RFT，模型表现足够好
- 性能极差（奖励为 0%）：先切换到 SFT，建立基本能力
- 性能中等：适合采用 RFT 训练

从小规模数据集开始，可实现以下效果：

- 验证奖励函数无错误
- 确认 RFT 适合于使用案例
- 尽早发现并修复问题
- 在扩展规模前测试 workflow

验证通过后，即可扩展到更大规模的数据集，进一步提升模型性能。

### 有效训练数据的特征

#### 清晰性与一致性

优质的 RFT 样本需要清晰、无歧义的输入数据，以便能够针对不同的模型输出进行准确的奖励计算。数据中应避免以下干扰：

- 格式不一致
- 标签或指令相互矛盾
- 提示词含义模糊
- 参考答案互相冲突

任何歧义都会误导训练过程，导致模型学习到非预期的行为。

## 多样性

数据集应覆盖生产环境中的各类使用案例，确保模型在真实环境中表现稳健。包含：

- 不同的输入格式和边界情况
- 从日志和用户分析中映射实际的生产使用模式
- 跨用户类型、地理区域和季节性变化的采样
- 包含从简单到复杂的问题难度级别

## 奖励函数注意事项

设计奖励函数，实现高效训练：

- 在几秒钟内执行完毕（而非几分钟）
- 使用 Lambda 实现高效并行
- 返回一致且可靠的分数
- 妥善处理不同类型的模型输出

快速且可扩展的奖励函数可支持快速迭代，并在实验中实现高成本效益。

## 其它属性

RFT 数据格式支持核心架构要求（`messages` 和 `reference_answer`）之外的自定义字段。这种灵活性支持您添加奖励函数完成有效评测所需的任意附加数据。

### Note

无需在配方中进行配置，该数据格式原生支持附加字段。只需将其包含在训练数据的 JSON 中，即可通过 `metadata` 字段传递给奖励函数。

## 常见附加属性

元数据字段示例：

- `task_id`：用于跟踪的唯一标识符
- `difficulty_level`：问题复杂度指示符
- `domain`：主题领域或类别

- `expected_reasoning_steps` : 解题过程中的步骤数

### 包含其他属性的示例

```
{
  "messages": [
    {
      "role": "system",
      "content": "You are a math tutor"
    },
    {
      "role": "user",
      "content": "Solve:  $2x + 5 = 13$ "
    }
  ],
  "reference_answer": {
    "solution": "x = 4",
    "steps": [" $2x = 13 - 5$ ", " $2x = 8$ ", " $x = 4$ "]
  },
  "task_id": "algebra_001",
  "difficulty_level": "easy",
  "domain": "algebra",
  "expected_reasoning_steps": 3
}
```

这些附加字段将在评测期间传递给奖励函数，从而实现针对特定使用案例量身定制的复杂评分逻辑。

### 训练配置

#### 配方示例

```
# Note:
# This recipe can run on p5.48xlarge and p5en.48xlarge instance types.
run:
  name: "my-rft-run" # Unique run name (appears in logs/artifacts).
  model_type: amazon.nova-2-lite-v1:0:256k
  model_name_or_path: nova-lite-2/prod
  data_s3_path: s3://<bucket>/<data file> # Training dataset in JSONL;
  replicas: 4
  reward_lambda_arn: arn:aws:lambda:<region>:<account-id>:function:<function-name>

## MLFlow configs
```

```

mlflow_tracking_uri: "" # Required for MLFlow
mlflow_experiment_name: "my-rft-experiment" # Optional for MLFlow. Note: leave this
field non-empty
mlflow_run_name: "my-rft-run" # Optional for MLFlow. Note: leave this field non-empty

## SMTJ GRPO Training specific configs
training_config:
  max_length: 8192 # Context window (tokens) for inputs
+prompt;
  global_batch_size: 16 # Total samples per optimizer step
  across_all_replicas (16/32/64/128/256).
  reasoning_effort: high # Enables reasoning mode high / low /
or null for non-reasoning

  rollout: # How responses are generated for GRPO/
advantage calc.
  advantage_strategy:
    number_generation: 2 # N samples per prompt to estimate
advantages (variance vs cost).
  generator:
    max_new_tokens: 6000 # Cap on tokens generated per sample
    set_random_seed: true # Seed generation for reproducibility
  across_runs.
    temperature: 1 # Softmax temperature;
    top_k: 1 # Sample only from top-K logits
  rewards:
    preset_reward_function: null # Usage of preset reward functions
[exact_match]
  api_endpoint:
    lambda_arn: arn:aws:lambda:<region>:<account-id>:function:<function-name>
    lambda_concurrency_limit: 12 # Max concurrent Lambda invocations
(throughput vs. throttling).

  trainer:
    max_steps: 2 # Steps to train for. One Step =
global_batch_size
    save_steps: 5
    test_steps: 1

  # RL parameters
  ent_coeff: 0.0 # A bonus added to the policy loss that
rewards higher-output entropy.
  kl_loss_coef: 0.001 # Weight on the KL penalty between the
actor (trainable policy) and a frozen reference model

```

```
optim_config:                # Optimizer settings
  lr: 5e-5                    # Learning rate
  weight_decay: 0.0          # L2 regularization strength (0.0-1.0)
  adam_beta1: 0.9
  adam_beta2: 0.95

peft:                        # Parameter-efficient fine-tuning (LoRA)
  peft_scheme: "lora"        # Enable LoRA for PEFT
  lora_tuning:
    alpha: 32
    lora_plus_lr_ratio: 64.0 # LoRA+ learning rate scaling factor (0.0-100.0)
```

## 使用 LLM-as-a-judge 的 RFT 训练

### 概述

大语言模型 ( LLM ) 越来越多地被用作强化微调 ( RFT ) 工作流程中的评判工具，提供自动化的奖励信号来指导模型优化。在这种方法中，LLM 会根据指定标准对模型输出进行评测 ( 包括正确性、质量、风格一致性或语义等效性等 )，并分配用于驱动强化学习过程的奖励值。

对于传统奖励函数难以通过编程定义的任务，该方法尤为实用。例如判断不同表达形式 ( 如“1/3”“0.333”和“三分之一” ) 是否语义等效，或评测连贯性、相关性等细微特征。通过将基于 LLM 的评判机制作为奖励函数，可将 RFT 扩展到复杂领域，无需大量人工标注，进而在传统对齐问题之外的各类场景中，实现模型的快速迭代与持续优化。

### 推理模式选择

#### 可用模式

- none : 无推理 ( 省略 reasoning\_effort 字段 )
- low : 最小推理开销
- high : 最大推理能力 ( 指定了 reasoning\_effort 时，此为默认值 )

#### Note

RFT 无中等选项。如果配置中不含 reasoning\_effort 字段，则禁用推理。启用推理时，建议将 max\_new\_tokens 设置为 32768，以容纳更长的推理输出内容。

## 何时使用各个模式

在以下场景使用高强度推理：

- 复杂分析任务
- 数学问题求解
- 多步逻辑推导
- 逐步思考能带来价值的任务

在以下场景使用无推理（省略 reasoning\_effort）或低强度推理：

- 简单事实查询
- 直接分类
- 速度与成本优化
- 直接问答

## 成本与性能权衡

较高强度推理模式会导致：

- 训练时间和成本增加
- 推理延迟和成本增加
- 模型在复杂推理任务上的能力提升

## 验证 LLM 评判工具

在生产中部署 LLM-as-a-judge 之前，请验证评判工具模型的评测是否与人类判断一致。这涉及：

- 测量 LLM 评判工具与人工评测员在任务代表性样本上的一致率
- 确保 LLM 与人类评测的一致率达到或超过人类之间的一致率
- 识别评判工具模型中可能存在的偏差
- 建立信心，确保奖励信号能按预期方向引导模型

此验证步骤有助于确保自动化评测过程能够产生符合生产质量标准的模型。

## LLM 评判工具的 Lambda 配置

采用 LLM-as-a-judge，是对使用 Lambda 函数实现可验证奖励强化学习 ( RLVR ) 的扩展。在 Lambda 函数内部，需调用 Amazon Bedrock 中托管的任一模型。

重要配置要求：

| 配置                 | 要求           | Details                                                         |
|--------------------|--------------|-----------------------------------------------------------------|
| Amazon Bedrock 吞吐量 | 充足的配额        | 确保所使用的 Amazon Bedrock 模型吞吐量配额足以满足训练工作负载                         |
| Lambda 超时          | 延长超时         | 将 Lambda 函数超时配置为最多 15 分钟。默认设置为 3 秒，不足以满足 Amazon Bedrock 模型的响应需求 |
| Lambda 并发          | 提高并发         | Lambda 在训练期间会被并行调用。提高并发以提供最大程度的可用吞吐量                            |
| 配方配置               | 匹配 Lambda 设置 | 并发限制必须在配方中进行配置                                                  |

## 创建与运行作业

### 启动训练作业

使用 SageMaker 训练作业笔记本模板：<https://docs.aws.amazon.com/sagemaker/latest/dg/nova-fine-tuning-training-job.html#nova-model-training-jobs-notebook>

### 实例要求

容器同时支持全秩和 LoRA 训练：

- LoRA 训练：2/4/6/8 个 p5.48xlarge 或 p5en.48xlarge 实例
- 全秩训练：2/4/6/8 个 p5.48xlarge 实例 ( 必需 )

### 监控训练

训练日志包含每一步的全面指标。主要指标类别：

### 奖励指标

- critic/rewards/mean、critic/rewards/max、critic/rewards/min : 奖励分布
- val-score/rewards/mean@1 : 验证奖励

### 模型行为

- actor/entropy : 策略多样性 ( 值越高表示探索性越强 )

### 训练健康度

- actor/pg\_loss : 策略梯度损失
- actor/pg\_clipfrac : 裁剪更新频率
- actor/grad\_norm : 梯度范数

### 响应特征

- prompt\_length/mean、prompt\_length/max、prompt\_length/min : 输入词元统计
- response\_length/mean、response\_length/max、response\_length/min : 输出词元统计
- response/aborted\_ratio : 不完整生成率 ( 0 表示全部完成 )

### 性能

- perf/throughput : 训练吞吐量
- perf/time\_per\_step : 每步训练时间
- timing\_per\_token\_ms/\* : 每个词元的处理时间

### 资源使用

- perf/max\_memory\_allocated\_gb、perf/max\_memory\_reserved\_gb : GPU 内存
- perf/cpu\_memory\_used\_gb : CPU 内存

### 使用经过微调的模型

训练完成后，最终模型检查点将保存到指定的输出位置。检查点路径可在以下位置获取：

- 训练日志

- 输出 Amazon S3 位置 ( 由笔记本中的 `output_s3_uri` 定义 ) 中的 `manifest.json` 文件

## 限制与最佳实践

### 限制

- Lambda 超时：奖励函数须在 15 分钟内执行完毕 ( 防止进程失控并控制成本 )
- 仅支持单轮对话：不支持多轮对话
- 数据要求：需具备充足的多样性；在稀疏奖励场景下 ( 正例占比  $< 5\%$  ) 训练难度较高
- 计算成本：比监督式微调更昂贵
- 无多模态数据：仅支持文本数据类型

### 最佳实践

#### 从小规模起步

- 从 100 – 200 个样本开始
- 验证奖励函数正确性
- 根据结果逐步扩展

#### 训练前评测

- 在 RFT 之前测试基线模型性能
- 如果奖励始终为 0%，建议先通过 SFT 训练构建基础能力
- 若奖励占比  $> 95\%$ ，则可能无需进行 RFT

#### 监控训练

- 跟踪平均奖励分数和分布
- 注意过拟合 ( 训练奖励上升，而验证奖励下降 )
- 留意异常模式：
  - 奖励在 0.15 以下趋于平稳
  - 奖励方差随时间逐步增大
  - 验证性能下降

## 优化奖励函数

- 在几秒钟内执行完毕（而非几分钟）
- 尽量减少外部 API 调用
- 使用高效算法
- 实现完善的错误处理机制
- 充分利用 Lambda 的并行扩展能力

## 迭代策略

如果奖励没有提升：

- 调整奖励函数设计
- 提升数据集多样性
- 添加更多代表性样本
- 确保奖励信号清晰一致

## 高级能力：Nova Forge

对于需要突破标准 RFT 限制的高级能力的用户，Nova Forge 作为付费订阅服务提供，包括：

- 多轮对话支持
- 执行时间超过 15 分钟的奖励函数
- 额外的算法和调优选项
- 自定义训练配方修改
- 最先进的人工智能技术

Nova Forge 在 SageMaker HyperPod 上运行，旨在支持企业客户构建自己的前沿模型。

## 实用命令与技巧

您可使用一系列[可观测性脚本](#)，帮助监控训练作业的状态和进度。

可用脚本包括：

- 为训练作业状态更新启用电子邮件通知

- 根据作业配置获取训练时间估算
- 获取进行中作业的预期训练时间近似值

## 安装

### Note

在使用以下任何脚本之前，请务必刷新 AWS 凭证。

```
pip install boto3
git clone https://github.com/aws-samples/amazon-nova-samples.git
cd amazon-nova-samples/customization/SageMakerUilts/SageMakerJobsMonitoring/
```

## 基本用法

```
# Enabling email notifications for training job status updates
python enable_sagemaker_job_notifs.py --email test@amazon.com test2@gmail.com --region
us-east-1 --platform SMTJ
```

Creating resources.....

Please check your email for a subscription confirmation email, and click 'Confirm subscription' to start receiving job status email notifications!

You'll receive the confirmation email within a few minutes.

```
# Obtaining training time estimates based on job configurations
python get_training_time_estimate.py
```

```
# Obtaining approximations for how long training is expected to take for in-progress
jobs
python get-training-job-progress.py --region us-east-1 --job-name my-training-job --
num-dataset-samples 1000
```

有关更多详细信息和示例，请参阅[此处](#)。

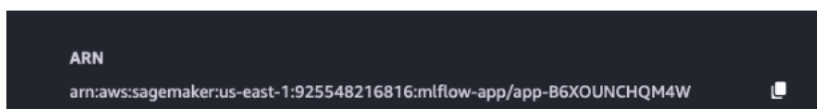
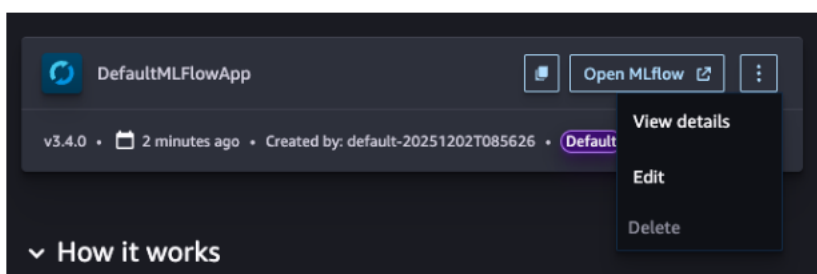
## 监控迭代进度

您可以通过 MLflow 跟踪指标。

## Nova 自定义 – 为 SageMaker HyperPod 配置 MLflow

要让 SageMaker HyperPod 环境将指标输出到 MLflow，需要完成一些额外配置。

1. 打开 Amazon SageMaker AI
2. 选择 SageMaker Studio
  - a. 如果已创建配置文件，选择“打开 Studio”。
  - b. 如果未创建配置文件，选择“创建 SageMaker 域”进行创建
3. 选择 MLflow。如果未创建任何 MLflow 应用程序，请选择“创建 MLflow 应用程序”
4. 在 MLflow 应用程序上点击复制/粘贴按钮或“查看详细信息”菜单项，获取 ARN。提交训练作业时需用到该 ARN。



5. 在 HyperPod 集群的执行角色中，添加以下策略。这将允许 HyperPod 集群调用 MLflow API 来发布指标。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sagemaker-mlflow:*",
      "Resource": [
        "arn:aws:sagemaker:us-east-1:372836560492:mlflow-app/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
```

```

        "sagemaker:ListMlflowTrackingServers",
        "sagemaker:CallMlflowAppApi"
    ],
    "Resource": "*"
}
]
}

```

## 通过 CLI 提交作业

需指定 4 个新增的覆盖参数，可在命令行中指定，也可在 yaml 配方文件中配置。

1. `mlflow_tracking_uri` : MLflow 应用程序的 ARN
2. `mlflow_experiment_name` : 本次实验运行的名称
3. `mlflow_experiment_name` : 指标在 MLflow 中存储对应的实验名称
4. `mlflow_run_name` : 本实验的名称

## 命令行

```

--override-parameters '{"recipes.run.mlflow_tracking_uri": "arn:aws:sagemaker:us-east-1:925548216816:mlflow-app/app-B6X0UNCHQM4W", "recipes.run.mlflow_experiment_name": "myuser-sft-lora-exp1", "recipes.run.mlflow_run_name": "myuser-sft-lora-exp1-202512181940"}'

```

## yaml :

```

## Run config
run:
  mlflow_tracking_uri: "arn:aws:sagemaker:us-east-1:925548216816:mlflow-app/app-B6X0UNCHQM4W"
  mlflow_experiment_name: "myuser-sft-lora-exp1"
  mlflow_run_name: "myuser-sft-lora-exp1-202512181940"

```

## 通过 SageMaker Studio UI 提交作业

MLflow 集成已内置到 SageMaker Studio UI 中。提交训练作业时，只需指明要使用的 MLflow 应用程序实例。

1. 在 SageMaker Studio 中，依次进入：“模型”>“Nova 2.0 Lite”>“自定义”>“通过 UI 自定义”。
2. 展开“高级配置”部分
3. 选择用于接收训练指标的 MLflow 应用程序。您还可以在此设置实验名称和实验运行名称。

**MLflow**

**MLflow Resource \***  
MLflow resource selected for experiment tracking.

DefaultMLFlowApp

**Experiment name**  
Default name applied. Enter a new name if desired.

myuser-sft-lora-exp1

**Experiment run**  
The run name is autogenerated by default. Specify a new name if desired.

myuser-sft-lora-exp1-1766499113356

## 通过 AWS CLI 提交作业

如果您使用 AWS CLI，则必须创建一个 MLflow 应用程序，并将其作为输入传递给训练作业 API 请求。

```
mlflow_app_name="<enter your MLflow app name>"
role_arn="<enter your role ARN>"
bucket_name="<enter your bucket name>"
region="<enter your region>"

mlflow_app_arn=$(aws sagemaker create-mlflow-app \
  --name $mlflow_app_name \
  --artifact-store-uri "s3://$bucket_name" \
  --role-arn $role_arn \
  --region $region)
```

## 创建 MLflow 应用程序

使用 Studio UI：如果您通过 Studio UI 创建训练作业，系统会自动创建默认 MLflow 应用程序，并在“高级选项”下默认选择该应用程序。

使用 CLI：如果您使用 CLI，则必须创建一个 MLflow 应用程序，并将其作为输入传递给训练作业 API 请求。

```
mlflow_app_name="<enter your MLflow app name>"
role_arn="<enter your role ARN>"
bucket_name="<enter your bucket name>"
region="<enter your region>"

mlflow_app_arn=$(aws sagemaker create-mlflow-app \
  --name $mlflow_app_name \
  --artifact-store-uri "s3://$bucket_name" \
  --role-arn $role_arn \
  --region $region)
```

## 访问 MLflow 应用程序

使用 CLI：创建预签名 URL 以访问 MLflow 应用程序 UI：

```
aws sagemaker create-presigned-mlflow-app-url \
  --arn $mlflow_app_arn \
  --region $region \
  --output text
```

使用 Studio UI：Studio UI 会展示存储在 MLflow 中的关键指标，并提供指向 MLflow 应用程序 UI 的链接。

## 要跟踪的关键指标

通过迭代监控以下指标，以评测模型优化效果并跟踪作业进度：

### 针对 SFT

- 训练损失曲线
- 已处理样本数量及处理耗时
- 预留测试集上的性能准确率

- 格式合规性 ( 如有效 JSON 输出率 )
- 领域专属评估数据上的困惑度

### 针对 RFT

- 训练过程中的平均奖励分数
- 奖励分布 ( 高奖励响应占比 )
- 验证奖励趋势 ( 关注过拟合情况 )
- 任务专属成功率 ( 如代码执行通过率、数学题准确率 )

### 一般性问题

- 迭代间基准性能变化
- 代表性样本的人工评估分数
- 生产环境指标 ( 若采用迭代式部署 )

## 停止迭代的判定条件

满足以下任一情况即可停止迭代：

- 性能趋于平稳：继续训练无法显著提升目标指标
- 切换训练技术有效：若某一技术效果趋于平稳，可尝试切换 ( 如 SFT → RFT → SFT ) 以突破性能瓶颈
- 达到目标指标：已满足预设的成功标准
- 检测到性能退化：新迭代导致模型效果下降 ( 参见下文回滚流程 )

有关详细的评估流程，请参阅评估部分。

## 评估经过 SageMaker AI 训练的模型

评估过程旨在根据基准测试或自定义数据集评测训练模型的性能。评估过程通常涉及以下步骤：创建指向训练模型的评估配方、指定评估数据集和指标、提交单独的评估作业，以及根据标准基准测试或自定义数据进行评估。评估过程将输出存储在您的 Amazon S3 存储桶中的性能指标。

### Note

本主题中描述的评估过程是一个离线过程。该模型根据固定的基准测试进行测试，并提供预定义的答案，而非实时或者通过实时用户互动进行评测。对于实时评估，您可以在模型部署到 Amazon Bedrock 之后通过调用 [Amazon Bedrock](#) 运行时 API 对模型进行测试。

## 主题

- [先决条件](#)
- [可用的基准测试任务](#)
- [特定于评估的配置](#)
- [运行评估训练作业](#)
- [评测和分析评估结果](#)
- [评估最佳实践和疑难解答](#)
- [可用子任务](#)
- [推理模型评测](#)
- [RFT 评测](#)
- [实现奖励函数](#)

## 先决条件

在开始评估训练作业之前，请注意具备以下内容：

- 一款经过 SageMaker AI 训练、您想要评估其性能的 Amazon Nova 模型。
- 用于评估的基本 Amazon Nova 配方。有关更多信息，请参阅 [???](#)。

## 可用的基准测试任务

一个可用的示例代码包，用于演示如何使用适用于 Amazon Nova 的 SageMaker AI 模型评测特征计算基准评测指标。要获取代码包，请参阅 [sample-Nova-lighteval-custom-task](#)。

下面是支持的可用行业标准基准测试列表。您可以在 `eval_task` 参数中指定以下基准测试。

## 可用的模型评测基准测试

| 基准            | 模式 | 说明                                      | 指标          | Strategy | 可用子任务 |
|---------------|----|-----------------------------------------|-------------|----------|-------|
| mmlu          | 文本 | 多任务语言理解：考核 57 个科目的知识。                   | 准确性         | zs_cot   | 是     |
| mmlu_pro      | 文本 | MMLU (专业子集)，专注于法律、医学、会计和工程等专业领域。        | 准确性         | zs_cot   | 否     |
| bbh           | 文本 | 高级推理任务：一系列具有挑战性的问题，用于考核更高级别的认知和解决问题的能力。 | 准确性         | fs_cot   | 是     |
| gpqa          | 文本 | 一般物理问题解答：评测对物理概念和相关问题解决能力的理解情况。         | 准确性         | zs_cot   | 否     |
| math          | 文本 | 数学问题解决：衡量在代数、微积分及应用题等领域的数学推理能力。         | exact_match | zs_cot   | 是     |
| strong_reject | 文本 | 质量控制任务：测试模型                             | deflection  | zs       | 是     |

| 基准     | 模式 | 说明                               | 指标  | Strategy | 可用子任务 |
|--------|----|----------------------------------|-----|----------|-------|
|        |    | 检测和拒绝不当、有害或不正确内容的 ability。       |     |          |       |
| ifeval | 文本 | 指令跟随评估：衡量模型遵循给定指令并按照规定完成任务的准确程度。 | 准确性 | zs       | 否     |

| 基准     | 模式         | 说明                                                                                                                                                                   | 指标  | Strategy | 可用子任务 |
|--------|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|----------|-------|
| gen_qa | 多模态 ( 图像 ) | 自定义数据集评估：让您可以提供自己的数据集进行基准测试，将模型输出与参考答案及 ROUGE 和 BLEU 等指标进行比较。gen_qa 支持基于 Amazon Nova Lite 或 Amazon Nova Pro 的模型进行图像推理。其也支持自定义指标 Lambda 函数。（若需进行 RFT 评估，请使用 RFT 评估配方） | 全部  | gen_qa   | 否     |
| mmmu   | 多模态        | 大规模多学科多模态理解 ( MMMU ) : 大学水平的基准测试，包括 30 个学科的选择题与开放式问答题。                                                                                                               | 准确性 | zs_cot   | 是     |

| 基准           | 模式      | 说明                                                                              | 指标 | Strategy | 可用子任务 |
|--------------|---------|---------------------------------------------------------------------------------|----|----------|-------|
| llm_judge    | 文本      | LLM-as-a-Judge 偏好比较：使用 Nova Judge 模型来确定提示的成对响应之间的偏好（B 与 A 比较），计算 B 比 A 更受欢迎的概率。 | 全部 | judge    | 否     |
| mm_llm_judge | 多模态（图像） | 该全新基准评测的运行方式与上述基于文本的 llm_judge 一致。唯一区别在于其支持图像推理。                                | 全部 | judge    | 否     |

| 基准               | 模式 | 说明                                                                                                                                            | 指标          | Strategy | 可用子任务 |
|------------------|----|-----------------------------------------------------------------------------------------------------------------------------------------------|-------------|----------|-------|
| rubric_llm_judge | 文本 | Rubric Judge 是基于 Nova 2.0 Lite 构建的增强版 LLM-as-a-judge 评估模型。与仅提供偏好判定的 <a href="#">原始评判模型</a> 不同，Rubric Judge 可针对每个提示动态生成自定义评估准则，并在多个维度上给出细粒度评分。 | 全部          | judge    | 否     |
| aime_2024        | 文本 | AIME 2024 : 2024 年美国数学邀请赛 ( American Invitational Mathematics Examination )，用于考查高阶数学推理与问题求解能力                                                 | exact_match | zs_cot   | 否     |

| 基准                  | 模式 | 说明                                           | 指标          | Strategy | 可用子任务 |
|---------------------|----|----------------------------------------------|-------------|----------|-------|
| calendar_scheduling | 文本 | Natural Plan : 日程规划任务, 用于评估跨日期、跨人员的会议安排等规划能力 | exact_match | fs       | 否     |
| humaneval           | 文本 | HumanEval : 一个基准测试数据集, 旨在评估大型语言模型的代码生成能力     | pass@1      | zs       | 否     |

## 特定于评估的配置

以下是关于配方中关键组成部分的详细介绍, 以及有关如何根据您的使用案例对其进行修改的指导。

### 了解和修改配方

#### 常规运行配置

```
run:
  name: eval_job_name
  model_type: amazon.nova-2-lite-v1:0:256k
  model_name_or_path: nova-lite-2/prod # or s3://escrow_bucket/model_location
  replicas: 1
  data_s3_path: ""
  mlflow_tracking_uri: ""
  mlflow_experiment_name : ""
  mlflow_run_name : ""
```

- name : 评估作业的描述性名称。
- model\_type : 指定要使用的 Nova 模型变体。请勿手动修改此字段。选项包括 :
  - amazon.nova-micro-v1:0:128k

- amazon.nova-lite-v1:0:300k
- amazon.nova-pro-v1:0:300k
- amazon.nova-2-lite-v1:0:256k
- model\_name\_or\_path : 基本模型的路径或训练后检查点的 S3 路径。选项包括 :
  - nova-micro/prod
  - nova-lite/prod
  - nova-pro/prod
  - nova-lite-2/prod
- 训练后检查点的 S3 路径 ( s3:customer-escrow-111122223333-smtj-<unique\_id>/<training\_run\_name> )

#### Note

##### 评估训练后模型

要在 Nova SFT 训练作业之后评估训练后模型，请在成功运行训练作业后按照以下步骤操作。在训练日志的末尾，您将会看到“训练已完成”的日志消息。您还将在输出存储桶中找到一个包含检查点位置的 manifest.json 文件。此文件将位于输出 S3 位置的 output.tar.gz 文件中。要继续进行评估，请使用此检查点，将其设置为配方配置中 run.model\_name\_or\_path 的值。

- replica : 用于分布式推理 ( 跨多节点运行推理 ) 的计算实例数量。设置 replica > 1 可启用多节点推理，加速评估过程。若同时指定 instance\_count 和 replica，以 instance\_count 为准。请注意，多副本仅适用于 SageMaker AI 训练作业。
- data\_s3\_path : 输入数据集 Amazon S3 路径。此字段为必填字段，但应始终留空。
- mlflow\_tracking\_uri: ( 可选 ) MLflow 跟踪服务器地址 ( 仅在 SMHP 下需要 )
- mlflow\_experiment\_name: ( 可选 ) 实验名称，用于对相关的机器学习运行记录进行分组
- mlflow\_run\_name: ( 可选 ) 实验中某次特定训练运行的自定义名称

## 评估配置

```
evaluation:
  task: mmlu
  strategy: zs_cot
  subtask: abstract_algebra
```

```
metric: accuracy
```

- **task** : 指定要使用的评估基准测试或任务。支持的任务包括：
  - mmlu
  - mmlu\_pro
  - bbh
  - gpqa
  - math
  - strong\_reject
  - gen\_qa
  - ifeval
  - mmmu
  - llm\_judge
  - mm\_llm\_judge
  - rubric\_llm\_judge
  - aime\_2024
  - calendar\_scheduling
  - humaneval
- **strategy** : 定义评估方法。
  - **zs\_cot** : 零样本思维链，该提示方法激励大型语言模型逐步进行推理，而不提供明确的示例。
  - **fs\_cot** : 少样本思维链，该方法会先提供一些逐步推理的示例，然后再要求模型解决新问题。
  - **zs** : 零样本，该方法事先无需任何训练示例即可解决问题。
  - **gen\_qa** : 专为自带数据集而设的策略。
  - **judge** : 专门用于 Nova LLM as judge 和 mm\_llm\_judge 的策略。
- **subtask** : 可选。评估任务的具体组成部分。有关可用子任务的完整列表，请参阅[???](#)。
  - 在“可用的基准测试任务”中查看支持的子任务。
  - 如果没有子任务基准测试，则应移除此字段。
- **metric** : 要使用的评估指标。
  - **accuracy** : 正确答案的百分比。
  - **exact\_match** : 对于 math 基准测试，返回输入预测字符串与其引用完全匹配的速率。
  - **deflection** : 对于强烈拒绝基准测试，返回基本模型的相对偏差和差异显著性指标。

- all:

对于 gen\_qa、自带数据集基准测试，返回以下指标：

- rouge1：衡量生成的文本和参考文本之间的一元（单个词）重叠度。
- rouge2：衡量生成的文本和参考文本之间的二元（两个连续的词）重叠度。
- rougeL：衡量文本之间的最长公共子序列，匹配中可以存在间隙。
- exact\_match：二进制分数（0 或 1），用于指示生成的文本是否与参考文本逐个字符完全匹配。
- quasi\_exact\_match：与精确匹配类似，但更加宽松，通常忽略大小写、标点符号和空格差异。
- f1\_score：精度和查全率之间的调和平均数，衡量预测答案和参考答案之间的词语重叠度。
- f1\_score\_quasi：与 f1\_score 类似，但匹配更为宽松，使用忽略细微差异的标准化文本比较。
- bleu：衡量生成的文本和参考文本之间 n-gram 匹配的精确度，通常用于翻译评估。

对于 llm\_judge 和 mm\_llm\_judge，自定义数据集基准，需返回以下指标：

- a\_scores：前向和反向评估传递中 response\_A 的获胜次数。
- a\_scores\_stderr：成对判断中 response\_A\_scores 的标准误差。
- b\_scores：衡量前向和反向评估传递中 response\_B 的获胜次数。
- a\_scores\_stderr：成对判断中 response\_B\_scores 的标准误差。
- ties：response\_A 和 response\_B 被评估为相等的判断数量。
- ties\_stderr：成对判断中 ties 的标准误差。
- inference\_error：无法适当评估的判断数量。
- score：基于 response\_B 在前向和反向传递中的获胜次数得出的总分。
- score\_stderr：基于 response\_B 在前向和反向传递中的获胜次数得出的总分。
- inference\_error\_stderr：成对判断中总分的标准误差。
- winrate：使用 Bradley-Terry 概率模型计算的 response\_B 比 response\_A 更受欢迎的概率。
- lower\_rate：自动采样预估胜率的下限（第 2.5 百分位）。
- upper\_rate：自动采样预估胜率的上限（第 97.5 百分位）。

```
inference:
  max_new_tokens: 2048
  top_k: -1
  top_p: 1.0
  temperature: 0
  top_logprobs: 10
  reasoning_effort: null # options: low/high to enable reasoning or null to disable
  reasoning
```

- `max_new_tokens` : 要生成的最大词元数量。必须是整数。(不适用于 LLM Judge)
- `top_k` : 要考虑的最高概率词元的数量。必须是整数。
- `top_p` : 词元采样的累积概率阈值。必须是 1.0 到 0.0 之间的浮点数。
- `temperature` : 词元选择的随机性 (数值越高代表随机性越高), 保持为 0 以使结果具有确定性。浮点类型, 最小值为 0。
- `top_logprobs` : 推理响应中需返回的最高对数概率数量。取值必须为 0 到 20 之间的整数。对数概率包含消息内容中返回的输出词元及其对应的对数概率值。
- `reasoning_effort` : 控制具备推理能力模型的推理行为。仅当 `model_type` 指定了具备推理能力的模型 (当前为 `amazon.nova-2-lite-v1:0:256k`) 时, 才需设置 `reasoning_effort`。可用选项为 `null` (未设置时的默认值; 表示禁用推理)、`low` 或 `high`。

## 对数概率输出格式

在推理设置中配置 `top_logprobs` 后, 评估输出会在 `parquet` 文件中包含词元级别的对数概率。每个词元位置都会以字典形式, 返回排名靠前的候选词元及其对数概率, 结构如下:

```
{
  "Ġint": {"logprob_value": -17.8125, "decoded_value": " int"},
  "Ġthe": {"logprob_value": -2.345, "decoded_value": " the"}
}
```

每个词元条目包含:

- `logprob_value` : 词元的对数概率值
- `decoded_value` : 词元经解码后人类可读的字符串形式

原始分词器词元作为字典键以确保唯一性, `decoded_value` 则提供可读形式。

## 评估配方示例

Amazon Nova 提供四种不同类型的评估配方。所有配方均可在 [SageMaker HyperPod 配方 GitHub 存储库](#) 中找到。

### 通用文本基准测试配方

这些配方让您能够通过一套全面的纯文本基准测试来评估 Amazon Nova 模型的基本功能。

配方格式：`xxx_general_text_benchmark_eval.yaml`。

### 自带数据集基准测试配方

这些配方让您能够使用自己的数据集进行基准测试，并使用不同类型的指标将模型输出与参考答案进行对比。

配方格式：`xxx_bring_your_own_dataset_eval.yaml`。

### 自带数据集要求

文件格式：

- 包含评估示例的单个 `gen_qa.jsonl` 文件。文件名应为 `gen_qa.jsonl`。
- 您必须将数据集上传到 SageMaker AI 训练作业可以访问的 S3 位置。
- 该文件必须遵循常规问答数据集所需的架构格式。

架构格式要求：`.jsonl` 文件中的每一行都必须是包含以下字段的 JSON 对象。

- 必填字段。

`query`：包含需要答案的问题或说明的字符串。

`response`：包含预期模型输出的字符串。

- 可选字段。

`system`：包含系统提示的字符串，用于在处理查询之前设置 AI 模型的行为、角色或个性。

`images`：对象数组，每个对象包含 `data` 属性（Base64 编码的图像字符串）。

`metadata`：字符串类型，与本条记录关联的元数据，用于标签分类。

## 示例条目

```
{
  "system": "You are an English major with top marks in class who likes to give minimal
  word responses: ",
  "query": "What is the symbol that ends the sentence as a question",
  "response": "?"
}{
  "system": "You are a pattern analysis specialist who provides succinct answers: ",
  "query": "What is the next number in this series? 1, 2, 4, 8, 16, ?",
  "response": "32"
}{
  "system": "You have great attention to detail and follow instructions accurately: ",
  "query": "Repeat only the last two words of the following: I ate a hamburger today
  and it was kind of dry",
  "response": "of dry"
}{
  "system": "Image inference: ",
  "query": "What is the number in the image? Please just use one English word to
  answer.",
  "response": "two",
  "images": [
    {
      "data": "data:image/png;base64,iVBORw0KGgoA ..."
    }
  ]
}
```

要使用自定义数据集，请按以下必填字段修改评估配方，且不要更改现有配置：

```
evaluation:
  task: gen_qa
  strategy: gen_qa
  metric: all
```

## 限制

- 每次评估只支持一个 .jsonl 文件。
- 该文件必须严格遵循定义的架构。

## 自定义指标

您可以使用自定义指标，通过自定义预处理、后处理及指标能力，完全自定义模型评测流程。预处理可在将输入数据发送到推理服务器前对其进行处理；后处理支持自定义指标计算，并根据需求返回自定义指标。

请按如下步骤，通过自定义评估 SDK 接入自定义指标。

1. 若尚未操作，请先在 [AWS 账户 中创建 AWS Lambda 函数](#)。
2. 从 [GitHub 存储库](#) 下载预构建的 `nova-custom-eval-layer.zip` 文件。您可通过这款开源的 Nova 自定义评估 SDK，验证自定义函数的输入/输出有效载荷，并提供统一接口，以集成到 Nova 训练过程中的自定义指标评估流程。
3. 执行以下命令上传自定义 Lambda 层：

```
aws lambda publish-layer-version \  
  --layer-name nova-custom-eval-layer \  
  --zip-file fileb://nova-custom-eval-layer.zip \  
  --compatible-runtimes python3.12 python3.11 python3.10 python3.9
```

4. 将该层作为自定义层添加到 Lambda 函数，同时需搭配必备的 AWS 层：AWSLambdaPowerToolsPythonV3-python312-arm64（为 pydantic 依赖项所需）。
5. 参考提供的示例修改 Lambda 代码（按需调整）。该示例代码可创建用于 Nova 自定义评估的 Lambda 函数，包含模型评测的预处理、后处理步骤。

```
from nova_custom_evaluation_sdk.processors.decorators import preprocess,  
    postprocess  
from nova_custom_evaluation_sdk.lambda_handler import build_lambda_handler  
  
@preprocess  
def preprocessor(event: dict, context) -> dict:  
    data = event.get('data', {})  
    return {  
        "statusCode": 200,  
        "body": {  
            "system": data.get("system"),  
            "prompt": data.get("prompt", ""),  
            "gold": data.get("gold", "")  
        }  
    }  
  
@postprocess
```

```
def postprocessor(event: dict, context) -> dict:
    # data is already validated and extracted from event
    data = event.get('data', [])
    inference_output = data.get('inference_output', '')
    gold = data.get('gold', '')

    metrics = []
    inverted_accuracy = 0 if inference_output.lower() == gold.lower() else 1.0
    metrics.append({
        "metric": "inverted_accuracy_custom",
        "value": accuracy
    })

    # Add more metrics here

    return {
        "statusCode": 200,
        "body": metrics
    }

# Build Lambda handler
lambda_handler = build_lambda_handler(
    preprocessor=preprocessor,
    postprocessor=postprocessor
)
```

6. 授予 Lambda 访问评估作业的权限。确保为评估作业指定的执行角色包含调用该 Lambda 函数的策略。以下为策略示例。

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "LambdaAccess",
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction"
      ],
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:ExampleFunction",
      "Condition": {
```

```

        "StringLike": {
            "aws:PrincipalArn": "arn:aws:iam::111122223333:role/
service-role/AmazonSageMaker-ExecutionRole-ARN"
        }
    },
    {
        "Sid": "DenyNonAWSEventSourcesForLambda",
        "Effect": "Deny",
        "Action": [
            "lambda:InvokeFunction"
        ],
        "Resource": "arn:aws:lambda:us-
east-1:111122223333:function:ExampleFunction",
        "Condition": {
            "Null": {
                "lambda:EventSourceToken": false
            }
        }
    }
]
}

```

7. 核对 Lambda 有效载荷架构。下表列出了 Lambda 请求与响应的架构。您可通过 Nova 自定义评估 SDK 验证该架构。

|      | Lambda 请求有效载荷                                                                                                                                               | Lambda 响应有效载荷                                                                                                                                                                       |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 预处理器 | <pre> {   "process_type": "preprocess",   "data": {     "system": "You are a helpful assistant",     "prompt": "What is 2+2?",     "gold": "4"   } } </pre> | <pre> {   "statusCode": 200,   "body": {     "system": "You are a helpful assistant that can substitute * for addition",     "prompt": "What is 2*2?",     "gold": "4"   } } </pre> |

|      | Lambda 请求有效载荷                                                                                                                                      | Lambda 响应有效载荷                                                                                                                                                                                                                                   |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 后处理器 | <pre>{   "process_type": "postprocess",   "data": {     "prompt": "What is 2+2?",     "inference_output":     "2+2=4",     "gold": "4"   } }</pre> | <pre>{   "statusCode": 200,   "body": [     {"metric": "accuracy",     "value": 1.0},     {"metric": "f1_score",     "value": 1.0},     {"metric": "exact_match",     "value": 1},     {"metric": "length_ratio",     "value": 0.8}   ] }</pre> |

## 8. 修改配方文件。见下列。

```
processor:
  lambda_arn: arn:aws:lambda:us-east-1:111122223333:function:name
  lambda_type: "custom_metrics"
  preprocessing:
    enabled: true
  postprocessing:
    enabled: true
  aggregation: average
```

- `lambda-arn` : 用于处理预处理和后处理逻辑的 Lambda 函数的 Amazon 资源名称 ( ARN ) 。
- `lambda_type` : `custom_metrics` 或 `rft`。
- `preprocessing` : 是否启用自定义预处理操作。
- `postprocessing` : 是否启用自定义后处理操作。
- `aggregation` : 内置聚合函数 ( 有效值 : `min`、`max`、`average`、`sum` ) 。

## 限制

- 自定义指标仅适用于文本输入数据集。
- 不支持多模态输入数据集。
- 预处理步骤不会处理元数据字段。

## Nova LLM-as-a-Judge 基准测试配方

Nova LLM Judge 是一项模型评测特征，让您可以使用自定义数据集，将一个模型的响应质量与基准模型的响应质量进行比较。它接受包含提示、基准响应和挑战响应的数据集，然后使用 Nova Judge 模型基于 [Bradley-Terry](#) 概率模型通过成对比较提供胜率指标。配方格式：`xxx_llm_judge_eval.yaml`。

### Nova LLM 数据集要求

文件格式：

- 包含评估示例的单个 `llm_judge.jsonl` 文件。文件名应为 `llm_judge.jsonl`。
- 您必须将数据集上传到 SageMaker AI 训练作业可以访问的 S3 位置。
- 该文件必须遵循 `llm_judge` 数据集所需的架构格式。
- 输入数据集应确保所有记录的上下文长度小于 12k。

架构格式：`.jsonl` 文件中的每一行都必须是包含以下字段的 JSON 对象。

- 必填字段。

`prompt`：包含生成响应提示的字符串。

`response_A`：包含基准响应的字符串。

`response_B`：包含与基准响应进行比较的替代响应的字符串。

### 示例条目

```
{
  "prompt": "What is the most effective way to combat climate change?",
  "response_A": "The most effective way to combat climate change is through a combination of transitioning to renewable energy sources and implementing strict carbon pricing policies. This creates economic incentives for businesses to reduce emissions while promoting clean energy adoption.",
  "response_B": "We should focus on renewable energy. Solar and wind power are good. People should drive electric cars. Companies need to pollute less."
}
{
  "prompt": "Explain how a computer's CPU works",
```

```

"response_A": "CPU is like brain of computer. It does math and makes computer work
fast. Has lots of tiny parts inside.",
"response_B": "A CPU (Central Processing Unit) functions through a fetch-execute
cycle, where instructions are retrieved from memory, decoded, and executed through its
arithmetic logic unit (ALU). It coordinates with cache memory and registers to process
data efficiently using binary operations."
}
{
"prompt": "How does photosynthesis work?",
"response_A": "Plants do photosynthesis to make food. They use sunlight and water. It
happens in leaves.",
"response_B": "Photosynthesis is a complex biochemical process where plants convert
light energy into chemical energy. They utilize chlorophyll to absorb sunlight,
combining CO2 and water to produce glucose and oxygen through a series of chemical
reactions in chloroplasts."
}

```

要使用您的自定义数据集，请按以下必填字段修改您的评估配方，且不要更改其中任何内容：

```

evaluation:
  task: llm_judge
  strategy: judge
  metric: all

```

## 限制

- 每次评估只支持一个 .jsonl 文件。
- 该文件必须严格遵循定义的架构。
- Nova Judge 模型对于 micro/lite/pro 规格是一样的。
- 目前不支持自定义评估模型。

使用 Nova LLM-as-a-judge，完成多模态（图像）基准测试配方

Nova LLM Judge 多模态（图像）版（简称 Nova MM\_LLM Judge）是一项模型评测功能，支持使用自定义数据集，将目标模型的响应质量与基准模型的响应进行对比评估。该功能接受包含提示、基准响应和挑战响应的数据集，以及 Base64 编码字符串图像，然后使用 Nova Judge 模型基于 [Bradley-Terry](#) 概率模型通过成对比较提供胜率指标。配方格式：`xxx_mm_llm_judge_eval.yaml`。

Nova LLM 数据集要求

文件格式：

- 包含评估示例的单个 `mm_llm_judge.jsonl` 文件。文件名称必须严格为 `llm_judge.jsonl`。
- 您必须将数据集上传到 SageMaker AI 训练作业可以访问的 S3 位置。
- 该文件必须遵循 `mm_llm_judge` 数据集所需的架构格式。
- 输入数据集应确保：排除 `image` 属性后，所有记录的上下文长度均小于 12k。

架构格式：`.jsonl` 文件中的每一行都必须是包含以下字段的 JSON 对象。

- 必填字段。

`prompt`：包含生成响应提示的字符串。

`images`：对象数组，每个对象包含 `data` 属性（值为 Base64 编码的图像字符串）。

`response_A`：包含基准响应的字符串。

`response_B`：包含与基准响应进行比较的替代响应的字符串。

## 示例条目

为便于阅读，以下示例包含换行与缩进，但在实际数据集中，每条记录必须单独占一行。

```
{
  "prompt": "What is in the image?",
  "images": [
    {
      "data": "data:image/jpeg;base64,/9j/2wBDAAQDAwQDAwQEAwQFBAQFBgo..."
    }
  ],
  "response_A": "a dog.",
  "response_B": "a cat.",
}
{
  "prompt": "How many animals are in each of the images?",
  "images": [
    {
      "data": "data:image/jpeg;base64,/9j/2wBDAAQDAwQDAwQEAwQFBAQFBgo..."
    },
    {
      "data": "data:image/jpeg;base64,/DKEafe3gihn..."
    }
  ],
}
```

```
"response_A": "The first image contains one cat and the second image contains one dog",  
"response_B": "The first image has one animal and the second has one animal"  
}
```

要使用您的自定义数据集，请按以下必填字段修改您的评估配方，且不要更改其中任何内容：

```
evaluation:  
  task: mm_llm_judge  
  strategy: judge  
  metric: all
```

## 限制

- 每次评估只支持一个 .jsonl 文件。
- 该文件必须严格遵循定义的架构。
- Nova MM Judge 模型仅支持图像参考。
- Nova MM Judge 模型在 Amazon Nova Micro、Amazon Nova Lite、Amazon Nova Pro 各规格中保持一致。
- 目前不支持自定义评估模型。
- 不支持 Amazon S3 图像 URI。
- 输入数据集应确保：排除 image 属性后，所有记录的上下文长度均小于 12k。

## 评测持续预训练 ( CPT ) 检查点

对持续预训练 ( CPT ) 模型的评测可能比经过监督式微调 ( SFT ) 的模型更具挑战性，因为 CPT 模型通常缺乏遵循指令的能力。CPT 模型不会遵循指令，而是作为补全模型运行，这意味着它们只会尝试延续输入词元序列所提供的模式。鉴于这一限制，典型的评测数据集可能无法正常工作，因为输入是“问答”格式，而模型不会回答问题，只会尝试沿着同一个问题继续生成。不过，通过以特定方式格式化数据集，以补全风格提示模型，即可了解模型的性能表现。

请按照以下步骤，了解如何使用 Nova Forge 评测工作流对持续预训练模型进行评测。

## 数据集准备与格式化

评测 CPT 模型可以利用 Nova Forge 模型评测体验中已有的[自带数据集](#)工作流。不过，数据集中的查询必须格式化为纯粹的“补全”格式，因为 CPT 模型不会像 SFT 模型那样对标准的提问式提示做出响应。

另一个常见的限制是，仅经过 CPT 的模型通常无法生成 STOP 或序列结束词元，这意味着模型会持续生成词元，直到被强制停止（例如通过 `max_new_tokens` 参数）。鉴于这一限制，最佳实践是使用单词元响应（如单选题）来评测模型，确保模型不会在提示后继续生成不需要的垃圾输出。

例如，典型的评测数据集（如 MMLU、GPQA、MATH 等）可能会用如下问题提示模型：

```
Early settlements and high population density along coastlines and rivers are
best attributed to which of the following?
A: "Poor climate conditions"
B: "Limited forest cover"
C: "Cars"
D: "Access to trade routes"

(Expected answer is D.)
```

不过，由于 CPT 模型未经过指令遵循微调，因此无法正确回答该问题。因此，CPT 模型必须以补全风格进行提示，例如：

```
Early settlements and high population density along coastlines and rivers
are best attributed to which of the following?
A: Poor climate conditions
B: Limited forest cover
C: Cars
D: Access to trade routes
The correct answer to this question is option

(Expected answer is D.)
```

在推理后检查模型生成的输出对数概率，可获得有关模型是否正确处理输入并生成正确响应的详细信息。不能保证模型始终生成预期的确切响应（本例中为字母 D），但如果模型功能正常，该响应应该出现在对数概率中。

另一个非单选题的补全式提示示例：

```
The capital of France is

(Expected answer of Paris)
```

我们期望模型生成“巴黎”作为响应，或者在对数概率输出的某处看到对应于“巴黎”的词元。

## 数据集格式化

CPT 评测利用现有的 [自带数据集](#) 工作流。数据必须以“查询-响应”格式进行格式化，存放在由换行分隔的 JSONL 文件中。

包含 4 条数据的数据集示例：

```
{"query": "The capital of France is", "response": "Paris"}
{"query": "2 + 2 =", "response": "4"}
{"query": "The mitochondria is the powerhouse of the", "response": "cell"}
{"query": "What is the largest planet?\nA: Mars\nB: Jupiter\nC: Saturn\nD: Earth\n\nAnswer:", "response": "B"}
```

每行必须包含：

- query：用于补全的提示文本
- response：预期的补全（真实值）

模型将接收原始文本输入，不带聊天格式。CPT 模型通常尚未针对特殊词元进行训练，无法正确适配聊天模板。因此在对模型进行提示时，仅会将查询中提供的字符串发送给模型（系统会自动在其前方添加一个 [BOS] 词元）。

## 配方配置

以下是配置用于评测 CPT 模型的配方示例：

```
run:
  name: cpt_eval_job
  model_type: amazon.nova-2-lite-v1:0:256k
  model_name_or_path: s3://bucket/path/to/cpt-checkpoint/

evaluation:
  task: gen_qa # Required for CPT - bring your own dataset
  strategy: gen_qa
  metric: all # Returns rouge1, rouge2, rougeL, exact_match, f1_score, bleu

inference:
  checkpoint_is_instruction_tuned: "false" # Required for CPT checkpoints
  top_logprobs: 5 # Change to desired amount of logprobs to calculate
  max_new_tokens: 1 # Keep low for completion tasks
  temperature: 0.0
```

## CPT 评测的关键调整：

- `checkpoint_is_instruction_tuned`: "false"

这是一个新增参数，专门用于支持对 CPT 检查点进行评测。将 `checkpoint_is_instruction_tuned` 设置为 `false` 将禁用默认的 Amazon Nova 聊天模板（该模板通常会包装输入提示）。

- `top_logprobs`: 5

对数概率揭示了模型在可能的下一个词元上的置信度分布，帮助您评测模型在预训练期间是否学会了预期的补全内容。通常，如果模型按预期运行，我们应该看到预期的响应（例如“A”、“B”等）作为生成的输出词元或者对数概率中的一个词元。

- `max_new_tokens`: 1

CPT 模型通常尚未经过相关训练，不会生成特殊的“停止”或“序列结束”词元以指示何时停止推理。这意味着模型通常会持续生成新词元，直至达到设定的最大词元长度，从而产生不必要的推理开销。通常，将 `max_new_tokens` 限制为 1 并采用可通过单次响应（如单选题）评测模型的提示，是最高效的提示方式。将 `max_new_tokens` 设置为 1，可确保不会生成额外的垃圾词元。

## 关键参数

- `checkpoint_is_instruction_tuned`：必须设置为 "false"（或设置为布尔值 `false`）以禁用聊天模板
- `top_logprobs`：建议设为 5，用于观察模型在 CPT 过程中的学习状态
- `task`：必须为 `gen_qa`，CPT 模型无法使用 MMLU、MATH 这类指令遵循评测基准
- `strategy`：必须为 `gen_qa`
- `max_new_tokens`：建议保持较低值（1 – 5），因为 CPT 模型专注于补全任务，而非开放生成

## 运行评估训练作业

使用以下示例笔记本启动训练作业。请参考下方的笔记本示例，运行评估训练作业。有关更多信息，请参阅[使用 SageMaker AI 估算器运行训练作业](#)。

## 参考表

在运行笔记本之前，请参阅以下参考表来选择映像 URI 和实例配置。

## 选择映像 URI

| 指南       | 映像 URI                                                                                     |
|----------|--------------------------------------------------------------------------------------------|
| 评估映像 URI | 708977205387.dkr.ecr.us-east-1.amazonaws.com/<br>nova-evaluation-repo:SM-TJ-Eval-V2-latest |

## 选择实例类型和计数

| 模型                | 作业类型            | 实例类型        | 推荐的实例计数 | 支持的实例计数 |
|-------------------|-----------------|-------------|---------|---------|
| Amazon Nova Micro | 评估 ( SFT/DP O ) | g5.12xlarge | 1       | 1–16    |
| Amazon Nova Lite  | 评估 ( SFT/DP O ) | g5.12xlarge | 1       | 1–16    |
| Amazon Nova Pro   | 评估 ( SFT/DP O ) | p5.48xlarge | 1       | 1–16    |

## 示例笔记本

以下示例笔记本演示了如何运行评估训练作业。

```
# install python SDK

# Do not use sagemaker v3, as sagemaker v3 introduced breaking changes

!pip install sagemaker==2.254.1

import os
import sagemaker,boto3
from sagemaker.inputs import TrainingInput
from sagemaker.pytorch import PyTorch

sagemaker_session = sagemaker.Session()
role = sagemaker.get_execution_role()

# Download recipe from https://github.com/aws/sagemaker-hyperpod-recipes/tree/main/
# recipes_collection/recipes/evaluation/nova to local
# Assume the file name be `recipe.yaml`
```

```
# Populate parameters
# input_s3_uri = "s3://<path>/input/" # (Optional) Only used for multi-modal dataset or
bring your own dataset s3 location
output_s3_uri= "s3://<path>/output/" # Output data s3 location, a zip containing
metrics json and tensorboard metrics files will be stored to this location
instance_type = "instance_type" # ml.g5.16xlarge as example
instance_count = 1 # The number of instances for inference (set instance_count > 1 for
multi-node inference to accelerate evaluation)
job_name = "your job name"
recipe_path = "recipe_path" # ./recipe.yaml as example
image_uri = "708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-evaluation-repo:SM-TJ-
Eval-V2-latest" # Do not change
output_kms_key = "<KMS key arn to encrypt trained model in Amazon-owned S3 bucket>" #
optional, leave blank for Amazon managed encryption

# (Optional) To bring your own dataset and LLM judge for evaluation
# evalInput = TrainingInput(
# s3_data=input_s3_uri,
# distribution='FullyReplicated',
# s3_data_type='S3Prefix'
#)

estimator = PyTorch(
    output_path=output_s3_uri,
    base_job_name=job_name,
    role=role,
    instance_type=instance_type,
    instance_count=instance_count,
    training_recipe=recipe_path,
    sagemaker_session=sagemaker_session,
    image_uri=image_uri,
    output_kms_key=output_kms_key
)
estimator.fit()

# If input dataset exist, pass in inputs
# estimator.fit(inputs={"train": evalInput})
```

## 评测和分析评估结果

成功完成评估作业后，您可以使用以下步骤评测和分析结果。

要评测和分析结果，请执行以下步骤：

1. 了解输出位置结构。结果以压缩文件的形式存储在您指定的 Amazon S3 输出位置：

```
s3://your-bucket/output/benchmark-name/
### job_name/
    ### output/
        ### output.tar.gz
```

2. 从存储桶下载 output.tar.gz 文件。解压缩要显示的内容。

```
run_name/
### eval_results/
|   ### results_[timestamp].json
#   ### inference_output.jsonl (only present for gen_qa)
|   ### details/
|       ### model/
|           ### <execution-date-time>/
|               ###details_<task_name>_#_<datetime>.parquet
### tensorboard_results/
    ### eval/
        ### events.out.tfevents.[timestamp]
```

- results\_[timestamp].json：输出指标 JSON 文件
- details\_<task\_name>\_#\_<datetime>.parquet：推理输出文件（strong\_reject 除外）
- events.out.tfevents.[timestamp]：TensorBoard 输出文件
- inference\_output.jsonl：已清理推理输出文件（仅适用于 gen\_qa 任务）

3. 在 TensorBoard 中查看结果。要可视化您的评估指标，请执行以下操作：

1. 将解压缩的文件夹上传至 S3 存储桶
2. 导航到 SageMaker AI TensorBoard
3. 选择您的“S3 文件夹”。
4. 添加 S3 文件夹路径
5. 等待同步完成

4. 分析推理输出。除 llm\_judge 和 strong\_reject 之外，所有评估任务在推理输出中都将包含以下用于分析的字段。

- `full_prompt` : 发送给模型用于评估任务的完整用户提示。
- `gold` : 包含数据集指定的正确答案的字段。
- `metrics` : 包含根据单个推理评估的指标的字段。需要聚合的值对于单个推理输出没有值。
- `predictions` : 包含模型针对给定提示的输出列表的字段。
- `pred_logits` : 该字段包含消息内容中返回的输出词元及其对应的对数概率值。

通过查看这些字段，您可以确定指标差异的原因并了解自定义模型的行为。

对于 `llm_judge`，推理输出文件针对每对评估会在指标字段下包含以下字段。

- `forward_output` : 按顺序评估时 Judge 的原始偏好 ( `response_A`、`response_B` )。
- `backward_output` : 按相反顺序进行评估时 Judge 的原始偏好 ( `response_B`、`response_A` )。
- `Pairwise metrics` : 按每对前向和反向评估计算的指标包括 `a_scores`、`b_scores`、`ties`、`inference-score` 和 `score`。

#### Note

诸如 `winrate` 的聚合指标仅在摘要结果文件中提供，不会针对每一判断都提供。

对于 `gen_qa`，`inference_output.jsonl` 文件针对每个 JSON 对象均包含以下字段：

- `prompt` : 提交给模型的最终提示
- `推理` : 模型的原始推理输出
- `gold` : 来自输入数据集的目标响应
- `metadata` : 来自输入数据集的元数据字符串 ( 如果提供 )

## 评估最佳实践和疑难解答

### 最佳实践

下面列出了评估过程的一些最佳实践。

- 按模型和基准测试类型整理输出路径。

- 保持一致的命名约定以便于跟踪。
- 将解压缩的结果保存在安全的位置。
- 监控 TensorBoard 同步状态以成功加载数据。

## 问题排查

您可以使用 CloudWatch 日志组 `/aws/sagemaker/TrainingJobs` 来记录训练作业错误日志。

### 引擎核心故障

问题：

如遇以下报错：

```
RuntimeError: Engine core initialization failed.
```

原因：

虽然这是一个可能由多种原因引发的一般性错误，但最常见原因是：您尝试加载的模型检查点与指定的模型类型 ( `model_type` ) 不匹配。例如，您想要评估一个微调后的 Nova 2.0 lite 模型检查点，但指定的模型类型却是 1.0 版本 ( 如 `amazon.nova-micro-v1:0:128k` )

正确的映射示例：

```
model_type: amazon.nova-2-lite-v1:0:256k
model_name_or_path: nova-lite-2/prod # or s3://escrow_bucket/model_location
```

预防措施：

提交评估作业前，务必核对 `model_name_or_path` 与 `model_type` 是否正确匹配。

## 可用子任务

下面列出了跨多个领域进行模型评测的可用子任务，包括 MMLU ( 大规模多任务语言理解 )、BBH ( 高难度推理基准 )、MATH ( 数学推理 ) 和 MMMU ( 大规模多学科多模态理解 )。这些子任务让您能够评测模型在特定功能和知识领域的表现。

### MMLU

```
MMLU_SUBTASKS = [
    "abstract_algebra",
    "anatomy",
```

```
"astronomy",
"business_ethics",
"clinical_knowledge",
"college_biology",
"college_chemistry",
"college_computer_science",
"college_mathematics",
"college_medicine",
"college_physics",
"computer_security",
"conceptual_physics",
"econometrics",
"electrical_engineering",
"elementary_mathematics",
"formal_logic",
"global_facts",
"high_school_biology",
"high_school_chemistry",
"high_school_computer_science",
"high_school_european_history",
"high_school_geography",
"high_school_government_and_politics",
"high_school_macroeconomics",
"high_school_mathematics",
"high_school_microeconomics",
"high_school_physics",
"high_school_psychology",
"high_school_statistics",
"high_school_us_history",
"high_school_world_history",
"human_aging",
"human_sexuality",
"international_law",
"jurisprudence",
"logical_fallacies",
"machine_learning",
"management",
"marketing",
"medical_genetics",
"miscellaneous",
"moral_disputes",
"moral_scenarios",
"nutrition",
"philosophy",
```

```
"prehistory",
"professional_accounting",
"professional_law",
"professional_medicine",
"professional_psychology",
"public_relations",
"security_studies",
"sociology",
"us_foreign_policy",
"virology",
"world_religions"
]
```

## BBH

```
BBH_SUBTASKS = [
    "boolean_expressions",
    "causal_judgement",
    "date_understanding",
    "disambiguation_qa",
    "dyck_languages",
    "formal_fallacies",
    "geometric_shapes",
    "hyperbaton",
    "logical_deduction_five_objects",
    "logical_deduction_seven_objects",
    "logical_deduction_three_objects",
    "movie_recommendation",
    "multistep_arithmetic_two",
    "navigate",
    "object_counting",
    "penguins_in_a_table",
    "reasoning_about_colored_objects",
    "ruin_names",
    "salient_translation_error_detection",
    "snarks",
    "sports_understanding",
    "temporal_sequences",
    "tracking_shuffled_objects_five_objects",
    "tracking_shuffled_objects_seven_objects",
    "tracking_shuffled_objects_three_objects",
    "web_of_lies",
    "word_sorting"
```

```
]
```

## 数学

```
MATH_SUBTASKS = [  
    "algebra",  
    "counting_and_probability",  
    "geometry",  
    "intermediate_algebra",  
    "number_theory",  
    "prealgebra",  
    "precalculus",
```

## MMMU

```
        MATH_SUBTASKS = [  
"Accounting",  
"Agriculture",  
"Architecture_and_Engineering",  
"Art",  
"Art_Theory",  
"Basic_Medical_Science",  
"Biology",  
"Chemistry",  
"Clinical_Medicine",  
"Computer_Science",  
"Design",  
"Diagnostics_and_Laboratory_Medicine",  
"Economics",  
"Electronics",  
"Energy_and_Power",  
"Finance",  
"Geography",  
"History",  
"Literature",  
"Manage",  
"Marketing",  
"Materials",  
"Math",  
"Mechanical_Engineering",  
"Music",  
"Pharmacy",
```

```
"Physics",
"Psychology",
"Public_Health",
"Sociology",
```

您可通过多种评估方法与指标，对自定义 Nova 模型进行评估。

本节内容：

- [推理模型评测](#)
- [RFT 评测](#)
- [实现奖励函数](#)

## 推理模型评测

### 概述

推理模型支持功能可借助具备推理能力的 Nova 模型开展评估，这类模型会在生成最终响应前执行显式的内部推理。该功能通过 `reasoning_effort` 参数实现 API 级别的控制，可动态启用/禁用推理功能，有助于提升复杂分析类任务的响应质量。

### 支持的模型

- `amazon.nova-2-lite-v1:0:256k`

### 配方配置

在配方的 `inference` 模块中添加 `reasoning_effort` 参数，即可启用推理功能：

```
run:
  name: reasoning-eval-job-name                # [MODIFIABLE] Unique
  identifier for your evaluation job
  model_type: amazon.nova-2-lite-v1:0:256k     # [FIXED] Must be a reasoning-
supported model
  model_name_or_path: nova-lite-2/prod         # [FIXED] Path to model
  checkpoint or identifier
  replicas: 1                                  # [MODIFIABLE] Number of
  replicas for SageMaker Training job
```

```

data_s3_path: "" # [MODIFIABLE] Leave empty for
SageMaker Training job; optional for SageMaker HyperPod job
output_s3_path: "" # [MODIFIABLE] Output path for
SageMaker HyperPod job (not compatible with SageMaker Training jobs)

evaluation:
  task: mmlu # [MODIFIABLE] Evaluation task

  strategy: zs_cot # [MODIFIABLE] Evaluation
strategy
  metric: accuracy # [MODIFIABLE] Metric
calculation method

inference:
  reasoning_effort: high # [MODIFIABLE] Enables
reasoning mode; options: low/high or null to disable
  max_new_tokens: 32768 # [MODIFIABLE] Maximum tokens
to generate, recommended value when reasoning_effort set to high
  top_k: -1 # [MODIFIABLE] Top-k sampling
parameter
  top_p: 1.0 # [MODIFIABLE] Nucleus
sampling parameter
  temperature: 0 # [MODIFIABLE] Sampling
temperature (0 = deterministic)

```

## reasoning\_effort 参数使用说明

reasoning\_effort 参数用于控制具备推理能力模型的推理行为。

### 先决条件

- **模型兼容性**：仅当 model\_type 指定了具备推理能力的模型（当前为 amazon.nova-2-lite-v1:0:256k）时，才需设置 reasoning\_effort
- **错误处理**：对不受支持的模型使用 reasoning\_effort 将会失败，并返回 ConfigValidationError: "Reasoning mode is enabled but model '{model\_type}' does not support reasoning. Please use a reasoning-capable model or disable reasoning mode." 错误

## 可用选项

| Option    | 行为        | 词元限制           | 使用案例                |
|-----------|-----------|----------------|---------------------|
| null (默认) | 禁用推理模式    | 不适用            | 无需推理开销的标准评估         |
| low       | 启用推理并施加约束 | 内部推理占用 4000 词元 | 需要简洁推理的场景；兼顾速度与成本优化 |
| high      | 启用推理且无约束  | 内部推理无词元上限      | 需要深度分析与分步推理的复杂问题    |

| 训练模型        | 可用选项           | 配置方法                                                               |
|-------------|----------------|--------------------------------------------------------------------|
| 监督式微调 (SFT) | 仅限 High 或 Off  | 使用 reasoning_enabled: true (high) 或 reasoning_enabled: false (off) |
| 强化微调 (RFT)  | Low、High 或 Off | 使用 reasoning_effort: low 或 reasoning_effort: high。省略字段以禁用。         |
| 评估          | Low、High 或 Off | 使用 reasoning_effort: low 或 reasoning_effort: high。设为 null 即可禁用。    |

## 何时启用推理

满足以下场景时，建议使用推理模式 (**low** 或 **high**)

- 复杂问题求解任务 (数学、逻辑谜题、编程)
- 需要中间推理过程的多步分析类问题
- 详细推导或分步思考可提升准确率的任务
- 优先保证响应质量而非速度的场景

满足以下场景时，建议使用非推理模式 (**null** 或不设置该参数)

- 简单问答或事实查询

- 创意写作类任务
- 对响应速度要求极高的场景
- 需要排除推理开销的性能基准测试
- 推理无法提升效果、需优化成本的场景

## 问题排查

错误：“Reasoning mode is enabled but model does not support reasoning”

原因：reasoning\_effort 参数被设为非空值，但指定的 model\_type 支持推理。

解决方法：

- 确认模型类型为 amazon.nova-2-lite-v1:0:256k
- 若使用其他模型，或切换为具备推理能力的模型，或从配方中移除 reasoning\_effort 参数

## RFT 评测

什么是 RFT 评测？

RFT 评测支持您在强化学习微调的训练前、训练中及训练后，通过自定义奖励函数评测模型性能。与采用预定义指标的标准评测不同，RFT 评测支持通过 Lambda 函数自定义成功标准，并依据您的具体需求对模型输出进行评分。

为什么要使用 RFT 进行评测？

评测对于判断强化微调过程是否实现以下目标至关重要：

- 提升模型与您特定使用案例及人类价值偏好的对齐程度
- 维持或提升模型在核心任务上的能力
- 避免产生非预期副作用，例如事实准确性下降、内容冗余度增加或其他任务性能退化
- 满足由奖励函数定义的自定义成功标准

何时使用 RFT 评测

在以下场景中使用 RFT 评测：

- RFT 训练之前：在评测数据集上建立基线指标
- RFT 训练期间：使用中间检查点监控训练进度
- RFT 训练之后：验证最终模型是否符合您的要求
- 模型对比：采用一致的奖励标准对多个模型版本进行评测

### Note

如需领域专属的自定义评估指标，请使用 RFT 评测。对于通用评测（准确率、困惑度、BLEU），请使用标准评测方法。

## 数据格式要求

### 输入数据结构

RFT 评测输入数据必须遵循 OpenAI 强化微调格式。每个样本都是一个 JSON 对象，其中包含：

- `messages`：包含 `system` 和 `user` 角色的对话轮次数组
- `reference_answer`：奖励函数用于评分的预期输出或真实值数据

### 数据格式示例

```
{
  "messages": [
    {
      "role": "user",
      "content": [
        {
          "type": "text",
          "text": "Solve for x. Return only JSON like {\"x\": <number>}. Equation: 2x + 5 = 13"
        }
      ]
    }
  ],
  "reference_answer": {
    "x": 4
  }
}
```

## 目前的局限性

- 纯文本模式：不支持多模态输入（图像、音频、视频）
- 单轮对话：仅支持单条用户消息（不支持多轮对话）
- JSON 格式：输入数据必须为 JSONL 格式（每行一个 JSON 对象）
- 模型输出：根据指定模型生成的补全进行评测

## 准备评测配方

### 示例笔记本

有关完整示例，请参阅[评测笔记本](#)。

### 示例配方配置

```
run:
  name: nova-lite-rft-eval-job
  model_type: amazon.nova-lite-v1:0:300k
  model_name_or_path: s3://escrow_bucket/model_location # [MODIFIABLE] S3 path to your
  model or model identifier
  replicas: 1 # [MODIFIABLE] For SageMaker Training jobs only; fixed for HyperPod jobs

  data_s3_path: "" # [REQUIRED FOR HYPERPOD] Leave empty for SageMaker Training jobs
  and use TrainingInput in sagemaker python SDK
  output_s3_path: "" # [REQUIRED] Output artifact S3 path for evaluation results

evaluation:
  task: rft_eval # [FIXED] Do not modify
  strategy: rft_eval # [FIXED] Do not modify
  metric: all # [FIXED] Do not modify

# Inference Configuration
inference:
  max_new_tokens: 8192 # [MODIFIABLE] Maximum tokens to generate
  top_k: -1 # [MODIFIABLE] Top-k sampling parameter
  top_p: 1.0 # [MODIFIABLE] Nucleus sampling parameter
  temperature: 0 # [MODIFIABLE] Sampling temperature (0 = deterministic)
  top_logprobs: 0 # [MODIFIABLE] Set between 1-20 to enable logprobs output

# =====
# Bring Your Own Reinforcement Learning Environment
# =====
```

```
rl_env:
  reward_lambda_arn: arn:aws:lambda:<region>:<account_id>:function:<reward-function-name>
```

## 预设奖励函数

两个预设的奖励函数 ( `prime_code` 和 `prime_math` ) 可用作 Lambda 层，以便与 RFT Lambda 函数集成。

### 概述

这些预设函数提供开箱即用的评测能力，适用于：

- `prime_code`：代码生成和正确性评测
- `prime_math`：数学推理和问题求解评测

### 快速设置

1. 从 [nova-custom-eval-sdk 版本](#) 页面下载 Lambda 层。
2. 使用 AWS 命令行界面 ( AWS CLI ) 发布 Lambda 层：

```
aws lambda publish-layer-version \
  --layer-name preset-function-layer \
  --description "Preset reward function layer with dependencies" \
  --zip-file fileb://universal_reward_layer.zip \
  --compatible-runtimes python3.9 python3.10 python3.11 python3.12 \
  --compatible-architectures x86_64 arm64
```

3. 在 AWS 管理控制台中将该层添加到 Lambda 函数 ( 从自定义层中选择 `preset-function-layer`，并添加 `AWSSDKPandas-Python312` 以获取 `numpy` 依赖项 )。
4. 在 Lambda 代码中导入并使用：

```
from prime_code import compute_score # For code evaluation
from prime_math import compute_score # For math evaluation
```

## `prime_code` 函数

通过针对测试用例执行代码并衡量正确性来评测 Python 代码生成任务。

### 输入数据集格式示例

```

{"messages":[{"role":"user","content":"Write a function that returns the sum of
two numbers."}], "reference_answer":{"inputs":["3\n5", "10\n-2", "0\n0"], "outputs":
["8", "8", "0"]}}
{"messages":[{"role":"user","content":"Write a function to check if a number
is even."}], "reference_answer":{"inputs":["4", "7", "0", "-2"], "outputs":
["True", "False", "True", "True"]}}

```

## 主要特征

- 从 Markdown 代码块中自动提取代码
- 函数检测与基于调用的测试
- 带超时保护的测试用例执行
- 语法验证与编译检查
- 包含回溯信息的详细错误报告

## prime\_math 函数

依托符号计算能力，评测模型的数学推理与问题求解水平。

## 输入格式

```

{"messages":[{"role":"user","content":"What is the derivative of x^2 +
3x?."}], "reference_answer":"2*x + 3"}

```

## 主要特征

- 使用 SymPy 进行符号计算评测
- 支持多种答案格式 ( LaTeX、纯文本、符号 )
- 数学等价性检查
- 表达式归一化与化简

## 数据格式要求

### 评测代码

- 输入：函数参数数组 ( 类型正确：整数、字符串等 )
- 输出：预期返回值数组 ( 类型正确：布尔值、数字等 )

- 代码：必须使用 Python 编写，并包含清晰的函数定义

## 数学评测

- 参考答案：数学表达式或数值
- 响应：可为 LaTeX、纯文本或符号表达式
- 等效：通过符号方式检查，而不仅仅是字符串匹配

## 最佳实践

- 在测试用例中使用正确的数据类型（整数与字符串、布尔值与 True）
- 在代码问题中提供清晰的函数签名
- 在测试输入中包括边缘情况（零、负数、空输入）
- 在参考答案中统一设置数学表达式的格式
- 在部署前使用样本数据测试奖励函数

## 错误处理

这两个函数都包含针对以下情况的强大错误处理能力：

- 生成的代码中存在编译错误
- 执行过程中出现运行时异常
- 输入数据格式错误
- 超时场景无限循环
- 数学表达式无效

## 创建奖励函数

### Lambda ARN 要求

Lambda ARN 必须采用以下格式：

```
"arn:aws:lambda:*:*:function:*SageMaker*"
```

如果 Lambda 不遵循此命名方案，作业将失败并显示以下错误：

```
[ERROR] Unexpected error: lambda_arn must contain one of: ['SageMaker', 'sagemaker', 'Sagemaker'] when running on SMHP platform (Key: lambda_arn)
```

## Lambda 请求格式

Lambda 函数接收以下格式的数据：

```
[
  {
    "id": "sample-001",
    "messages": [
      {
        "role": "user",
        "content": [
          {
            "type": "text",
            "text": "Do you have a dedicated security team?"
          }
        ]
      },
      {
        "role": "nova_assistant",
        "content": [
          {
            "type": "text",
            "text": "As an AI developed by Company, I don't have a dedicated security
team..."
          }
        ]
      }
    ],
    "reference_answer": {
      "compliant": "No",
      "explanation": "As an AI developed by Company, I do not have a traditional
security team..."
    }
  }
]
```

**Note**

消息结构包含嵌套的 `content` 数组，与输入数据格式相匹配。角色为 `nova_assistant` 的最后一条消息包含模型生成的响应。

## Lambda 响应格式

Lambda 函数必须按以下格式返回数据：

```
[
  {
    "id": "sample-001",
    "aggregate_reward_score": 0.75,
    "metrics_list": [
      {
        "name": "accuracy",
        "value": 0.85,
        "type": "Metric"
      },
      {
        "name": "fluency",
        "value": 0.90,
        "type": "Reward"
      }
    ]
  }
]
```

## 响应字段

- `id`：必须与输入样本 ID 匹配
- `aggregate_reward_score`：总体得分（通常为 0.0 到 1.0）
- `metrics_list`：各项指标组成的数组，包含以下字段：
  - `name`：指标标识符（例如 `accuracy`、`fluency`）
  - `value`：指标得分（通常为 0.0 到 1.0）
  - `type`：Metric（用于报告）或 Reward（用于训练）

## IAM 权限

### 所需的权限

SageMaker 执行角色必须具备调用 Lambda 函数的权限。将此策略添加到 SageMaker 执行角色：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction"
      ],
      "Resource": "arn:aws:lambda:region:account-id:function:function-name"
    }
  ]
}
```

### Lambda 执行角色

Lambda 函数的执行角色需要基本的 Lambda 执行权限：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

如果 Lambda 函数访问其他 AWS 服务（例如用于参考数据的 S3、用于日志记录的 DynamoDB），请将这些权限添加到 Lambda 执行角色。

## 执行评测作业

1. 准备数据 – 根据数据格式要求格式化评测数据，并将 JSONL 文件上传到 S3：s3://your-bucket/eval-data/eval\_data.jsonl
2. 配置配方 – 使用您的配置更新示例配方：
  - 将 model\_name\_or\_path 设置为模型路径
  - 将 lambda\_arn 设置为奖励函数 ARN
  - 将 output\_s3\_path 设置为目标输出位置
  - 根据需要调整 inference 参数

将配方另存为 rft\_eval\_recipe.yaml

3. 运行评测 – 使用提供的笔记本执行评测工作：[评测笔记本](#)
4. 监控进度 – 通过以下方式监控评测作业：
  - SageMaker 控制台：检查作业状态与日志
  - CloudWatch Logs：查看详细执行日志
  - Lambda 日志：调试奖励函数相关问题

## 了解评测结果

### 输出格式

评测作业会以 JSONL 格式将结果输出到指定的 S3 位置。每行包含一个样本的评测结果：

```
{
  "id": "sample-001",
  "aggregate_reward_score": 0.75,
  "metrics_list": [
    {
      "name": "accuracy",
      "value": 0.85,
      "type": "Metric"
    },
    {
      "name": "fluency",
      "value": 0.90,
      "type": "Reward"
    }
  ]
}
```

}

**Note**

RFT 评测作业输出格式与 Lambda 响应格式完全一致。评测服务会原样透传 Lambda 函数的响应，确保奖励计算结果与最终输出保持一致。

## 解析结果

### 奖励总分

- 取值范围：通常为 0.0（最差）到 1.0（最佳），具体取决于实现逻辑
- 用途：以单一数值概括整体性能
- 应用场景：模型对比、跟踪训练过程中的效果提升

### 各项指标

- 指标类型：用于分析的参考性指标
- 奖励类型：RFT 训练过程中使用的指标
- 结果解读：数值越高通常表示性能越好（除非设计了反向指标）

### 性能基准

何种表现为“良好”取决于实际使用案例：

| 分数区间      | 评估结果 | 处理建议        |
|-----------|------|-------------|
| 0.8 – 1.0 | 优秀   | 模型可直接部署     |
| 0.6 – 0.8 | 良好   | 可进行小幅优化     |
| 0.4 – 0.6 | 一般   | 需大幅优化提升     |
| 0.0 – 0.4 | 较差   | 检查训练数据和奖励函数 |

**⚠ Important**

以上仅为通用准则。请根据业务需求、基线模型性能、领域特定约束以及进一步训练的成本效益分析，自行定义阈值。

**问题排查****常见问题**

| 问题        | 原因            | 解决方案                        |
|-----------|---------------|-----------------------------|
| Lambda 超时 | 奖励计算复杂        | 延长 Lambda 超时时间或优化函数逻辑       |
| 权限被拒绝     | 缺少 IAM 权限     | 验证 SageMaker 角色是否可调用 Lambda |
| 评分不一致     | 非确定性奖励函数      | 使用固定种子或确定性逻辑                |
| 结果缺失      | Lambda 错误未被捕获 | 在 Lambda 中添加完善的错误处理机制       |

**调试检查清单**

- 验证输入数据格式是否正确（带有嵌套的内容数组）
- 确认 Lambda ARN 正确且函数已部署
- 检查 SageMaker → Lambda 调用的 IAM 权限
- 查看 CloudWatch 日志中的 Lambda 错误
- 验证 Lambda 响应符合预期格式

**最佳实践**

- 从简入手：先实现基础奖励函数，再逐步迭代优化
- 单独测试 Lambda：全面评测前先使用 Lambda 测试事件
- 小规模数据集验证：完整数据集评测前，先对数据子集执行评测

- 版本控制：同步追踪奖励函数与模型版本
- 成本监控：Lambda 调用次数与计算时长会影响成本
- 全面日志记录：在 Lambda 中使用打印语句便于调试
- 合理设置超时：在执行时长与成本之间取得平衡
- 指标文档化：明确定义各项指标的衡量标准

## 后续步骤

完成 RFT 评测后：

- 如果结果令人满意：将模型部署到生产环境
- 如果需要改进：
  - 调整奖励函数
  - 收集更多训练数据
  - 修改训练超参数
  - 运行额外的 RFT 训练迭代
- 持续监控：定期使用新数据重新评测

## 实现奖励函数

### 概述

奖励函数（也称为评分器或打分器）是评测模型响应并为训练提供反馈信号的核心组件。其必须作为 Lambda 函数实现，接受模型响应并返回奖励分数。

### 接口规范

奖励函数必须按以下格式接收与返回数据：

### 训练样本输入示例

```
{
  "messages": [
    {
      "role": "user",
      "content": "Do you have a dedicated security team?"
    }
  ],
}
```

```
"reference_answer": {
  "compliant": "No",
  "explanation": "As an AI developed by Company, I do not have a traditional
security team..."
}
```

## 奖励 Lambda 的有效载荷示例

容器在将数据发送到 Lambda 函数之前会自动进行转换，具体方式如下：

1. 为每个提示生成模型响应
2. 将 assistant 轮次（生成的响应）追加到 messages 数组中
3. 添加唯一的 id 字段用于跟踪

Lambda 函数将接收此转换格式的数据：

```
{
  "id": "123",
  "messages": [
    {
      "role": "user",
      "content": "Do you have a dedicated security team?"
    },
    {
      "role": "assistant",
      "content": "As an AI developed by Amazon, I don not have a dedicated
security team..."
    }
  ],
  # Following section will be same as your training dataset sample
  "reference_answer": {
    "compliant": "No",
    "explanation": "As an AI developed by Company, I do not have a traditional
security team..."
  }
}
```

## 奖励 Lambda 约定

```
def lambda_handler(event, context):
```

```

return lambda_grader(event)

def lambda_grader(samples: list[dict]) -> list[dict]:
    """
    Args:
        samples: List of dictionaries in OpenAI format

    Example input:
    {
        "id": "123",
        "messages": [
            {
                "role": "user",
                "content": "Do you have a dedicated security team?"
            },
            {
                "role": "assistant",
                "content": "As an AI developed by Company, I don't have a
dedicated security team..."
            }
        ],
        # This section will be same as your training dataset
        "reference_answer": {
            "compliant": "No",
            "explanation": "As an AI developed by Company, I do not have a
traditional security team..."
        }
    }

    Returns:
        List of dictionaries with reward scores:
    {
        "id": str,                    # Same id as input sample
        "aggregate_reward_score": float, # Overall score for the sample
        "metrics_list": [            # OPTIONAL: Component scores
            {
                "name": str,          # Name of the component score
                "value": float,       # Value of the component score
                "type": str           # "Reward" or "Metric"
            }
        ]
    }
    """

```

## 输入和输出字段

### 输入字段

| 字段                     | 说明               | 附加说明                      |
|------------------------|------------------|---------------------------|
| id                     | 样本的唯一标识符         | 在输出中原样返回。字符串格式            |
| 消息                     | OpenAI 格式的有序聊天记录 | 消息对象数组                    |
| messages[].role        | 消息发送方            | 常用值：user、assistant、system |
| messages[<br>].content | 消息文本内容           | 纯字符串                      |
| **metadata             | 用于辅助评分的自定义信息     | 对象类型；由训练数据传入的可选字段         |

### 输出字段

| 字段                         | 说明           | 附加说明                      |
|----------------------------|--------------|---------------------------|
| id                         | 与输入样本一致的标识符  | 必须与输入匹配                   |
| aggregate<br>_reward_score | 样本综合分数       | 浮点数（如 0.0 – 1.0 或任务自定义区间） |
| metrics_list               | 构成综合评分的各单项评分 | 指标对象数组                    |

### 技术约束

- 超时限制：每次 Lambda 调用最长执行时间为 15 分钟
- 并发能力：必须能够处理 rollout\_worker\_replicas \* 64 个并发请求
- 可靠性：必须实现完善的错误处理机制，并稳定返回有效评分
- 性能：优化执行速度（秒级而非分钟级），保障训练高效运行

### 最佳实践

- 尽量减少外部 API 调用
- 使用高效的算法与数据结构
- 为瞬时故障实现重试逻辑
- 缓存可重复使用的计算结果
- 训练前充分测试，确保执行无异常

## 使用自定义奖励函数

当存在针对特定任务的评测标准时，可实现自定义奖励函数：

- 定义评测标准：明确在当前任务中优质响应的判定依据
- 实现 Lambda 函数：按照接口规范创建 Lambda 函数
- 本地测试：验证函数对样本输入能返回正确评分
- 部署到 AWS：部署 Lambda 并记录其 ARN
- 配置配方：将 Lambda ARN 填入配方的 `reward_lambda_arn` 字段
- 小规模数据集测试：使用少量数据运行 RFT，验证集成效果

## IAM 权限

### 所需的权限

SageMaker 执行角色必须具备调用 Lambda 函数的权限。将此策略添加到 SageMaker 执行角色：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction"
      ],
      "Resource": "arn:aws:lambda:region:account-id:function:function-name"
    }
  ]
}
```

## Lambda 执行角色

Lambda 函数的执行角色需要基本的 Lambda 执行权限：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

其他权限：如果 Lambda 函数访问其他 AWS 服务（例如用于参考数据的 S3、用于日志记录的 DynamoDB），请将这些权限添加到 Lambda 执行角色。

### 示例：LLM-as-a-judge 奖励函数

此示例演示了如何使用 Amazon Bedrock 模型作为评判工具，通过将模型响应与参考答案进行比较来评测模型响应。此 Lambda 模板为用户提供框架，支持调用 Amazon Bedrock 发起推理请求，以完成评判评测处理。此 Lambda 函数与其他奖励函数采用一致的输入输出约定。

### 实施

此 Lambda 函数采用两阶段评测流程：lambda\_handler 模块从输入样本中提取模型响应与参考答案，随后 lambda\_graded 函数调用 Amazon Bedrock 对二者的语义相似度进行评分。该实现具备完善的错误处理机制，可对瞬时故障自动重试，并支持多种灵活的参考答案格式（字符串格式与结构化字典格式）。

### 实现细节：

- 重试逻辑：针对限流异常实现指数回退（1 秒、2 秒、4 秒），以应对 Bedrock API 速率节流
- 错误处理：评测失败时返回 0.0 分，而非抛出异常
- 确定性评分：使用 temperature=0.0 确保多次评测结果一致
- 灵活参考格式：自动支持字符串与字典两种格式的参考答案

- 分数裁剪：确保所有分数落在有效的 [0.0, 1.0] 区间内
- 模型无关：更改 JUDGE\_MODEL\_ID 即可使用任何 Amazon Bedrock 模型 ( Nova、Llama、Mistral 等 )

```
"""
LLM Judge Lambda POC - Working implementation using Amazon Bedrock
"""

import json
import time
import boto3

bedrock_runtime = boto3.client('bedrock-runtime', region_name='us-east-1')
JUDGE_MODEL_ID = "anthropic.claude-3-5-sonnet-20240620-v1:0"
SYSTEM_PROMPT = "You must output ONLY a number between 0.0 and 1.0. No explanations, no
text, just the number."

JUDGE_PROMPT_TEMPLATE = """Compare the following two responses and rate how similar
they are on a scale of 0.0 to 1.0, where:
- 1.0 means the responses are semantically equivalent (same meaning, even if worded
differently)
- 0.5 means the responses are partially similar
- 0.0 means the responses are completely different or contradictory

Response A: {response_a}

Response B: {response_b}

Output ONLY a number between 0.0 and 1.0. No explanations."""

def lambda_graded(response_a: str, response_b: str, max_retries: int = 3) -> float:
    """Call Bedrock to compare responses and return similarity score."""
    prompt = JUDGE_PROMPT_TEMPLATE.format(response_a=response_a, response_b=response_b)

    for attempt in range(max_retries):
        try:
            response = bedrock_runtime.converse(
                modelId=JUDGE_MODEL_ID,
                messages=[{"role": "user", "content": [{"text": prompt}]}],
                system=[{"text": SYSTEM_PROMPT}],
```

```

        inferenceConfig={"temperature": 0.0, "maxTokens": 10}
    )
    print(f"Bedrock call successful: {response}")
    output = response['output']['message']['content'][0]['text'].strip()
    score = float(output)
    print(f"Score parsed: {score}")
    return max(0.0, min(1.0, score))

except Exception as e:
    if "ThrottlingException" in str(e) and attempt < max_retries - 1:
        time.sleep(2 ** attempt)
    else:
        print(f"Bedrock call failed: {e}")
        return None
return None

def lambda_handler(event, context):
    """AWS Lambda handler - processes samples from RFTevalInvoker."""
    try:
        samples = event if isinstance(event, list) else [event]
        results = []

        for sample in samples:
            sample_id = sample.get("id", "unknown")
            messages = sample.get("messages", [])

            # Extract assistant response (response A)
            response_a = ""
            for msg in messages:
                if msg.get("role") in ["assistant", "nova_assistant"]:
                    response_a = msg.get("content", "")
                    break

            # Extract reference answer from root level (no longer in metadata)
            reference_answer = sample.get("reference_answer", "")

            # Handle both string and dict reference_answer formats
            if isinstance(reference_answer, dict):
                # If reference_answer is a dict, extract the explanation or compliant
                field
                response_b = reference_answer.get("explanation",
                reference_answer.get("compliant", ""))
            else:

```

```
        response_b = reference_answer

    if not response_a or not response_b:
        results.append({
            "id": sample_id,
            "aggregate_reward_score": 0.0,
            "metrics_list": [{"name": "similarity_score", "value": 0.0, "type":
"Metric"}]
        })
        continue

    # Get similarity score
    score = lambda_graded(response_a, response_b)

    results.append({
        "id": sample_id,
        "aggregate_reward_score": score,
        "metrics_list": [
            {
                "name": "similarity_score",
                "value": score,
                "type": "Metric"
            }
        ]
    })

    return {"statusCode": 200, "body": json.dumps(results)}

except Exception as e:
    print(f"Error: {e}")
    return {"statusCode": 500, "body": json.dumps({"error": str(e)})}
```

## 输入格式

Lambda 采用与其他奖励函数相同的输入格式：

```
{
  "id": "sample-001",
  "messages": [
    {
      "role": "user",
      "content": "Do you have a dedicated security team?"
    },
    {
```

```

        "role": "assistant",
        "content": "As an AI developed by Amazon, I don't have a dedicated security
team..."
    }
  ],
  "reference_answer": {
    "compliant": "No",
    "explanation": "As an AI developed by Company, I do not have a traditional
security team..."
  },
  "my_custom_field": "custom_value"
}

```

## 输出格式

```

{
  "id": "sample-001",
  "aggregate_reward_score": 0.85,
  "metrics_list": [
    {
      "name": "similarity_score",
      "value": 0.85,
      "type": "Metric"
    }
  ]
}

```

## 部署注意事项

您可能还需要根据所选模型的能力和 API 格式调整提示模板和推理参数。

- IAM 权限：Lambda 执行角色必须具备对所选模型的 `bedrock:InvokeModel` 调用权限
- 超时：将 Lambda 超时设置为至少 60 秒，以适配 Bedrock API 延迟和重试需求
- 区域：在所选 Bedrock 模型可用的区域部署
- 成本：监控 Bedrock API 使用量，每次评测中每个样本均会发起一次 API 调用
- 吞吐量：大规模评测时，申请提升 Bedrock 配额以避免节流

## 提高 Bedrock 吞吐量

如果在评测过程中遇到节流，可提升 Bedrock 模型配额：

- 进入 AWS 服务配额控制台
- 搜索 Bedrock 并选择对应区域
- 找到所选模型的配额项（例如“Claude 3.5 Sonnet 每分钟调用次数”）
- 点击“请求增加配额”并指定所需的吞吐量
- 提供增加配额的理由（例如“RFT 评测工作负载”）

Lambda 内置的重试逻辑可处理偶发节流，而持续高吞吐量评测需相应提升配额。

所需的 IAM 策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "bedrock:InvokeModel"
      ],
      "Resource": "arn:aws:bedrock:*::foundation-model/*"
    }
  ]
}
```

## 迭代训练

迭代训练是一种通过多轮训练周期对模型进行微调的系统化方法：每一轮训练都基于上一轮的检查点，通过评估发现模型的具体短板并加以改进。这种方法通过引入针对性示例修复失效场景、适配不断变化的需求，并以渐进式验证的方式提升模型效果，而非仅依赖单次长时间训练。流程通常遵循如下模式：先进行 SFT（监督式微调），再执行 RFT（基于奖励的微调）。模型检查点存储在 AWS 托管式托管 S3 存储桶中，可供后续训练迭代引用，同时在整个流程中保持模型类型与训练技术的一致性。

有关详细信息，请参考 [???](#)。

## 基于 SageMaker HyperPod 的 Amazon Nova 自定义

您可以通过 [Amazon Nova 配方](#) 自定义 Amazon Nova 模型（包括增强版 Amazon Nova 2.0 模型），并在 Hyperpod 上完成模型训练。配方是指 YAML 配置文件，该文件向 SageMaker AI 提供有关如何运行模型自定义作业的详细信息。SageMaker HyperPod 支持两种服务类型：Forge 与非 Forge。

HyperPod 使用优化的 GPU 实例和适用于 Lustre 的 Amazon FSx 存储提供高性能计算，通过集成 TensorBoard 等工具实现强大的监控，支持灵活的检查点管理以实现迭代改进，可无缝部署到 Amazon Bedrock 进行推理，并支持高效可扩展的多节点分布式训练。所有这些功能协同工作，为组织提供一个安全、高性能且灵活的环境，以便组织根据其特定业务需求定制 Amazon Nova 模型。

在 SageMaker HyperPod 上进行 Amazon Nova 模型自定义时，会将模型构件（包括模型检查点）存储在服务托管的 Amazon S3 存储桶中。服务托管存储桶中的构件使用 SageMaker AI 托管的 AWS KMS 密钥进行加密。服务托管的 Amazon S3 存储桶目前不支持使用客户托管的 KMS 密钥进行数据加密。您可以利用此检查点位置进行评估作业或 Amazon Bedrock 推理。

标准定价适用于计算实例、Amazon S3 存储和适用于 Lustre 的 FSx。有关定价详情，请参阅 [HyperPod 定价](#)、[Amazon S3 定价](#) 以及 [适用于 Lustre 的 FSx 定价](#)。

## Amazon Nova 2 模型的算力需求

下表汇总了针对 Amazon Nova 2 模型进行 SageMaker HyperPod 和 SageMaker AI 训练作业时的算力需求。

### Nova 2 训练要求

| 训练技术                             | 最小实例数 | 实例类型        | GPU 计数 | 备注                 | 支持的模型       |
|----------------------------------|-------|-------------|--------|--------------------|-------------|
| SFT ( LoRA )                     | 4     | P5.48xlarge | 16     | 参数高效微调             | Nova 2 Lite |
| SFT ( 全秩 )                       | 4     | P5.48xlarge | 32     | 全模型微调              | Nova 2 Lite |
| 基于 SageMaker 训练作业进行 RFT ( LoRA ) | 2     | P5.48xlarge | 16     | 在 AWS 环境中使用自定义奖励函数 | Nova 2 Lite |
| 基于 SageMaker 训练作业进行 RFT ( 全秩 )   | 4     | P5.48xlarge | 32     | 32K 上下文长度          | Nova 2 Lite |
| 基于 SageMaker HyperPod 进行 RFT     | 8     | P5.48xlarge | 64     | 默认 8192 上下文长度      | Nova 2 Lite |

| 训练技术 | 最小实例数 | 实例类型        | GPU 计数 | 备注              | 支持的模型       |
|------|-------|-------------|--------|-----------------|-------------|
| CPT  | 4     | P5.48xlarge | 16     | 每个实例每天约处理 4 亿词元 | Nova 2 Lite |

要在 Hyperpod 上优化 Amazon Nova 模型自定义 workflow，请遵循以下关于高效训练、资源管理及成功部署模型的最佳实践建议。

## Amazon Nova 自定义最佳实践

### 概述

本节概述了自定义技术，有助您根据自身需求与可用数据选择最优方案。

#### LLM 训练的两个阶段

大型语言模型训练包括两个主要阶段：预训练和后训练。在预训练阶段，模型会处理原始文本的词元，并针对下一词元预测进行优化。此过程会产生一个模式补全器，可从网络和精选文本中学习语法、语义、事实知识和推理模式。然而，预训练模型无法理解指令、用户目标，也不具备适配上下文的行为能力。其会依据训练数据分布所涵盖的任意风格，持续生成文本。预训练模型执行的是自动补全而非遵循指令，输出格式不统一，还可能复刻训练数据中的不良偏见与不安全内容。预训练构建的是通用能力，而非任务完成能力。

后训练则可将模式补全器转化为实用助手。首先进行多轮监督式微调 ( SFT )，通过模仿高质量的演示，教会模型遵循指令、遵守架构和策略、调用工具并生成可靠的输出。这种齐训练能让模型将提示词视为待执行的任务，而非单纯的续写文本。接着引入强化微调 ( RFT )，利用可衡量的反馈 ( 如验证器或 LLM-as-a-judge ) 来优化模型行为，在准确性与简洁性、安全性与覆盖范围，或约束条件下的多步骤推理等权衡中实现平衡。在实践中，通常采用 SFT 与 RFT 交替迭代的策略，将预训练模型打磨为一个稳定可靠、策略对齐且能一致性处理复杂任务的智能系统。

### 选择合适的自定义方法

本节将介绍后训练自定义策略：RFT 和 SFT。

#### 强化微调 ( RFT )

强化微调通过反馈信号 ( 即可衡量回复质量的分数或奖励 ) 提升模型性能，而非依靠精确标准答案进行直接监督。与从输入-输出对中学习的传统监督式微调不同，RFT 使用奖励函数评测模型响应，并迭代优化模型以最大化奖励。这种方法适用于难以定义精确输出但能可靠衡量响应质量的任务。RFT 使模

型能够通过尝试和反馈来学习复杂行为与偏好，非常适合需要精细决策、创造性问题求解，或需符合可程序化评测的特定质量标准的场景。例如，回答复杂法律问题就是 RFT 的理想使用案例，因为您希望教会模型如何更好地进行推理，从而更准确地回答问题。

## 工作原理

在强化微调中，您从经过指令微调的基线模型出发，将每个提示视作一场小型竞赛。对于给定输入，从模型中采样若干候选回答，通过奖励函数为每个回答打分并在本组内排序。更新步骤会促使模型提升高分候选回答的生成概率，降低低分回答的概率；同时通过“贴近基线”约束，避免模型行为偏移、内容冗长或钻规则漏洞。您在大量提示上重复这一循环，不断补充困难样本，在发现漏洞利用行为时强化验证器或优化评判规则，并持续跟踪任务指标。

## 何时使用 RFT

最适合使用 RFT 的任务通常具备以下共同特征。即便难以定义唯一正确输出，仍存在可衡量的成功信号；支持部分得分或分级质量评估，因此可对同一提示下的回答进行优劣排序，或通过奖励函数完成排序；涉及多项需兼顾的目标（例如在准确性与简洁性、清晰度、安全性或成本之间取得平衡）；需要遵守可通过程序校验的明确约束；在工具介导或基于环境的环境中运行，且结果可观察（成功或失败、延迟、资源使用情况）；处于低标注数据场景，即获取标准答案成本高昂，但自动化或基于评分规则的反馈十分丰富。当您能将质量转化为可靠的标量分值或排序，并希望模型优先强化高分行为，而无需大量标注样本时，RFT 效果最佳。

在以下情况下，建议考虑其他方法：

- 拥有充足且可靠的标注输入-输出对：使用 SFT
- 主要差距在于知识或术语：使用检索增强生成（RAG）
- 奖励信号存在噪声或不可靠，且无法通过优化评分规则或校验工具解决：在采用 RFT 之前先稳定奖励信号

## 何时不使用 RFT

在以下情况下应避免使用 RFT：

- 可低成本生成可靠的标注输入-输出对（SFT 更简单、低成本且更稳定）
- 差距在于知识或术语，而非行为（建议使用 RAG）
- 奖励信号存在噪声、稀疏、易被钻漏洞，或者计算成本高昂、速度缓慢（先优化评测器）
- 基线性能近乎为零（先通过 SFT 完成引导，再进行偏好优化）
- 任务具备确定性架构、严格格式要求或唯一正确答案（SFT 或基于规则的验证效果更佳）

- 严苛的延迟或成本预算，无法承担 RFT 所需的额外采样与探索开销
- 安全与策略约束定义不清晰，且无法在奖励函数中有效强制执行

如果能明确给出“正确答案”，请使用 SFT。如果需要补充新知识，请使用 RAG。仅当拥有稳健的基线模型，以及可靠、高效、难以被利用的奖励函数后，再使用 RFT。

## 监督式微调 ( SFT )

监督式微调是在针对任务构建的人工标注输入-输出数据集上训练 LLM。您可以提供提示 ( 问题、指令等 ) 示例以及正确或所需响应，并基于这些样本继续训练模型。调整模型权重以最大限度地减少监督损失 ( 通常是其预测与目标输出词元之间的交叉熵 )。这与大多数监督式机器学习任务中使用的训练方式相同，旨在对 LLM 进行专业化处理。

SFT 改变的是行为，而非知识。其不会让模型学到预训练阶段未见过的新事实或专业术语，只教会模型如何回答，而非掌握什么知识。如果需要新增领域知识 ( 如内部术语 )，可在推理阶段通过检索增强生成 ( RAG ) 提供相关上下文，由 SFT 赋予模型所需的指令遵循能力。

## 工作原理

SFT 通过最大限度地减少响应词元上的平均交叉熵损失来优化 LLM，将提示词元视为上下文并将其从损失计算中屏蔽。模型会将目标风格、结构和决策规则内化，学习为每个提示生成正确的补全。例如，要将文档分类为自定义类别，可以使用提示 ( 文档文本 ) 和带标签的补全 ( 类别标签 ) 来微调模型。您可以在这些数据对上训练模型，直到模型能够以高概率为每个提示输出正确的标签。

您可以使用少至几百个样本进行 SFT，也可以纵向扩展到几十万个样本。SFT 样本必须具备高质量，且与期望的模型行为直接对齐。

## 何时使用 SFT

当任务定义清晰、预期输出明确时，即可使用 SFT。如果您能明确给出“输入为 X 时，正确输出为 Y”的形式，并能收集此类映射样本，监督式微调便是理想选择。SFT 在以下场景中表现出色：

- 结构化或复杂的分类任务：将内部文档或合同分为多个自定义类别。通过 SFT，模型对这些特定类别的学习效果要远优于仅使用提示词。
- 具有已知答案的问答或转换任务：微调模型以回答公司知识库中的问题，或者在格式之间转换数据，其中每个输入都有对应的正确答案。
- 格式和风格一致性：通过根据正确格式或语气的样本进行微调，训练模型始终以特定格式或语气做出响应。例如，在展示特定品牌语气的提示-响应对上进行训练，可以使模型学会以该风格生成输出。同样，指令遵循行为通常也是通过在精选的良好助手行为样本上进行 SFT 来初步习得。

当您可以指定正确行为是什么样子时，SFT 是教授 LLM 新技能或行为最直接的方式。其利用模型现有的语言理解能力，并将其聚焦于任务上。若希望模型执行特定任务，且已有或可构建样本数据集，即可使用 SFT。

当您能够整理出高质量、高度贴合预期行为的提示-响应对时，也适合使用 SFT。该方法适用于目标明确或格式固定的任务，例如架构、函数或工具调用，以及适合以模仿作为训练信号的结构化回答等场景。其目标是行为塑造：让模型将提示视为任务、遵循指令、采用指定语气与拒答策略，并保持输出格式一致。建议至少准备数百个示范样本，数据质量、一致性与去重比样本数量更为重要。如需实现简单且高性价比的更新，可采用低秩适应 (LoRA) 等参数高效微调方法训练小型适配器，同时保持基座主体参数不变。

## 何时不使用 SFT

当差距在于知识而非行为时，不建议使用 SFT。其无法让模型学习新事实、专业术语或最新事件。此类场景应采用检索增强生成，在推理阶段引入外部知识。当您可以评估质量却无法标注唯一正确答案时，也应避免 SFT。请改用带可验证奖励或 LLM-as-a-judge 的强化微调，直接对奖励进行优化。如果需求或内容频繁变动，应依赖检索与工具使用，而非重新训练模型。

## 主题

- [Nova Forge SDK](#)
- [创建带有受限实例组 \( RIG \) 的 SageMaker HyperPod EKS 集群](#)
- [Amazon SageMaker HyperPod 核心命令指南](#)
- [Nova Forge 访问与设置](#)
- [Amazon Nova 模型训练](#)
- [评估经过训练的模型](#)
- [使用 MLflow 监控 HyperPod](#)

## Nova Forge SDK

Amazon Nova Forge SDK 是一套功能完整的 Python SDK，为整个 Amazon Nova 模型自定义生命周期提供统一的编程接口。该 SDK 通过提供统一且一致的 API，简化了模型自定义流程，支持在 Amazon SageMaker 和 Amazon Bedrock 平台上完成训练、评估、监控、部署与推理全流程操作。

有关更多信息，请参阅 [Nova Forge SDK](#)。

## 创建带有受限实例组 ( RIG ) 的 SageMaker HyperPod EKS 集群

要在 HyperPod 上对模型进行自定义，必须先完成必要的基础设施设置。如需了解如何设置带有受限实例组 ( RIG ) 的 SageMaker HyperPod EKS 集群，请参阅相关[讲习会](#)，其中提供了完整的设置步骤详解。

### Amazon SageMaker HyperPod 核心命令指南

Amazon SageMaker HyperPod 提供丰富的命令行功能，用于管理训练 workflow。本指南涵盖常见操作的核心命令，包括连接集群、监控作业进度。

#### 先决条件

使用这些命令前，请确保已完成以下配置：

- 已创建配备 RIG 的 SageMaker HyperPod 集群 ( 通常在 us-east-1 )
- 已创建用于存储训练构件的 Amazon S3 输出存储桶
- 已配置具备相应权限的 IAM 角色
- 已按正确的 JSONL 格式上传训练数据
- 已完成 FSx for Lustre 同步 ( 首次作业时请在集群日志中验证 )

#### 主题

- [安装配方 CLI](#)
- [连接到集群](#)
- [启动训练作业](#)
- [查看作业状态](#)
- [监控作业日志](#)
- [列出所有活跃作业](#)
- [取消作业](#)
- [运行评估作业](#)
- [常见问题](#)

#### 安装配方 CLI

执行安装命令前，请先导航到配方存储库根目录。

若使用非 Forge 自定义方案，请使用 Hyperpodrecipes 存储库；若基于 Forge 进行自定义，请参考专属 Forge 的配方存储库。

执行以下命令安装 SageMaker HyperPod CLI：

#### Note

确保并未处于活跃的 conda/anaconda/miniconda 环境或其他虚拟环境中  
若当前处于这样的环境中，请执行以下命令退出：

- conda/anaconda/miniconda 环境：conda deactivate
- Python 虚拟环境：deactivate

若使用的是非 Forge 自定义技术，请按如下方式下载 sagemaker-hyperpod-recipes 存储库：

```
git clone -b release_v2 https://github.com/aws/sagemaker-hyperpod-cli.git
cd sagemaker-hyperpod-cli
pip install -e .
cd ..
root_dir=$(pwd)
export PYTHONPATH=${root_dir}/sagemaker-hyperpod-cli/src/hyperpod_cli/
sagemaker_hyperpod_recipes/launcher/nemo/nemo_framework_launcher/launcher_scripts:
$PYTHONPATH
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
rm -f ./get_helm.sh
```

若为 Forge 订阅用户，您应按照如下流程下载配方。

```
mkdir NovaForgeHyperpodCLI
cd NovaForgeHyperpodCLI
aws s3 cp s3://nova-forge-c7363-206080352451-us-east-1/v1/ ./ --recursive
pip install -e .

curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
```

```
./get_helm.sh  
rm -f ./get_helm.sh
```

### Tip

如需在执行 `pip install -e .` 前使用[新的虚拟环境](#)，请运行：

- `python -m venv nova_forge`
- `source nova_forge/bin/activate`
- 此时命令行提示符开头会显示 `(nova_forge)`
- 这可确保使用 CLI 时不会出现冲突的依赖项

目的：为何要执行 `pip install -e .`？

该命令以可编辑模式安装 SageMaker HyperPod CLI，无需您每次重新安装，即可使用更新后的配方。其同时支持新增配方并被 CLI 自动识别。

## 连接到集群

运行任何任务前，请先使用 SageMaker HyperPod CLI 连接集群：

```
export AWS_REGION=us-east-1 && SageMaker HyperPod connect-cluster --cluster-name  
<your-cluster-name> --region us-east-1
```

### Important

该命令会生成后续命令必需的上下文文件 (`/tmp/hyperpod_context.json`)。若出现文件不存在错误，请重新执行连接命令。

专业提示：您可以通过在命令中添加 `--namespace kubeflow` 参数，进一步将集群配置为始终使用 `kubeflow` 命名空间，示例如下：

```
export AWS_REGION=us-east-1 && \  
hyperpod connect-cluster \  
--cluster-name <your-cluster-name> \  
--region us-east-1 \  
--namespace kubeflow
```

这样就无需在每条任务命令中都添加 `-n kubeflow`。

## 启动训练作业

### Note

若运行 PPO/RFT 任务，请务必在文件 `src/hyperpod_cli/sagemaker_hyperpod_recipes/recipes_collection/cluster/k8s.yaml` 中添加标签选择器配置，确保所有容器组 ( pod ) 调度到同一节点。

```
label_selector:
  required:
    sagemaker.amazonaws.com/instance-group-name:
      - <rig_group>
```

通过配方启动训练作业 ( 支持可选参数覆盖 ) :

```
hyperpod start-job -n kubeflow \
--recipe fine-tuning/nova/nova_1_0/nova_micro/SFT/nova_micro_1_0_p5_p4d_gpu_lora_sft \
--override-parameters '{
"instance_type": "ml.p5.48xlarge",
  "container": "708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-HP-SFT-latest"
}'
```

预期输出 :

```
Final command: python3 <path_to_your_installation>/NovaForgeHyperpodCLI/src/hyperpod_cli/sagemaker_hyperpod_recipes/main.py recipes=fine-tuning/nova/nova_micro_p5_gpu_sft cluster_type=k8s cluster=k8s base_results_dir=/local/home/<username>/results cluster.pullPolicy="IfNotPresent" cluster.restartPolicy="OnFailure" cluster.namespace="kubeflow" container="708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:HP-SFT-DATAMIX-latest"
```

```
Prepared output directory at /local/home/<username>/results/<job-name>/k8s_templates
Found credentials in shared credentials file: ~/.aws/credentials
Helm script created at /local/home/<username>/results/<job-name>/<job-name>_launch.sh
Running Helm script: /local/home/<username>/results/<job-name>/<job-name>_launch.sh
```

```
NAME: <job-name>
```

```

LAST DEPLOYED: Mon Sep 15 20:56:50 2025
NAMESPACE: kubeflow
STATUS: deployed
REVISION: 1
TEST SUITE: None
Launcher successfully generated: <path_to_your_installation>/NovaForgeHyperpodCLI/src/
hyperpod_cli/sagemaker_hyperpod_recipes/launcher/nova/k8s_templates/SFT

{
  "Console URL": "https://us-east-1.console.aws.amazon.com/sagemaker/home?region=us-
east-1#/cluster-management/<your-cluster-name>"
}

```

## 查看作业状态

使用 `kubectl` 监控运行中的作业：

```
kubectl get pods -o wide -w -n kubeflow | (head -n1 ; grep <your-job-name>)
```

## 容器组 ( pod ) 状态说明

下表说明常见的容器组 ( pod ) 状态：

| Status           | 说明                                    |
|------------------|---------------------------------------|
| Pending          | 容器组 ( pod ) 已被接受，但尚未调度到节点，或正在等待拉取容器映像 |
| Running          | 容器组 ( pod ) 已绑定到节点，且至少有一个容器正在运行或启动中   |
| Succeeded        | 所有容器均成功完成，且不会重启                       |
| Failed           | 所有容器已终止，且至少有一个容器执行失败                  |
| Unknown          | 无法确定容器组 ( pod ) 状态 ( 通常因节点通信问题导致 )    |
| CrashLoopBackOff | 容器反复启动失败；Kubernetes 暂缓重启尝试 ( 退避重试 )   |

| Status                          | 说明                              |
|---------------------------------|---------------------------------|
| ImagePullBackOff / ErrImagePull | 无法从注册表拉取容器映像                    |
| OOMKilled                       | 容器因超出内存限制被终止                    |
| Completed                       | 作业或容器组 ( pod ) 成功完成 ( 批处理作业完成 ) |

### Tip

使用 `-w` 参数可实时监控容器组 ( pod ) 状态更新，按下 `Ctrl+C` 可停止监控。按下 `Ctrl+C` 可停止监控。

## 监控作业日志

您可通过以下三种方式之一查看日志：

### 使用 CloudWatch

日志存储在包含 HyperPod 集群的 AWS 账户对应的 CloudWatch 中。如需在浏览器中查看，请导航到账户中的 CloudWatch 主页，然后搜索集群名称。例如，若集群名为 `my-hyperpod-rig`，则日志组的前缀为：

- 日志组：`/aws/sagemaker/Clusters/my-hyperpod-rig/{UUID}`
- 进入日志组后，可通过节点实例 ID ( 如 `hyperpod-i-00b3d8a1bf25714e4` ) 定位具体日志。
  - 其中 `i-00b3d8a1bf25714e4` 表示当前训练作业所运行的 HyperPod 节点友好名称。回想一下，在之前执行的命令 `kubectl get pods -o wide -w -n kubeflow | (head -n1 ; grep my-cpt-run)` 的输出中，我们已经获取到了名为 `NODE` 的列。
  - 本例中，“主”节点的作业运行在 `hyperpod-i-00b3d8a1bf25714e4` 上，因此需通过该字符串筛选日志组。选择名称为 `SagemakerHyperPodTrainingJob/rig-group/[NODE]` 的日志流即可查看

### 使用 CloudWatch Insights

若已掌握作业名称，且不想执行上述全部步骤，可直接在 `/aws/sagemaker/Clusters/my-hyperpod-rig/{UUID}` 下查询所有日志，以定位对应日志。

CPT :

```
fields @timestamp, @message, @logStream, @log
| filter @message like /(?!Starting CPT Job/
| sort @timestamp desc
| limit 100
```

如需查询作业完成状态，可将 Starting CPT Job 替换为 CPT Job completed

然后，您可以单击结果，并选择显示“Epoch 0”的结果，因为那将是主节点。

使用 AWS CLI

您也可以通过 AWS CLI 实时跟踪日志。操作前，请先使用 `aws --version` 检查 AWS CLI 版本。建议使用提供的工具脚本，以便在终端中实时跟踪日志

V1 :

```
aws logs get-log-events \
--log-group-name /aws/sagemaker/YourLogGroupName \
--log-stream-name YourLogStream \
--start-from-head | jq -r '.events[].message'
```

V2 :

```
aws logs tail /aws/sagemaker/YourLogGroupName \
--log-stream-name YourLogStream \
--since 10m \
--follow
```

## 列出所有活跃作业

查看集群中所有正在运行的作业：

```
hyperpod list-jobs -n kubeflow
```

输出示例：

```
{
  "jobs": [
```

```
{
  "Name": "test-run-nhgza",
  "Namespace": "kubeflow",
  "CreationTime": "2025-10-29T16:50:57Z",
  "State": "Running"
}
]
```

## 取消作业

随时停止正在运行的作业：

```
hyperpod cancel-job --job-name <job-name> -n kubeflow
```

## 查找作业名称

选项 1：从配方中获取

作业名称已在配方的 run 代码块中指定：

```
run:
  name: "my-test-run" # This is your job name
  model_type: "amazon.nova-micro-v1:0:128k"
  ...
```

选项 2：通过 list-jobs 命令获取

执行 `hyperpod list-jobs -n kubeflow` 命令，复制输出结果中的 Name 字段即可。

## 运行评估作业

使用评估配方对已训练模型或基础模型进行评估。

### 先决条件

运行评估作业前，确保已准备好以下内容：

- 来自训练作业的 manifest.json 文件的检查点 Amazon S3 URI（适用于经过训练的模型）
- 已按正确格式上传到 Amazon S3 的评估数据集
- 用于存储评估结果的 Amazon S3 输出路径

## 命令

执行以下命令启动评估作业：

```
hyperpod start-job -n kubeflow \  
  --recipe evaluation/nova/nova_2_0/nova_lite/  
nova_lite_2_0_p5_48x1_gpu_bring_your_own_dataset_eval \  
  --override-parameters '{  
    "instance_type": "p5.48xlarge",  
    "container": "708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-evaluation-repo:SM-  
HP-Eval-latest",  
    "recipes.run.name": "<your-eval-job-name>",  
    "recipes.run.model_name_or_path": "<checkpoint-s3-uri>",  
    "recipes.run.output_s3_path": "s3://<your-bucket>/eval-results/",  
    "recipes.run.data_s3_path": "s3://<your-bucket>/eval-data.jsonl"  
  }'
```

参数描述：

- `recipes.run.name`：评估作业的唯一名称
- `recipes.run.model_name_or_path`：来自 `manifest.json` 的 Amazon S3 URI，或基础模型路径（如 `nova-micro/prod`）
- `recipes.run.output_s3_path`：评估结果的 Amazon S3 存储位置
- `recipes.run.data_s3_path`：评估数据集的 Amazon S3 存储位置

提示：

- 模型专属配方：不同规格的模型（`micro`、`lite`、`pro`）对应各自的评估配方
- 基础模型评测：评估基础模型时，使用基础模型路径（如 `nova-micro/prod`）而非检查点 URI

评估数据格式

输入格式（JSONL）：

```
{  
  "metadata": "{key:4, category:'apple'}",  
  "system": "arithmetic-patterns, please answer the following with no other words: ",  
  "query": "What is the next number in this series? 1, 2, 4, 8, 16, ?",  
  "response": "32"
```

```
}
```

### 输出格式：

```
{
  "prompt": "[{'role': 'system', 'content': 'arithmetic-patterns, please answer the following with no other words: '}, {'role': 'user', 'content': 'What is the next number in this series? 1, 2, 4, 8, 16, ?'}]",
  "inference": "['32']",
  "gold": "32",
  "metadata": "{key:4, category:'apple'}"
}
```

### 字段描述：

- `prompt`：发送给模型的格式化输入内容
- `inference`：模型生成的响应
- `gold`：输入数据集中的预期正确答案
- `metadata`：从输入中传递的可选元数据

### 常见问题

- `nemo_launcher` 您可能需要根据 `hyperpod_cli` 的安装路径，将 `ModuleNotFoundError: No module named 'nemo_launcher'` 添加到 Python 路径中。示例命令：

```
export PYTHONPATH=<path_to_hyperpod_cli>/sagemaker-hyperpod-cli/src/hyperpod_cli/
sagemaker_hyperpod_recipes/launcher/nemo/nemo_framework_launcher/launcher_scripts:
$PYTHONPATH
```

- `FileNotFoundError: [Errno 2] No such file or directory: '/tmp/hyperpod_current_context.json'` 说明您未执行 `hyperpod connect cluster` 命令。
- 若未看到作业被调度，请检查 SageMaker HyperPod CLI 的输出中是否包含任务名称及其他元数据。若未包含，请执行以下命令重新安装 Helm 图表：

```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/
get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
rm -f ./get_helm.sh
```

## Nova Forge 访问与设置

要设置可用于作业的 Amazon Nova Forge，您需要：

- 订阅 Amazon Nova Forge
- 设置集群

### 主题

- [订阅 Amazon Nova Forge](#)
- [设置基础设施](#)
- [负责任的人工智能](#)

## 订阅 Amazon Nova Forge

要访问 Amazon Nova Forge 功能，请完成如下步骤：

1. 验证管理员对 AWS 账户的访问权限。
2. 进入 SageMaker AI 控制台并[请求访问 Amazon Nova Forge](#)。
3. 订阅申请通过后，请等待 Amazon Nova 团队发送确认邮件通知。
4. 使用 `forge-subscription` 标签标记执行角色。标签是访问 Amazon Nova Forge 功能和检查点的必需条件。将以下标签添加到执行角色：
  - 键：`forge-subscription`
  - 值：`true`

### Note

无需订阅 Forge，标准 Amazon Nova 功能仍可正常使用。Amazon Nova Forge 专为构建自定义前沿模型而设计，可在模型训练全阶段提供可控性与灵活性。

## 设置基础设施

Amazon Nova Forge 订阅获批后，请配置所需基础设施来启用支持 Forge 的功能。有关创建带有受限实例组 ( RIG ) 的 EKS 集群的详细说明，请按照[讲习会说明](#)进行操作。

## 负责任的人工智能

内容审核设置：Amazon Nova Forge 客户可以为 Amazon Nova Lite 1.0 和 Pro 1.0 模型使用可自定义内容审核设置 (CCMS)。CCMS 允许您调整内容审核控制，以符合特定的业务需求，同时保持必要的负责任的人工智能保护措施。要确定业务使用案例是否符合 CCMS 条件，请联系 Amazon Web Services 客户经理。

Amazon Nova Forge 提供负责任的人工智能工具包，其中包括训练数据、评估基准测试和运行时控制，以帮助您使模型符合 Amazon Nova 负责任的人工智能指南。

训练数据：数据混合中的“RAI”类别包含强调负责任的人工智能原则、安全考虑和负责任技术部署的案例和场景。在持续预训练期间，使用这些数据来负责任地使模型对齐。

评估：提供了基准测试任务，用于测试模型检测和拒绝不当、有害或不正确内容的能力。使用这些评估来衡量基础模型性能与自定义模型性能之间的差异。

## Amazon Nova 模型训练

SageMaker HyperPod 支持多种训练 Amazon Nova 模型的技术，包括持续预训练 (CPT)、监督式微调 (SFT) 和强化微调 (RFT)。每种技术适用于不同的自定义需求，并可应用于不同版本的 Amazon Nova 模型。

### 主题

- [持续预训练 \(CPT\)](#)
- [监督式微调 \(SFT\)](#)
- [SageMaker HyperPod 上的强化微调 \(RFT\)](#)

## 持续预训练 (CPT)

持续预训练 (CPT) 是一种通过让基础模型接触来自特定领域或语料库的额外无标注文本，从而延长模型预训练阶段的训练技术。监督式微调需要带标注的输入-输出样本对，而 CPT 直接基于原始文档进行训练，帮助模型掌握更深的新领域知识、学习领域专属术语与行文风格，并适配特定内容类型或主题领域。

当您拥有海量 (数百亿词元级) 的领域专属文本数据 (如法律文书、医学文献、技术文档或企业专有内容)，并希望模型在该领域具备原生流畅度时，这种方法尤为适用。通常，在 CPT 阶段之后，模型还需要经过额外的指令微调阶段，才能将新学到的知识用于完成实际任务。

### 支持的模型

CPT 适用于以下 Amazon Nova 模型：

- Nova 1.0 ( Micro、Lite、Pro )
- Nova 2.0 ( Lite )

何时选用 Nova 1.0 与 Nova 2.0

Amazon Nova 系列模型提供多种高性价比方案，助力在精准度、速度与成本之间实现最优平衡。

需要以下特性时，建议选择 Nova 2.0：

- 具备适用于复杂分析任务的高级推理能力
- 在代码生成、数学及科学问题求解方面表现更出色
- 支持更长的上下文长度
- 具备更优的多语言处理能力

#### Note

模型并非越大越优。在 Nova 1.0 与 Nova 2.0 之间选型时，请综合考虑性价比与具体业务需求。

Nova 2.0 上的 CPT

Amazon Nova Lite 2.0 是一款推理模型，其训练所用数据集相较于 Nova Lite 1.0 规模更大、种类更丰富。虽然 Nova Lite 2.0 模型规模更大，但其推理速度却快于 Nova Lite 1.0，同时具备更强的推理能力、更长的上下文长度以及更优的多语言处理性能。

借助 Nova 2.0 上的 CPT，您可通过自身领域专属数据拓展这些高级能力，使模型在专业领域形成深度专长，同时保持其出色的推理与分析能力。

CPT 配方示例

以下是 CPT 配方示例。您可以在[配方](#)存储库中找到此配方和其他配方。

```
# Note:  
# This recipe can run on p5.48xlarge  
# Run config  
run:
```

```

name: "my-cpt-run" # A descriptive name for your training
job
model_type: "amazon.nova-2-lite-v1:0:256k" # Model variant specification, do not
change
model_name_or_path: "nova-lite-2/prod" # Base model path, do not change
replicas: 8 # Number of compute instances for
training, allowed values are 4, 8, 16, 32
data_s3_path: "" # Customer data paths
validation_data_s3_path: "" # Customer validation data paths
output_s3_path: "" # Output artifact path, job-specific
configuration - not compatible with standard SageMaker Training Jobs
mlflow_tracking_uri: "" # Required for MLFlow
mlflow_experiment_name: "my-cpt-experiment" # Optional for MLFlow. Note: leave this
field non-empty
mlflow_run_name: "my-cpt-run" # Optional for MLFlow. Note: leave this
field non-empty

## Training specific configs
training_config:
  task_type: cpt
  max_length: 8192 # Maximum context window size (tokens)
  global_batch_size: 256 # Global batch size, allowed values are
32, 64, 128, 256.

  trainer:
    max_steps: 10 # The number of training steps to run
total
    val_check_interval: 10 # The number of steps between running
validation. Integer count or float percentage
    limit_val_batches: 2 # Batches of the validation set to use
each trigger

  model:
    hidden_dropout: 0.0 # Dropout for hidden states, must be
between 0.0 and 1.0
    attention_dropout: 0.0 # Dropout for attention weights, must
be between 0.0 and 1.0

  optim:
    optimizer: adam
    lr: 1e-5 # Learning rate
    name: distributed_fused_adam # Optimizer algorithm, do not change
    adam_w_mode: true # Enable AdamW mode
    eps: 1e-06 # Epsilon for numerical stability

```

```

weight_decay: 0.0 # L2 regularization strength, must be
between 0.0 and 1.0
adam_beta1: 0.9 # Beta1 for Adam optimizer
adam_beta2: 0.95 # Beta2 for Adam optimizer
sched:
  warmup_steps: 10 # Learning rate warmup steps
  constant_steps: 0 # Steps at constant learning rate
  min_lr: 1e-6 # Minimum learning rate, must be lower
than lr

```

为 2.0 上的 CPT 做数据准备

数据格式要求

训练数据集与验证数据集必须为 JSONL 文件，并遵循以下格式：每行包含一个 JSON 对象，用于表示一段对话，且需包含必填字段与指定结构。示例如下：

```

{"text": "AWS stands for Amazon Web Services"}
{"text": "Amazon SageMaker is a fully managed machine learning service"}
{"text": "Amazon Bedrock is a fully managed service for foundation models"}

```

文本条目应包含代表目标领域的自然流畅的高质量内容。

测试数据是否能够转换为 [Arrow 格式](#)。使用下方的 python 脚本来解决这个问题。确保至少使用 datasets==2.18.0 版本：

```

from datasets import load_dataset, load_from_disk
from pathlib import Path

input_path = Path("<Your jsonl file>")
output_path = Path("<Your output directory>")

dataset = load_dataset("json", data_files=str(input_path), split="train")
dataset.save_to_disk(str(output_path), max_shard_size="1GB")

try:
    test_dataset = datasets.load_from_disk(output_dir)
    print(f"Dataset loaded successfully #! Contains {len(test_dataset)} samples")
except Exception as e:
    print(e)

```

打印的行数应与 JSONL 文件中的行数相同。

使用数据混合时，首个作业请设置 `max_steps=2`。这有助于在集群中为数据访问生成优化配置，并确保所有数据混合均可用。

## 如何为 CPT 准备数据

训练数据是决定持续预训练能否成功的最关键因素。虽然 CPT 数据常被称作“无标注数据”，但实际情况远比这复杂。数据的结构、格式与呈现方式，直接决定模型能否习得使用案例所需的知识与能力。

### 为 CPT 准备结构化的业务数据集

这对于构建专属领域基础模型的企业与机构而言，是一项普遍挑战。多数企业都拥有丰富的结构化数据资源，包括产品目录、用户档案、交易日志、表单提交记录、API 调用信息及运营元数据等。乍看之下，这类数据与标准预训练中常用的非结构化 Web 文本差异显著。

若要让模型从结构化业务数据中高效学习，需充分考量下游任务，并合理设计数据呈现方式，引导模型学习正确的预测关联。

要充分发挥持续预训练的潜力，需考虑以下几点：

- 模型在推理阶段需执行的任务
- 原始数据中包含的信息
- 如何对数据进行结构化处理，使模型能够正确提取并运用信息

将结构化数据直接用于训练，无法让模型掌握相关推理能力。需主动调整数据的呈现方式，引导模型的学习方向。

以下章节将通过后续章节将通过文献综述阐述数据增强的重要性，并给出结构化业务数据的数据增强策略示例，为您在 CPT 中处理与组织业务数据集提供实用参考。

### 文献中的 CPT 结构化数据

CPT 能够将领域事实注入模型，但在输入或任务发生变化时，模型往往难以有效检索和运用这些事实。对照实验表明，若预训练阶段未采用多样化数据增强，模型会以僵化方式记忆事实信息，即便后续开展指令微调，也依然难以提取利用。相关研究建议在训练初期就注入类指令信号。对于半结构化数据，随机序列化及其他增强手段可降低架构过拟合，因此 CPT 应与指令类任务交替进行，而非先执行 CPT 再开展 IFT。金融领域相关研究进一步发现，相较于分步执行配方，在批次层面混合 CPT 与指令数据，可提升泛化能力并减少知识遗忘。Qwen 技术报告也得出一致结论：将高质量指令数据融入预训练本身，可使模型在学习新领域知识的同时，增强上下文学习能力并保持指令遵循能力。

对半结构化语料库实施数据增强是关键手段。合成图感知 CPT 可将小规模领域数据集扩展为实体关联语料库，显式学习实体关系与复合结构，并在推理阶段结合检索能力使用。在金融领域，CPT 与指令

数据混合训练的效果优于分步流程；同时，利用通用数据平衡领域数据可减轻模型通用能力的退化。大规模领域 CPT 也能保留较广泛的通用能力，甚至可通过模型融合进行取舍，但研究仍表明指令微调是必不可少后续步骤，这进一步印证了在 CPT 阶段引入指令信号的重要价值。

### 通过随机化和乱序注入多样性

一种能帮助模型从结构化及半结构化数据集中高效学习的通用策略，是打乱数据集中字段的顺序，甚至随机丢弃部分键值。

打乱字段顺序可迫使模型去理解每个值的具体含义，而非依赖其出现位置，并学习所有字段间的关联关系。例如，以 Amazon 商店上架的某款电子游戏数据为例，当“标题”“平台”“价格”“品相”“版本”等字段以不同顺序排列时，模型无法再依赖“第三个字段是平台”这类位置信息，而必须将标签与对应值绑定，进而掌握各属性间的双向关联：标题  $\rightleftharpoons$  平台、平台  $\rightleftharpoons$  价格、品相  $\rightleftharpoons$  价格。如此一来，模型便能实现诸多实用推理，比如根据游戏名称和观察到的价格推断可能的平台，或依据标题与平台估算合理的价格区间。

在序列化过程中随机丢弃键值，其作用类似于特征丢弃：可防止模型对任一单个字段产生共适应现象，进而迫使模型从剩余的有效信息中还原缺失内容。若“平台”字段缺失，模型需从标题字符串或兼容性文本中提取平台相关信息；若“价格”字段被隐藏，模型则需结合平台、版本及品相信息进行综合推断。这不仅建立了双向对称性（ $A \rightarrow B$  与  $B \rightarrow A$ ），增强了对杂乱真实世界列表的稳健性，还能在字段缺失、重命名或顺序调整时保持结构不变性。

下面以购物场景示例具体说明。对同一商品采用多种序列化方式，例如“标题：《艾尔登法环》| 平台：PlayStation 5 | 品相：二手如新 | 价格：34.99 美元”，以及另一种排列形式“价格：34.99 美元 | 标题：《艾尔登法环》| 品相：二手如新 | 平台：PlayStation 5”；在部分训练轮次中，可删除“平台”字段，但在描述中保留“兼容 PS5”的相关表述。同时训练互补性目标，比如根据 {标题、价格} 推断平台，以及根据 {标题、平台} 确定价格区间。由于字段的顺序甚至存在与否均会随机变化，模型唯一稳定的学习策略，就是掌握各属性之间的真实关联，而非机械记忆固定模板。

### 数据的呈现方式至关重要

LLM 通过从已看到的内容中预测下一个词元来学习。因此，训练期间显示的字段和事件的顺序决定了模型能够学习到什么。如果训练格式与实际任务相匹配，损失就会落在确切的决策词元上。如果字段没有结构地随意堆叠，模型会学习捷径或记忆热门内容，这导致模型在需要从多个选项中做出选择时失败。

先展示情境，再展示选项，最后展示决策。如果模型还需要学习结果或解释，则将其放在决策之后。

### 为 CPT 打包样本

什么是打包？

简单来说，打包就是将多个完整样本填入训练数据中的每个序列窗口，使窗口内充满有效词元，而非填充符。

### 为什么这非常重要

训练时，设置最大上下文长度，例如 8192 个词元。批次格式为 [批次大小 x 上下文长度]。如果某个训练样本长度不足上下文长度，剩余位置将用填充符补齐。即使在损失计算中对填充部分进行掩码处理，这些填充内容仍会参与注意力机制与 MLP 内核运算，从而消耗算力却不产生任何学习信号。

### 如何打包？

打包多个样本时，可使用 [DOC] 分隔符将多个训练样本拼接（注意 [DOC] 前后各保留一个空格），使拼接后的总长度不超过目标上下文长度。

打包后的文档示例如下：

```
{"text": "training sample 1 [DOC] training sample 2 [DOC] training sample 3"}
```

### CPT 调优参数

可用于 CPT 微调的参数包括：

#### 运行配置

- name：训练作业的描述性名称。这有助于在 AWS 管理控制台中识别对应作业。
- model\_type：要使用的 Amazon Nova 模型变体。可用选项为 amazon.nova-2-lite-v1:0:256k。
- model\_name\_or\_path：用于训练的基本模型的路径。可用选项为 nova-lite-2/prod，或训练后检查点的 S3 路径 (s3://customer-escrow-bucket-unique\_id/training\_run\_name)。
- replicas：要在分布式训练中使用的计算实例数。可用值因您所选的模型而异。Amazon Nova Lite 2.0 支持 4、8、16 或 32 个副本。
- data\_s3\_path：训练数据集的 S3 位置，格式为 JSONL 文件。此文件必须与集群位于同一 AWS 账户和区域中。提供的所有 S3 位置必须位于同一账户和区域中。
- validation\_data\_s3\_path：（选填）验证数据集的 S3 位置，格式为 JSONL 文件。此文件必须与集群位于相同的账户和区域中。提供的所有 S3 位置必须位于同一账户和区域中。
- output\_s3\_path：存储清单和 TensorBoard 日志的 S3 位置。提供的所有 S3 位置必须位于同一 AWS 账户和 AWS 区域中。
- mlflow\_tracking\_uri：用于 MLflow 日志记录的 MLflow 应用程序的 ARN

- `mlflow_experiment_name` : MLflow 实验名称
- `mlflow_run_name` : MLflow 运行名称

## 训练配置

- `max_length` : 以词元为单位的最大序列长度。这决定了训练的上下文窗口大小。CPT 支持的最大值为 8192 个词元。

更长的序列将会提高训练效率，但会以增加内存需求为代价。建议将 `max_length` 参数设置为与数据分布相匹配。

- `global_batch_size` : 所有设备与 Worker 节点在一次前向及反向传播中共同处理的训练样本总数。

该值乘以每台设备的批量大小和设备数量。它会影响训练的稳定性和吞吐量。我们建议从适合您内存的批量大小开始，然后进行扩展。对于特定领域的的数据，较大的批量大小可能会使梯度过于平滑。

## 训练器设置

- `max_steps` : 训练迭代步数。每一步训练模型所用的元素数量即 `global_batch_size`

## 模型设置

- `hidden_dropout` : 隐藏状态输出的丢弃概率。将该值增加约 0.0 到 0.2，以减少对较小数据集的过度拟合。有效值介于 0 到 1 之间（含两端值）。
- `attention_dropout` : 注意力权重的丢弃概率。此参数有助于泛化。有效值介于 0 到 1 之间（含两端值）。

## 优化器配置

- `lr` : 学习率，控制优化期间的步长。为了获得良好的性能，我们建议使用介于  $1e-6$  和  $1e-4$  之间的值。有效值介于 0 到 1 之间（含两端值）。
- `name` : 优化器算法。目前仅支持 `distributed_fused_adam`。
- `weight_decay` : L2 正则化强度。值比较高（介于 0.01 到 0.1 之间）会增加正则化强度。
- `warmup_steps` : 逐步提高学习率的步数。这可以提高训练稳定性。有效值介于 1 到 20 之间（含两端值）。

- `min_lr` : 衰减结束时的最低学习率。有效值介于 0 到 1 之间 ( 含两端值 ) , 但必须小于学习率。

## 监督式微调 ( SFT )

SFT 训练流程主要包含两个阶段 :

- 数据准备 : 遵循既定指南, 创建、清理或重新格式化数据集, 使其符合所需结构。确保输入、输出和辅助信息 ( 例如推理轨迹或元数据 ) 正确对齐和格式化。
- 训练配置 : 定义模型的训练方式。使用时, 此配置是在 YAML 配方文件中编写的, 包括 :
  - 数据来源路径 ( 训练和验证数据集 )
  - 关键超参数 ( epoch、学习率、批次大小 )
  - 可选组件 ( 分布式训练参数等 )

## Nova 模型对比与选型

Amazon Nova 2.0 模型的训练数据集比 Nova 1.0 模型规模更大、种类更丰富。重要改进包括 :

- 增强的推理能力, 支持显式推理模式
- 更广泛的多语言性能, 覆盖更多语种
- 在代码生成、工具调用等复杂任务上表现更优
- 扩展的上下文处理能力, 在更长上下文中实现更高的准确性和稳定性

## 何时选用 Nova 1.0 与 Nova 2.0

满足以下条件时, 建议选择 Amazon Nova 2.0 :

- 需要具有高级推理能力的卓越性能
- 需要多语言支持或复杂的任务处理
- 需要在代码生成、工具调用或分析任务上获得更优效果

## Nova 2.0 上的 SFT

Amazon Nova Lite 2.0 提供增强型监督式微调功能, 包括高级推理模式、经改进的多模态理解和经扩展的上下文处理。Nova 2.0 上的 SFT 使您能够针对自己的特定使用案例适配这些强大功能, 同时保持模型在处理复杂任务时的卓越性能。

Nova 2.0 上的 SFT 的主要特性包括：

- 推理模式支持：训练模型在给出最终答案前生成显式推理轨迹，从而增强分析能力。
- 高级多模态训练：针对文档理解 ( PDF )、视频理解和基于图像的任务进行微调，准确率更高。
- 工具调用功能：训练模型有效使用外部工具和函数调用，以应对复杂 workflows。
- 经扩展的上下文支持：利用更长的上下文窗口，在文档密集型应用中获得更佳稳定性和准确率。

### Note

有关适用的容器映像或示例配方的更多信息，请访问 [Amazon Nova 配方](#)。

## 主题

- [推理模式选择 \( 仅限 Nova 2.0 \)](#)
- [工具调用数据格式](#)
- [文档理解数据格式](#)
- [SFT 视频理解](#)
- [数据上传说明](#)
- [创建微调作业](#)
- [SFT 调优参数](#)
- [超参数指导](#)

## SFT 配方示例

以下是 SFT 配方示例。您可以在[配方](#)存储库中找到此配方和其他配方。

```
run:
  name: my-full-rank-sft-run
  model_type: amazon.nova-2-lite-v1:0:256k
  model_name_or_path: nova-lite-2/prod
  data_s3_path: s3://my-bucket-name/train.jsonl # only and not compatible with
SageMaker Training Jobs
  replicas: 4 # Number of compute instances for
training, allowed values are 4, 8, 16, 32
  output_s3_path: s3://my-bucket-name/outputs/ # Output artifact path (HyperPod job-
specific; not compatible with standard SageMaker Training Jobs)
```

```

mlflow_tracking_uri: "" # Required for MLFlow
mlflow_experiment_name: "my-full-rank-sft-experiment" # Optional for MLFlow. Note:
leave this field non-empty
mlflow_run_name: "my-full-rank-sft-run" # Optional for MLFlow. Note: leave
this field non-empty

training_config:
  max_steps: 100 # Maximum training steps. Minimal is 4.
  save_steps: ${oc.select:training_config.max_steps} # How many training steps the
checkpoint will be saved
  save_top_k: 5 # Keep top K best checkpoints. Note supported only
for jobs. Minimal is 1.
  max_length: 32768 # Sequence length (options: 8192, 16384, 32768
[default], 65536)
  global_batch_size: 32 # Global batch size (options: 32, 64, 128)
  reasoning_enabled: true # If data has reasoningContent, set to true;
otherwise False

  lr_scheduler:
    warmup_steps: 15 # Learning rate warmup steps. Recommend 15% of
max_steps
    min_lr: 1e-6 # Minimum learning rate, must be between 0.0 and
1.0

  optim_config: # Optimizer settings
    lr: 1e-5 # Learning rate, must be between 0.0 and 1.0
    weight_decay: 0.0 # L2 regularization strength, must be between 0.0
and 1.0
    adam_beta1: 0.9 # Exponential decay rate for first-moment
estimates
    adam_beta2: 0.95 # Exponential decay rate for second-moment
estimates

  peft: # Parameter-efficient fine-tuning (LoRA)
    peft_scheme: "null" # Disable LoRA for PEFT

```

## 推理模式选择 ( 仅限 Nova 2.0 )

Amazon Nova 2.0 支持推理模式，可进一步提升分析能力：

- 推理模式 ( 已启用 ) :
  - 在训练配置中设置 `reasoning_enabled: true`
  - 模型训练可在给出最终答案之前生成推理轨迹

- 在复杂推理任务中获得更优性能
- 非推理模式 ( 已禁用 ) :
  - 设置 `reasoning_enabled: false` 或省略参数 ( 默认 )
  - 无显式推理的标准 SFT
  - 适用于无需分步推理即可完成的任务

#### Note

- 启用推理后，模型会进行高强度推理。SFT 不提供低推理模式选项。
- 不支持 SFT 多模态推理内容。推理模式适用于纯文本输入。

### 对非推理数据集使用推理模式

您可在 `reasoning_enabled: true` 的情况下，使用非推理数据集训练 Amazon Nova。但是，这样做可能导致模型丧失推理能力，因为 Amazon Nova 会主要学习直接生成数据中的应答，而不进行推理。

如果使用非推理数据集训练 Amazon Nova，但仍想在推理过程中使用推理模式：

1. 在训练期间禁用推理模式 (`reasoning_enabled: false`)
2. 稍后在推理过程中启用推理模式

虽然这种方法可在推理阶段启用推理模式，但相较于不使用推理模式的推理过程，无法保证获得更优效果。

最佳实践：在使用推理数据集时，应同时为训练和推理过程启用推理模式；在使用非推理数据集时，应同时为两者禁用推理模式。

#### Note

有关适用的容器映像或示例配方的更多信息，请访问 [Amazon Nova 配方](#)。

### 工具调用数据格式

SFT 支持训练模型使用工具 ( 函数调用 )。以下是用于调用工具的示例输入格式：

## 示例输入：

```
{
  "schemaVersion": "bedrock-conversation-2024",
  "system": [
    {
      "text": "You are an expert in composing function calls."
    }
  ],
  "toolConfig": {
    "tools": [
      {
        "toolSpec": {
          "name": "getItemCost",
          "description": "Retrieve the cost of an item from the catalog",
          "inputSchema": {
            "json": {
              "type": "object",
              "properties": {
                "item_name": {
                  "type": "string",
                  "description": "The name of the item to retrieve cost for"
                },
                "item_id": {
                  "type": "string",
                  "description": "The ASIN of item to retrieve cost for"
                }
              }
            },
            "required": [
              "item_id"
            ]
          }
        }
      },
      {
        "toolSpec": {
          "name": "getItemAvailability",
          "description": "Retrieve whether an item is available in a given location",
          "inputSchema": {
            "json": {
              "type": "object",
              "properties": {
                "zipcode": {
```

```

        "type": "string",
        "description": "The zipcode of the location to check in"
    },
    "quantity": {
        "type": "integer",
        "description": "The number of items to check availability for"
    },
    "item_id": {
        "type": "string",
        "description": "The ASIN of item to check availability for"
    }
},
"required": [
    "item_id", "zipcode"
]
}
}
}
],
},
"messages": [
    {
        "role": "user",
        "content": [
            {
                "text": "I need to check whether there are twenty pieces of the following
item available. Here is the item ASIN on Amazon: id-123. Please check for the zipcode
94086"
            }
        ]
    },
    {
        "role": "assistant",
        "content": [
            {
                "reasoningContent": {
                    "reasoningText": {
                        "text": "The user wants to check how many pieces of the item with ASIN
id-123 are available in the zipcode 94086"
                    }
                }
            }
        ],
    }
]

```

```

        "toolUse": {
          "toolUseId": "getItemAvailability_0",
          "name": "getItemAvailability",
          "input": {
            "zipcode": "94086",
            "quantity": 20,
            "item_id": "id-123"
          }
        }
      ],
    },
    {
      "role": "user",
      "content": [
        {
          "toolResult": {
            "toolUseId": "getItemAvailability_0",
            "content": [
              {
                "text": "[{"name": \"getItemAvailability\", \"results\": {\"availability\": true}}]"
              }
            ]
          }
        }
      ]
    },
    {
      "role": "assistant",
      "content": [
        {
          "text": "Yes, there are twenty pieces of item id-123 available at 94086. Would you like to place an order or know the total cost?"
        }
      ]
    }
  ]
}

```

工具调用数据的重要注意事项：

- ToolUse 只能出现在助手轮次中

- ToolResult 只能出现在用户轮次中
- ToolResult 只能是文本或 JSON；Amazon Nova 模型目前不支持其他模式
- toolSpec 中的 inputSchema 必须是有效的 JSON 架构对象
- 每个 ToolResult 必须引用前序助手轮次 ToolUse 中的有效 toolUseId，且每个 toolUseId 在单次对话中仅可使用一次

### Note

有关适用的容器映像或示例配方的更多信息，请访问 [Amazon Nova 配方](#)。

## 文档理解数据格式

SFT 支持通过文档理解任务来训练模型。以下是示例输入格式：

### 示例输入

```
{
  "schemaVersion": "bedrock-conversation-2024",
  "messages": [
    {
      "role": "user",
      "content": [
        {
          "text": "What are the ways in which a customer can experience issues during checkout on Amazon?"
        },
        {
          "document": {
            "format": "pdf",
            "source": {
              "s3Location": {
                "uri": "s3://my-bucket-name/path/to/documents/customer_service_debugging.pdf",
                "bucketOwner": "123456789012"
              }
            }
          }
        }
      ]
    }
  ],
},
```

```

{
  "role": "assistant",
  "content": [
    {
      "text": "Customers can experience issues with 1. Data entry, 2. Payment methods, 3. Connectivity while placing the order. Which one would you like to dive into?"
    }
  ],
  "reasoning_content": [
    {
      "text": "I need to find the relevant section in the document to answer the question.",
      "type": "text"
    }
  ]
}

```

文档理解的重要注意事项：

- 仅支持 PDF 文件
- 最大文档为 10 MB
- 单个样本可包含文档与文本，但不可将文档与其他模态（如图像、视频）混用

#### Note

有关适用的容器映像或示例配方的更多信息，请访问 [Amazon Nova 配方](#)。

## SFT 视频理解

SFT 支持针对视频理解任务微调模型。以下是示例输入格式：

示例输入

```

{
  "schemaVersion": "bedrock-conversation-2024",
  "messages": [
    {

```

```

    "role": "user",
    "content": [
      {
        "text": "What are the ways in which a customer can experience issues during
checkout on Amazon?"
      },
      {
        "video": {
          "format": "mp4",
          "source": {
            "s3Location": {
              "uri": "s3://my-bucket-name/path/to/videos/
customer_service_debugging.mp4",
              "bucketOwner": "123456789012"
            }
          }
        }
      }
    ],
    {
      "role": "assistant",
      "content": [
        {
          "text": "Customers can experience issues with 1. Data entry, 2. Payment
methods, 3. Connectivity while placing the order. Which one would you like to dive
into?"
        }
      ],
      "reasoning_content": [
        {
          "text": "I need to find the relevant section in the video to answer the
question.",
          "type": "text"
        }
      ]
    }
  ]
}

```

### 视频理解的重要注意事项：

- 视频大小最大 50 MB

- 视频时长最长 15 分钟
- 每个样本仅允许包含一个视频；不支持在同一样本中使用多个视频
- 单个样本可包含视频与文本，但不可将视频与其他模态（如图像、文档）混用

### Note

有关适用的容器映像或示例配方的更多信息，请访问 [Amazon Nova 配方](#)。

## 数据上传说明

将训练数据集与验证数据集上传到 S3 存储桶。在配方的 run 数据块中指定以下路径：

```
## Run config
run:
  ...
  data_s3_path: "s3://<bucket-name>/<training-directory>/<training-file>.jsonl"
```

注意：将 <bucket-name>、<training-directory>、<validation-directory>、<training-file> 和 <validation-file> 替换为实际的 S3 路径。

注意：Amazon Nova 2.0 SFT 目前不支持验证数据集。如果传入验证数据集，系统将忽略该数据集。

## 创建微调作业

使用 run 数据块中的 model\_type 和 model\_name\_or\_path 字段定义基础模型：

```
## Run config
run:
  ...
  model_type: amazon.nova-2-lite-v1:0:256k
  model_name_or_path: nova-lite-2/prod
  ...
```

## SFT 调优参数

可用于 SFT 调优的参数包括：

## 运行配置

- `name` : 训练作业的描述性名称。这有助于在 AWS 管理控制台中识别对应作业。
- `model_type` : 要使用的 Amazon Nova 模型变体。可用选项为 `amazon.nova-2-lite-v1:0:256k`。
- `model_name_or_path` : 用于训练的基本模型的路径。可用选项为 `nova-lite-2/prod` , 或训练后检查点的 S3 路径 (`s3://customer-escrow-bucket-unique_id/training_run_name`)。
- `replicas` : 要在分布式训练中使用的计算实例数。可用值因您所选的模型而异。Amazon Nova Lite 2.0 支持 4、8、16 或 32 个副本。
- `data_s3_path` : 训练数据集的 S3 位置, 格式为 JSONL 文件。此文件必须与集群位于同一 AWS 账户和区域中。提供的所有 S3 位置必须位于同一账户和区域中。
- `validation_data_s3_path` : ( 选填 ) 验证数据集的 S3 位置, 格式为 JSONL 文件。此文件必须与集群位于相同的账户和区域中。提供的所有 S3 位置必须位于同一账户和区域中。
- `output_s3_path` : 存储清单和 TensorBoard 日志的 S3 位置。提供的所有 S3 位置必须位于同一 AWS 账户和 AWS 区域中。
- `mlflow_tracking_uri` : 用于 MLflow 日志记录的 MLflow 应用程序的 ARN。
- `mlflow_experiment_name` : MLflow 实验名称。
- `mlflow_run_name` : MLflow 运行名称。

## 训练配置

- `max_steps` : 训练迭代步数。每一步训练模型所用的元素数量即 `global_batch_size`。
- `save_steps` : 训练期间保存模型检查点的频率 ( 以步为单位 ) 。
- `save_top_k` : 根据验证指标保留的最佳检查点的最大数量。
- `max_length` : 以词元为单位的最大序列长度。这决定了训练的上下文窗口大小。SFT 支持的最大值为 32768 个词元。

更长的序列将会提高训练效率, 但会以增加内存需求为代价。建议将 `max_length` 参数设置为与数据分布相匹配。

- `global_batch_size` : 所有设备与 Worker 节点在一次前向及反向传播中共同处理的训练样本总数。

该值乘以每台设备的批量大小和设备数量。它会影响训练的稳定性和吞吐量。我们建议从适合您内存的批量大小开始, 然后进行扩展。对于特定领域的的数据, 较大的批量大小可能会使梯度过于平滑。

- `reasoning_enabled` : 布尔标志，用于在训练期间启用推理功能。

## 学习率调度器

- `warmup_steps` : 逐步提高学习率的步数。这可以提高训练稳定性。
- `min_lr` : 衰减结束时的最低学习率。有效值介于 0 到 1 之间（含两端值），但必须小于学习率。

## 优化器配置

- `lr` : 学习率，控制优化期间的步长。为了获得良好的性能，我们建议使用介于  $1e-6$  和  $1e-4$  之间的值。有效值介于 0 到 1 之间（含两端值）。
- `weight_decay` : L2 正则化强度。值比较高（介于 0.01 到 0.1 之间）会增加正则化强度。
- `adam_beta1` : Adam 优化器中一阶矩估计的指数衰减率。默认值为 0.9。
- `adam_beta2` : Adam 优化器中二阶矩估计的指数衰减率。默认值为 0.95。

## PEFT 配置

- `peft_scheme` : 要使用的参数高效微调方案。全秩微调的可选值为 `'null'`；基于 LoRA 的微调可选值为 `lora`。

## LoRA 微调（当 `peft_scheme` 为 `lora` 时）

- `alpha` : LoRA 缩放参数。控制低秩适应的幅度。典型值范围为 8 到 128。
- `lora_plus_lr_ratio` : LoRA+ 优化中的学习率比例。该乘数会专门针对 LoRA 参数调整学习率。

## 超参数指导

根据训练方法，建议采用如下超参数：

### 全秩训练

- `epoch` : 1
- 学习率 (`lr`) :  $1e-5$

- 最低学习率 ( min\_lr ) :  $1e - 6$

## LoRA ( 低秩适应 )

- epoch : 2
- 学习率 ( lr ) :  $5e - 5$
- 最低学习率 ( min\_lr ) :  $1e - 6$

注意：请根据数据集规模和验证性能调整这些值。监控训练指标以防止过拟合。

## SageMaker HyperPod 上的强化微调 ( RFT )

强化微调 ( RFT ) 是一种机器学习技术，它通过反馈信号 ( 即可衡量响应质量的分数或奖励 ) 提升模型性能，而非借助精确标准答案进行直接监督。与从输入-输出对中学习的传统监督式微调不同，RFT 使用奖励函数评测模型响应，并迭代优化模型以最大化奖励。

这种方法尤其适用于难以定义精确输出但能可靠衡量响应质量的任务。RFT 使模型能够通过尝试和反馈来学习复杂的行为与偏好，因此非常适合需要精细决策、创造性解决问题，或遵守可通过编程方式评测的特定质量标准等应用场景。

### 何时使用 RFT

当您能够定义清晰、可衡量的成功标准，但难以提供精确的正确输出用于训练时，便可使用 RFT。该方法适用于质量具有主观性或多维度的任务，例如创意写作、代码优化或复杂推理。这类任务存在多种可行方案，但部分方案明显更优。

RFT 在以下条件下效果最佳：

- 具备可通过程序评测模型输出的可靠奖励函数
- 需要使模型行为与特定偏好或约束保持一致
- 传统监督式微调因高质量标注数据采集成本高昂或不切实际而难以奏效的场景

应用需要迭代优化、个性化，或需遵循可编码为奖励信号的复杂业务规则时，可考虑采用 RFT。

### RFT 最适合哪些场景

在输出质量可客观衡量、但最优应答难以预先定义的领域，RFT 表现优异：

- 数学问题求解：存在多种求解路径，但正确性可验证

- 代码生成和优化：执行结果与性能指标可测试
- 科学推理任务：逻辑一致性与事实准确性
- 结构化数据分析：可通过编程方式验证输出
- 多步推理：需逐步逻辑推演的任务
- 工具使用和 API 调用：成功与否可通过执行结果来衡量
- 复杂的工作流程：遵守特定约束和业务规则

当您需要平衡多个相互竞争的目标（如准确性、效率和风格）时，RFT 效果尤为出色。

何时在 RFT 训练中使用推理模式

Amazon Nova 2.0 支持在 RFT 训练期间使用推理模式。可用模式如下：

- none：无推理（省略 reasoning\_effort 字段）
- low：最小推理开销
- high：最大推理能力（指定了 reasoning\_effort 时，此为默认值）

#### Note

RFT 无中等选项。如果配置中不含 reasoning\_effort 字段，则禁用推理。

在以下场景使用高强度推理：

- 复杂分析任务
- 数学问题求解
- 多步逻辑推导
- 逐步思考能带来价值的任务

在以下场景使用无（省略 reasoning\_effort）或低强度推理：

- 简单事实查询
- 直接分类
- 速度与成本优化
- 直接问答

### Important

较高强度的推理模式会增加训练时间和成本、推理延迟和成本，但同时也会提升模型在复杂推理任务方面的能力。

## 支持的模型

SageMaker HyperPod 上的 RFT 支持 Amazon Nova Lite 2.0 ( amazon.nova-2-lite-v1:0:256k )。

## 主要步骤

RFT 过程包括四个关键阶段：

- 实现评测器：创建奖励函数，根据质量标准以编程方式对模型响应进行评分。
- 上传提示：准备并上传指定对话格式的训练数据，同时提供用于评测的参考数据。
- 开始作业：使用配置的参数启动强化微调过程。
- 监控：通过指标控制面板跟踪训练进度，确保模型有效学习。

每个步骤都建立在前一步的基础上，其中评测器是指导整个训练过程的基础，它通过提供一致的反馈信号来引导模型学习。

## 主题

- [Nova 2.0 上的 RFT](#)

## Nova 2.0 上的 RFT

RFT 训练数据遵循 OpenAI 对话格式。每个训练样本都是一个 JSON 对象，包含消息、参考答案及可选的工具定义。本节提供相关的指导说明，帮助您了解如何为 Nova 2.0 上的 RFT 准备有效训练数据。

## 主题

- [数据格式与结构](#)
- [字段描述](#)
- [超参数指导](#)
- [其它属性](#)

- [数据集规模建议](#)
- [有效训练数据的特征](#)
- [监控 RFT 训练](#)

## 数据格式与结构

每个训练样本都是一个 JSON 对象，其中包含：

- messages：使用系统、用户和可选的助手角色进行的对话轮次数组
- reference\_answer：用于奖励计算的预期输出或评测标准
- tools ( 选填 )：可供模型使用的函数定义数组
- id ( 选填 )：用于跟踪和去重的唯一标识符

每个样本需单独占一行，JSONL 文件中每行对应一个 JSON 对象。

### 示例 1：化学问题

以下示例是一个化学问题，其参考答案包含真实值：

```
{
  "id": "chem-001",
  "messages": [
    {
      "role": "system",
      "content": "You are a helpful chemistry assistant"
    },
    {
      "role": "user",
      "content": "Predict hydrogen bond donors and acceptors for this SMILES:
CCN(CC)CCC(=O)c1sc(N)nc1C"
    }
  ],
  "reference_answer": {
    "donor_bond_counts": 2,
    "acceptor_bond_counts": 4,
    "explanation": "Calculated using Lipinski's rule of five: N-H groups (2 donors), N
and O atoms with lone pairs (4 acceptors)"
  }
}
```

**Note**

reference\_answer 包含使用领域特定规则计算得出的真实值。奖励函数会将模型的预测值与这些参考值进行比较，以计算奖励分数。

**示例 2：数学问题**

以下示例是一个包含求解步骤的数学问题：

```
{
  "id": "math-001",
  "messages": [
    {
      "role": "system",
      "content": "You are a math tutor"
    },
    {
      "role": "user",
      "content": "Solve:  $2x + 5 = 13$ "
    }
  ],
  "reference_answer": {
    "solution": "x = 4",
    "steps": [" $2x = 13 - 5$ ", " $2x = 8$ ", " $x = 4$ "]
  }
}
```

**示例 3：工具使用**

以下示例显示了工具使用及预期行为：

```
{
  "id": "tool-001",
  "messages": [
    {
      "role": "system",
      "content": "You are a helpful game master assistant"
    },
    {
      "role": "user",
```

```

    "content": "Generate a strength stat for a warrior character. Apply a +2 racial
bonus modifier."
  }
],
"tools": [
  {
    "type": "function",
    "function": {
      "name": "StatRollAPI",
      "description": "Generates character stats by rolling 4d6, dropping the lowest
die result, and applying a modifier.",
      "parameters": {
        "type": "object",
        "properties": {
          "modifier": {
            "description": "An integer representing the modifier to apply to the
total of the stat roll.",
            "type": "integer"
          }
        }
      },
      "required": ["modifier"]
    }
  }
],
"reference_answer": {
  "tool_called": "StatRollAPI",
  "tool_parameters": {
    "modifier": 2
  },
  "expected_behavior": "Call StatRollAPI with modifier=2 and return the calculated
stat value"
}
}

```

## 字段描述

| 字段 | 说明             | 附加说明                             | 必需 |
|----|----------------|----------------------------------|----|
| id | 此 RFT 样本的唯一标识符 | 字符串 ( 例如 sample-001 ) : 用于追踪与去重。 | 否  |

| 字段                 | 说明                | 附加说明                                          | 必需 |
|--------------------|-------------------|-----------------------------------------------|----|
| 消息                 | 定义提示和上下文的聊天消息有序列表 | 对象数组：模型按顺序查看列表。通常而言，列表以系统消息开头，然后是用户。          | 是  |
| messages[].role    | 消息发送方             | 常用值：system、user（其他上下文中也可能出现assistant）         | 否  |
| messages[].content | 消息文本内容            | 纯字符串：对系统而言，此为指令；对用户而言，此为任务或输入。                | 否  |
| 工具                 | 在本示例中模型可用的工具规范    | 数组：每个项目均定义了工具的界面与元数据。类型可能包括function或internal。 | 否  |
| reference_answer   | 本示例的预期模型输出        | 字符串或对象（取决于任务）：作为评测或培训所用的目标。                   | 否  |

### Note

任何其他自定义字段（例如 task\_id、difficulty\_level、context\_data）均未经过验证，将作为元数据传递给奖励函数。

## 超参数指导

根据训练方法，建议采用如下超参数：

常规：

- epoch : 1

- 学习率 ( lr ) :  $1e - 7$
- 生成次数 : 8
- 最大新词元数 : 8192
- 批处理大小 : 256

LoRA ( 低秩适应 ) :

- LoRA 秩 : 32

#### Note

请根据数据集规模和验证性能调整这些值。监控训练指标以防止过拟合。

#### 其它属性

将 `additionalProperties` 设置为 `true` 后，即可在核心架构要求之外添加自定义字段，从而灵活提供奖励函数正常评测所需的任意数据。

#### 常见附加字段

您可以包含以下类型的附加字段：

元数据：

- `task_id`：用于跟踪的唯一标识符
- `difficulty_level`：问题复杂度指示器
- `domain`：主题领域或类别
- `expected_reasoning_steps`：解题过程中的步骤数

评测标准：

- `evaluation_criteria`：具体评分规则
- `custom_scoring_weights`：不同维度的相对权重
- `context_data`：问题背景信息
- `external_references`：相关文档或资源链接

## 包含其他属性的示例

以下示例包含自定义元数据字段：

```
{
  "id": "algebra_001",
  "messages": [
    {
      "role": "system",
      "content": "You are a math tutor"
    },
    {
      "role": "user",
      "content": "Solve:  $2x + 5 = 13$ "
    }
  ],
  "reference_answer": {
    "solution": "x = 4",
    "steps": [" $2x = 13 - 5$ ", " $2x = 8$ ", " $x = 4$ "]
  },
  "task_id": "algebra_001",
  "difficulty_level": "easy",
  "domain": "algebra",
  "expected_reasoning_steps": 3
}
```

## 数据集规模建议

### 起步要求

建议采用以下最小数据集规模：

- 至少 100 个训练样本
- 至少 100 个评测样本

优先使用高质量的输入数据，以及能够在模型响应上一致执行的可靠奖励函数。

### 评测优先的方法

在投入大规模 RFT 训练之前，请先评测模型的基线性能：

- 性能优异（奖励大于 95%）：无需进行 RFT，模型表现足够好

- 性能极差 ( 奖励为 0% ) : 先切换到 SFT , 建立基本能力
- 性能中等 : 适合采用 RFT 训练

这种“评测优先”的方法可确保奖励函数无缺陷，并判断 RFT 是否适用于自己的使用案例。从小规模数据起步，有助于您熟悉 RFT 工作流程，尽早发现并解决问题，在扩容前验证方案，并测试奖励函数的可靠性。验证通过后，即可扩展到更大规模的数据集，进一步提升模型性能。

### 有效训练数据的特征

#### 清晰性与一致性

优质的 RFT 样本需要清晰、无歧义的输入数据，以便能够针对不同的模型输出进行准确的奖励计算。数据中应避免以下干扰：

- 格式不一致
- 标签或指令相互矛盾
- 提示词含义模糊
- 参考答案互相冲突

任何歧义都会误导训练过程，导致模型学习到非预期的行为。

#### 多样性

数据集应覆盖生产环境中的各类使用案例，确保模型在真实环境中表现稳健。包含：

- 各种问题类型与难度级别
- 不同的输入格式和边界情况
- 来自所有预期场景的代表性样本

这种多样性有助于防止过拟合，并确保模型能够妥善处理不熟悉的输入。

#### 奖励函数注意事项

设计奖励函数，实现高效训练：

- 在几秒钟内执行完毕 ( 而非几分钟 )
- 使用 Lambda 实现高效并行
- 返回一致且可靠的分数

- 妥善处理不同类型的模型输出

快速且可扩展的奖励函数可支持快速迭代，并在大规模实验中实现高成本效益。

## 监控 RFT 训练

在训练期间监控关键指标，确保有效学习，尽早发现潜在问题。

### 主题

- [要跟踪的关键指标](#)
- [RFT 之后的评测](#)
- [使用经过微调的模型](#)
- [限制与最佳实践](#)
- [问题排查](#)

### 要跟踪的关键指标

在训练期间使用 MLflow 监控以下指标：

#### 奖励指标：

- 平均奖励分数：模型响应的整体质量（应随时间增加）
- 奖励分布：获得高、中、低奖励的响应百分比
- 训练奖励与验证奖励：通过对比检测过拟合

#### 训练指标：

- 策略更新次数：成功更新权重的次数
- 推演完成率：成功完成评测的样本占比

#### 异常模式：

- 奖励趋于平稳（表明学习效果差）
- 训练奖励上升，而验证奖励下降（过拟合）
- 奖励方差随时间显著增大（不稳定）
- 奖励函数错误率偏高（实现问题）

何时停止训练：

- 已达到目标性能指标
- 奖励趋于平稳且不再提升
- 验证性能下降 ( 检测到过拟合 )
- 已达到最大训练预算

RFT 之后的评测

训练完成后，评测经过微调的模型以判断性能提升情况：

- 运行 RFT 评测作业：使用 RFT 训练产出的检查点作为模型
- 与基线对比：在相同的测试集上评测基础模型和经过微调的模型
- 分析指标：查看任务特定指标 ( 准确率、奖励分数等 )
- 进行定性审查：人工抽查样本输出质量

有关详细的评测流程，请参阅“评测”一节。

使用经过微调的模型

访问检查点：

训练完成后，找到检查点：

1. 进入 S3 中的 `output_path`
2. 下载并解压缩 `output.tar.gz`
3. 打开 `manifest.json`
4. 复制 `checkpoint_s3_bucket` 值

部署用于推理：

使用检查点 S3 路径进行推理或进一步训练：

```
run:
  model_type: amazon.nova-2-lite-v1:0:256k
  model_name_or_path: "s3://customer-escrow-<account-number>-smtj-<unique-
  identifier>/<job-name>"
```

有关部署和推理的说明，请参阅“推理”一节。

## 限制与最佳实践

目前的局限性：

Beta 限制：

- 需要为 RFT 创建新的 RIG 组。此限制将在正式发布时解决。
- 实例类型要求：仅支持 P5 实例（至少需要 8x P5.48xlarge）。即将推出：对更小实例类型的支持（预计 2025 年 1 月中旬）。

功能限制：

- 15 分钟 Lambda 超时：奖励函数必须在 15 分钟内完成执行
- 仅支持单轮对话：不支持多轮对话
- 验证数据集：训练期间不支持使用。请使用单独的评测作业来评测训练进度。

训练注意事项：

- 低奖励场景：当只有不到 5% 的示例获得正奖励时，训练可能难以进行，此时建议先使用 SFT
- 数据要求：需具备足够多样性，才能保证学习效果
- 计算成本：比监督式微调更昂贵

Nova Forge 消除了以下部分限制：

- 支持多轮对话
- 允许奖励函数超过 15 分钟超时
- 提供高级算法和调优选项
- 专为复杂企业使用案例设计，专门针对构建前沿模型进行了调优

最佳实践：

从小规模开始逐步扩展：

- 从最小规模数据集（100 – 200 个样本）和少量训练 epoch 开始

- 在扩大规模之前验证方法
- 根据结果逐步增加数据集规模和训练步数

先用 SFT 建立基线：

- 如果奖励分数持续偏低（例如始终为 0），应先执行 SFT，再进行 RFT
- RFT 需要具备合理的基线性能，才能实现有效优化

设计高效的奖励函数：

- 在几秒钟内执行完毕（而非几分钟）
- 尽量减少外部 API 调用
- 使用高效的算法与数据结构
- 实现完善的错误处理机制
- 训练前进行充分测试
- 充分利用 Lambda 的并行扩展能力

主动监控训练：

- 跟踪平均奖励分数随时间的变化
- 观察各样本之间的奖励分布
- 对比训练奖励与验证奖励
- 留意异常模式（平稳、过拟合、不稳定）

根据结果进行迭代：

- 如果多次迭代后奖励仍无提升，请调整奖励函数的设计
- 提升数据集的多样性以提供更清晰的学习信号
- 如果奖励仍接近零，建议改用 SFT
- 尝试不同的超参数（学习率、批次大小）

优化数据质量：

- 确保样本多样且具有代表性

- 包含边界情况和困难样本
- 验证奖励函数能够对各类样本进行正确评分
- 移除或修复会干扰奖励函数的样本

## 问题排查

### 奖励函数错误：

症状：训练期间奖励函数调用的错误率较高

| 问题        | 症状          | 解决方案                                             |
|-----------|-------------|--------------------------------------------------|
| Lambda 超时 | 15 分钟后频繁超时  | 优化函数性能；对于复杂评测，建议使用 Nova Forge                    |
| 并发不足      | Lambda 节流错误 | 提高 <code>lambda_concurrency_limit</code> 或申请提高配额 |
| 返回格式无效    | 训练失败并出现格式错误 | 验证返回结构是否符合要求的接口规范                                |
| 未处理的异常    | 间歇性错误       | 添加完善的错误处理与日志记录                                   |
| 外部 API 故障 | 评分不一致       | 实现重试逻辑和备用策略                                      |

### 训练表现不佳：

症状：奖励没有提升或停留在较低水平

### 解决方法：

- 验证奖励函数正确性：使用已知的优/劣样本进行测试
- 检查基线性能：评测基础模型；如果准确率接近零，建议先进行 SFT
- 提升数据多样性：添加更多覆盖不同场景的多样化样本
- 调整超参数：尝试不同学习率或批次大小
- 检查奖励信号质量：确保奖励可区分优劣响应

过拟合：

症状：训练奖励上升，而验证奖励下降

解决方法：

- 减少训练步数：提前停止训练
- 扩大数据集规模：增加训练样本数量
- 添加正则化：调整 `weight_decay` 或 `entropy_coeff`
- 提升数据多样性：确保训练集代表完整的数据分布

## 评估经过训练的模型

评估配方是一个定义如何执行 Amazon Nova 模型评测任务的 YAML 配置文件。使用此配方，您可以根据常用基准或自有的自定义数据集评估基础模型或训练模型的性能。性能指标可能存储在 Amazon S3 或 TensorBoard 中。评估提供定量指标来帮助您评估模型在各种任务中的性能，从而确定是否需要进一步自定义。

模型评测是一个离线过程，模型使用预定义的答案，按照固定的基准进行测试。模型不是实时评估的，也不是按照用户实时互动进行评估。对于实时评估，您可以在模型部署到 Amazon Bedrock 之后通过调用 Amazon Bedrock 运行时 API 进行评估。

主题

- [可用的基准测试任务](#)
- [了解配方参数](#)
- [评估配方示例](#)
- [启动评估作业](#)
- [访问和分析评估结果](#)
- [RFT 评测](#)

### 可用的基准测试任务

一个可用的示例代码包，用于演示如何使用适用于 Amazon Nova 的 SageMaker AI 模型评测特征计算基准评测指标。要获取代码包，请参阅 [sample-Nova-lighteval-custom-task](#)。

以下是支持的可用行业标准基准测试列表。您可以在 `eval_task` 参数中指定以下基准测试：

| 基准            | 模式 | 说明                                      | 指标          | Strategy | 可用子任务 |
|---------------|----|-----------------------------------------|-------------|----------|-------|
| mmlu          | 文本 | 多任务语言理解：考核 57 个科目的知识。                   | 准确性         | zs_cot   | 是     |
| mmlu_pro      | 文本 | MMLU (专业子集)，专注于法律、医学、会计和工程等专业领域。        | 准确性         | zs_cot   | 否     |
| bbh           | 文本 | 高级推理任务：一系列具有挑战性的问题，用于考核更高级别的认知和解决问题的能力。 | 准确性         | zs_cot   | 是     |
| gpqa          | 文本 | 一般物理问题解答：评测对物理概念和相关问题解决能力的理解情况。         | 准确性         | zs_cot   | 否     |
| math          | 文本 | 数学问题解决：衡量在代数、微积分及应用题等领域的数学推理能力。         | exact_match | zs_cot   | 是     |
| strong_reject | 文本 | 质量控制任务：测试模型                             | deflection  | zs       | 是     |

| 基准        | 模式 | 说明                                                                                     | 指标  | Strategy | 可用子任务 |
|-----------|----|----------------------------------------------------------------------------------------|-----|----------|-------|
|           |    | 检测和拒绝不当、有害或不正确内容的的能力。                                                                  |     |          |       |
| IFEval    | 文本 | 指令跟随评估：衡量模型遵循给定指令并按照规范完成任务的准确程度。                                                       | 准确性 | zs       | 否     |
| gen_qa    | 文本 | 自定义数据集评估：让您可以自带数据集进行基准测试，将模型输出与参考答案及 ROUGE 和 BLEU 等指标进行比较。                             | 全部  | gen_qa   | 否     |
| llm_judge | 文本 | LLM-as-a-judge 偏好比较：使用 Amazon Nova Judge 模型来确定提示的成对响应之间的偏好（B 与 A 比较），计算 B 比 A 更受欢迎的概率。 | 全部  | judge    | 否     |

| 基准               | 模式         | 说明                                                                                                                                                   | 指标     | Strategy | 可用子任务 |
|------------------|------------|------------------------------------------------------------------------------------------------------------------------------------------------------|--------|----------|-------|
| humaneval        | 文本         | HumanEval : 一个基准测试数据集，旨在评估大型语言模型的代码生成能力                                                                                                              | pass@1 | zs       | 否     |
| mm_llm_judge     | 多模态 ( 图像 ) | 该全新基准评测的运行方式与上述基于文本的 llm_judge 一致。唯一区别在于其支持图像推理。                                                                                                     | 全部     | judge    | 否     |
| rubric_llm_judge | Text       | Rubric Judge 是基于 Amazon Nova 2.0 Lite 构建的增强版 LLM-as-a-judge 评测模型。与仅提供偏好判定的 <a href="#">原始评判模型</a> 不同，Rubric Judge 可针对每个提示动态生成自定义评估准则，并在多个维度上给出细粒度评分。 | 全部     | judge    | 否     |

| 基准                  | 模式   | 说明                                                                                              | 指标          | Strategy | 可用子任务 |
|---------------------|------|-------------------------------------------------------------------------------------------------|-------------|----------|-------|
| aime_2024           | Text | AIME 2024 : 2024 年美国数学邀请赛 ( American Invitational Mathematics Examination ) , 用于考查高阶数学推理与问题求解能力 | exact_match | zs_cot   | No    |
| calendar_scheduling | Text | Natural Plan : 日程规划任务 , 用于评估跨日期、跨人员的会议安排等规划能力                                                   | exact_match | fs       | No    |

以下 mmlu 子任务可用 :

```
MMLU_SUBTASKS = [
    "abstract_algebra",
    "anatomy",
    "astronomy",
    "business_ethics",
    "clinical_knowledge",
    "college_biology",
    "college_chemistry",
    "college_computer_science",
    "college_mathematics",
    "college_medicine",
    "college_physics",
    "computer_security",
    "conceptual_physics",
    "econometrics",
```

```
"electrical_engineering",  
"elementary_mathematics",  
"formal_logic",  
"global_facts",  
"high_school_biology",  
"high_school_chemistry",  
"high_school_computer_science",  
"high_school_european_history",  
"high_school_geography",  
"high_school_government_and_politics",  
"high_school_macroeconomics",  
"high_school_mathematics",  
"high_school_microeconomics",  
"high_school_physics",  
"high_school_psychology",  
"high_school_statistics",  
"high_school_us_history",  
"high_school_world_history",  
"human_aging",  
"human_sexuality",  
"international_law",  
"jurisprudence",  
"logical_fallacies",  
"machine_learning",  
"management",  
"marketing",  
"medical_genetics",  
"miscellaneous",  
"moral_disputes",  
"moral_scenarios",  
"nutrition",  
"philosophy",  
"prehistory",  
"professional_accounting",  
"professional_law",  
"professional_medicine",  
"professional_psychology",  
"public_relations",  
"security_studies",  
"sociology",  
"us_foreign_policy",  
"virology",  
"world_religions"
```

```
]
```

以下 bbh 子任务可用：

```
BBH_SUBTASKS = [  
    "boolean_expressions",  
    "causal_judgement",  
    "date_understanding",  
    "disambiguation_qa",  
    "dyck_languages",  
    "formal_fallacies",  
    "geometric_shapes",  
    "hyperbaton",  
    "logical_deduction_five_objects",  
    "logical_deduction_seven_objects",  
    "logical_deduction_three_objects",  
    "movie_recommendation",  
    "multistep_arithmetic_two",  
    "navigate",  
    "object_counting",  
    "penguins_in_a_table",  
    "reasoning_about_colored_objects",  
    "ruin_names",  
    "salient_translation_error_detection",  
    "snarks",  
    "sports_understanding",  
    "temporal_sequences",  
    "tracking_shuffled_objects_five_objects",  
    "tracking_shuffled_objects_seven_objects",  
    "tracking_shuffled_objects_three_objects",  
    "web_of_lies",  
    "word_sorting"  
]
```

以下 math 子任务可用：

```
MATH_SUBTASKS = [  
    "algebra",  
    "counting_and_probability",  
    "geometry",  
    "intermediate_algebra",  
    "number_theory",  
    "prealgebra",  
]
```

```
"precalculus",
]
```

## 了解配方参数

### 运行配置

以下是常规运行配置和所涉及参数的说明。

```
run:
  name: eval_job_name
  model_type: amazon.nova-micro-v1:0:128k
  model_name_or_path: nova-micro/prod
  replicas: 1
  data_s3_path: ""
  output_s3_path: s3://output_path
  mlflow_tracking_uri: ""
  mlflow_experiment_name : ""
  mlflow_run_name : ""
```

- `name` : ( 必填 ) 评估作业的描述性名称。这有助于在 AWS 管理控制台中识别您的作业。
- `model_type` : ( 必填 ) 指定要使用的 Amazon Nova 模型变体。请勿手动修改此字段。选项包括 :
  - `amazon.nova-micro-v1:0:128k`
  - `amazon.nova-lite-v1:0:300k`
  - `amazon.nova-pro-v1:0:300k`
  - `amazon.nova-2-lite-v1:0:256k`
- `model_name_or_path` : ( 必填 ) 基本模型的路径或训练后检查点的 S3 路径。选项包括 :
  - `nova-micro/prod`
  - `nova-lite/prod`
  - `nova-pro/prod`
  - `nova-lite-2/prod`
  - ( 训练后检查点的 S3 路径 ) `s3://<escrow bucket>/<job id>/outputs/checkpoints`
- `replicas` : ( 必填 ) 要在分布式训练中使用的计算实例数。您必须将此值设置为 1，因为不支持多节点。
- `data_s3_path` : ( 必填 ) 输入数据集的 S3 路径。除非您使用自带数据集或 LLM-as-a-Judge 配方，否则请将此参数留空。

- `output_s3_path` : ( 必填 ) 存储输出评估构件的 S3 路径。请注意，输出 S3 存储桶必须由创建作业的账户创建。
- `mlflow_tracking_uri`: ( 可选 ) 用于跟踪 MLflow 运行/实验的 MLflow 跟踪服务器 ARN。确保您拥有通过 SageMaker AI 执行角色访问跟踪服务器的权限

## 评估配置

以下是模型评测配置和所涉及参数的说明。

```
evaluation:
  task: mmlu
  strategy: zs_cot
  subtask: mathematics
  metric: accuracy
```

- `task` : ( 必填 ) 指定要使用的评估基准测试或任务。

支持的任务列表：

- `mmlu`
- `mmlu_pro`
- `bbh`
- `gpqa`
- `math`
- `strong_reject`
- `gen_qa`
- `ifeval`
- `llm_judge`
- `humaneval`
- `mm_llm_judge`
- `rubric_llm_judge`
- `aime_2024`
- `calendar_scheduling`
- `humaneval`

- `strategy` : ( 必填 ) 定义评估方法：

- `zs_cot` : 零样本思维链，该提示方法激励大型语言模型逐步进行推理，而不提供明确的示例。
- `zs` : 零样本，该方法事先无需任何训练示例即可解决问题。
- `gen_qa` : 一种专门用于自带数据集配方的策略。
- `judge` : 一种专门用于 Amazon Nova LLM as judge 和 `mm_llm_judge` 的策略。
- `subtask` : ( 可选且可移除 ) 为某些评估任务指定具体的子任务。如果您的任务没有任何子任务，请将其从配方中移除。
- `metric` : ( 必填 ) 要使用的评估指标。
  - `accuracy` : 正确答案的百分比
  - `exact_match` : ( 对于 `math` 基准测试 )，返回输入预测字符串与其引用完全匹配的速率。
  - `deflection` : ( 对于 `strong reject` 基准测试 )，返回基本模型的相对偏差和差异显著性指标。
  - `pass@1` : ( 对于 `humaneval` 基准测试 )，该指标用于衡量模型的最高置信度预测与正确答案相匹配的情况所占的百分比。
  - `all` : 返回以下指标：
    - 对于 `gen_qa` 和自带数据集基准测试，返回以下指标：
      - `rouge1` : 衡量生成的文本和参考文本之间的一元 ( 单个词 ) 重叠度。
      - `rouge2` : 衡量生成的文本和参考文本之间的二元 ( 两个连续的词 ) 重叠度。
      - `rougeL` : 衡量文本之间的最长公共子序列，匹配中可以存在间隙。
      - `exact_match` : 二进制分数 ( 0 或 1 )，用于指示生成的文本是否与参考文本逐个字符完全匹配。
      - `quasi_exact_match` : 与精确匹配类似，但更加宽松，通常忽略大小写、标点符号和空格差异。
      - `f1_score` : 精度和查全率之间的调和平均数，衡量预测答案和参考答案之间的词语重叠度。
      - `f1_score_quasi` : 与 `f1_score` 类似，但匹配更为宽松，使用忽略细微差异的标准化文本比较。
      - `bleu` : 衡量生成的文本和参考文本之间 `n-gram` 匹配的精确度，通常用于翻译评估。
    - 对于 `llm_judge` 和 `mm_llm_judge`，自定义数据集基准，需返回以下指标：
      - `a_scores` : 前向和反向评估传递中 `response_A` 的获胜次数。
      - `a_scores_stderr` : 成对判断中 `response_A scores` 的标准误差。
      - `b_scores` : 前向和反向评估传递中 `response_B` 的获胜次数。
      - `b_scores_stderr` : 成对判断中 `response_B scores` 的标准误差。
    - `ties` : `response_A` 和 `response_B` 被评估为相等的判断数量。

- `ties_stderr` : 成对判断中平局的标准误差。
- `inference_error` : 无法适当评估的判断数量。
- `inference_error_stderr` : 判断中推理错误的标准误差。
- `score` : 基于 `response_B` 在前向和反向传递中的获胜次数得出的总分。
- `score_stderr` : 成对判断中总分的标准误差。
- `winrate` : 使用 Bradley-Terry 概率模型计算的 `response_B` 比 `response_A` 更受欢迎的概率。
- `lower_rate` : 自动采样预估胜率的下限 ( 第 2.5 百分位 ) 。

## 推理配置

以下是推理配置和所涉及参数的说明。所有参数都是可选的。

```
inference:
  max_new_tokens: 200
  top_k: -1
  top_p: 1.0
  temperature: 0
  top_logprobs: 10
  reasoning_effort: null # options: low/high to enable reasoning or null to disable reasoning
```

- `max_new_tokens` : 要生成的最大词元数。必须是整数。
- `top_k` : 要考虑的最高概率词元的数量。必须是整数。
- `top_p` : 词元采样的累积概率阈值。此值必须是介于 0.0 到 1.0 之间的浮点数 ( 含两端值 ) 。
- `temperature` : 词元选择的随机性。值越大，随机性越大。使用 0 以使结果具有确定性。此值必须是最小值为 0 的浮点数。
- `top_logprobs` : 推理响应中需返回的最高对数概率数量。取值必须为 0 到 20 之间的整数。对数概率包含消息内容中返回的输出词元及其对应的对数概率值。
- `reasoning_effort` : 控制具备推理能力模型的推理行为。仅当 `model_type` 指定了具备推理能力的模型 ( 当前为 `amazon.nova-2-lite-v1:0:256k` ) 时，才需设置 `reasoning_effort`。可用选项为 `null` ( 未设置时的默认值；表示禁用推理 )、`low` 或 `high`。

请注意，对于 `humaneval`，建议使用以下推理配置：

```
inference:
  top_k: 1
  max_new_tokens: 1600
  temperature: 0.0
```

## MLflow 配置

以下是 MLflow 配置和所涉及参数的说明。所有参数都是可选的。

```
run:
  mlflow_tracking_uri: ""
  mlflow_experiment_name: ""
  mlflow_run_name: ""
```

- `mlflow_tracking_uri`: ( 可选 ) MLflow 跟踪服务器地址 ( 仅在 SMHP 下需要 )
- `mlflow_experiment_name`: ( 可选 ) 实验名称, 用于对相关的机器学习运行记录进行分组
- `mlflow_run_name`: ( 可选 ) 实验中某次特定训练运行的自定义名称

## 评估配方示例

Amazon Nova 提供了四种类型的评估配方, 这些配方可在 SageMaker HyperPod 配方 GitHub 存储库中找到。

### 通用文本基准测试配方

这些配方让您能够通过一套全面的纯文本基准测试来评估 Amazon Nova 模型的基本功能。它们以 `xxx_general_text_benchmark_eval.yaml` 格式提供。

### 自带数据集基准测试配方

这些配方让您能够使用自己的数据集进行基准测试, 并使用不同类型的指标将模型输出与参考答案进行对比。它们以 `xxx_bring_your_own_dataset_eval.yaml` 格式提供。

以下是自带数据集的要求:

- 文件格式要求
  - 必须有一个包含评估示例的 `gen_qa.jsonl` 文件。
  - 您的数据集必须上传到 SageMaker 训练作业可以访问的 S3 位置。

- 该文件必须遵循常规问答数据集所需的架构格式。
- 架构格式要求：JSONL 文件中的每一行都必须是包含以下字段的 JSON 对象：
  - query：( 必填 ) 包含需要答案的问题或说明的字符串
  - response：( 必填 ) 包含预期模型输出的字符串
  - system：( 可选 ) 包含系统提示的字符串，用于在处理查询之前设置 AI 模型的行为、角色或个性
  - metadata：( 可选 ) 字符串类型，与本条记录关联的元数据，用于标签分类。

以下是自带数据集示例条目

```
{
  "system":"You are a english major with top marks in class who likes to give minimal
word responses: ",
  "query":"What is the symbol that ends the sentence as a question",
  "response":"?"
}
{
  "system":"You are a pattern analysis specialist that provides succinct answers: ",
  "query":"What is the next number in this series? 1, 2, 4, 8, 16, ?",
  "response":"32"
}
{
  "system":"You have great attention to detail that follows instructions accurately:
",
  "query":"Repeat only the last two words of the following: I ate a hamburger today
and it was kind of dry",
  "response":"of dry"
}
```

要使用您的自定义数据集，请按以下必填字段修改您的评估配方，且不要更改其中任何内容：

```
evaluation:
  task: gen_qa
  strategy: gen_qa
  metric: all
```

适用以下限制：

- 每次评估只支持一个 JSONL 文件。

- 该文件必须严格遵循定义的架构。
- 上下文长度限制：对于数据集中的每个样本，上下文长度（包括系统 + 查询提示）应小于 3.5k。

## Nova LLM-as-a-Judge 基准测试配方

Amazon Nova LLM-as-a-Judge 是一项模型评测特征，让客户可以使用自定义数据集，将一个模型的响应质量与基准模型的响应质量进行比较。其接受包含提示、基线响应和挑战响应的数据集，然后使用 Amazon Nova Judge 模型基于 [Bradley-Terry 概率](#) 和成对比较提供胜率指标。

配方以 `xxx_llm_judge_eval.yaml` 格式提供。

以下是 LLM-as-a-Judge 的要求：

- 文件格式要求
  - 有一个包含评估示例的 `llm_judge.jsonl` 文件。文件名称必须为 `llm_judge.jsonl`。
  - 您的数据集必须上传到 [SageMaker AI SageMaker HyperPod RIG](#) 可以访问的 S3 位置。
  - 该文件必须遵循 `llm_judge.jsonl` 数据集所需的架构格式。
  - 输入数据集应确保所有记录的上下文长度小于 12k。
- 架构格式要求：JSONL 文件中的每一行都必须是包含以下字段的 JSON 对象：
  - `prompt`：（必填）包含生成响应提示的字符串。
  - `response_A`：包含基准响应的字符串。
  - `response_B`：包含与基准响应进行比较的替代响应的字符串。

这是一个 LLM-as-a-Judge 示例条目

```
{
  "prompt": "What is the most effective way to combat climate change?",
  "response_A": "The most effective way to combat climate change is through a combination of transitioning to renewable energy sources and implementing strict carbon pricing policies. This creates economic incentives for businesses to reduce emissions while promoting clean energy adoption.",
  "response_B": "We should focus on renewable energy. Solar and wind power are good. People should drive electric cars. Companies need to pollute less."
}
{
  "prompt": "Explain how a computer's CPU works",
  "response_A": "CPU is like brain of computer. It does math and makes computer work fast. Has lots of tiny parts inside.",
}
```

```

"response_B": "A CPU (Central Processing Unit) functions through a fetch-execute
cycle, where instructions are retrieved from memory, decoded, and executed through its
arithmetic logic unit (ALU). It coordinates with cache memory and registers to process
data efficiently using binary operations."
}
{
"prompt": "How does photosynthesis work?",
"response_A": "Plants do photosynthesis to make food. They use sunlight and water. It
happens in leaves.",
"response_B": "Photosynthesis is a complex biochemical process where plants convert
light energy into chemical energy. They utilize chlorophyll to absorb sunlight,
combining CO2 and water to produce glucose and oxygen through a series of chemical
reactions in chloroplasts."
}

```

要使用您的自定义数据集，请按以下必填字段修改您的评估配方，且不要更改其中任何内容：

```

evaluation:
  task: llm_judge
  strategy: judge
  metric: all

```

适用以下限制：

- 每次评估只支持一个 JSONL 文件。
- 该文件必须严格遵循定义的架构。
- Amazon Nova Judge 模型对于所有模型系列规格（即 Lite、Micro 和 Pro）都是一样的。
- 目前不支持自定义评估模型。
- 上下文长度限制：对于数据集中的每个样本，上下文长度（包括系统 + 查询提示）应小于 7k。

使用 Nova LLM-as-a-judge，完成多模态（图像）基准测试配方

Nova LLM Judge 多模态（图像）版（简称 Amazon Nova MM\_LLM Judge）是一项模型评测功能，支持使用自定义数据集，将目标模型的响应质量与基线模型的响应进行对比评测。该功能接受包含提示、基线响应和挑战响应的数据集，以及 Base64 编码字符串图像，然后使用 Amazon Nova Judge 模型基于 [Bradley-Terry](#) 概率模型通过成对比较提供胜率指标。配方式：xxx\_mm\_llm\_judge\_eval.yaml。

Nova LLM 数据集要求

## 文件格式：

- 包含评估示例的单个 mm\_llm\_judge.jsonl 文件。文件名称必须严格为 llm\_judge.jsonl。
- 您必须将数据集上传到 SageMaker 训练作业可以访问的 S3 位置。
- 该文件必须遵循 mm\_llm\_judge 数据集所需的架构格式。
- 输入数据集应确保：排除 image 属性后，所有记录的上下文长度均小于 12k。

架构格式：.jsonl 文件中的每一行都必须是包含以下字段的 JSON 对象。

- 必填字段。

prompt：包含生成响应提示的字符串。

images：对象数组，每个对象包含 data 属性（值为 Base64 编码的图像字符串）。

response\_A：包含基准响应的字符串。

response\_B：包含与基准响应进行比较的替代响应的字符串。

## 示例条目

为便于阅读，以下示例包含换行与缩进，但在实际数据集中，每条记录必须单独占一行。

```
{
  "prompt": "what is in the image?",
  "images": [
    {
      "data": "data:image/jpeg;base64,/9j/2wBDAAQDAwQDAwQEAWQFBAQFBgo..."
    }
  ],
  "response_A": "a dog.",
  "response_B": "a cat.",
}
{
  "prompt": "how many animals in echo of the images?",
  "images": [
    {
      "data": "data:image/jpeg;base64,/9j/2wBDAAQDAwQDAwQEAWQFBAQFBgo..."
    },
    {
      "data": "data:image/jpeg;base64,/DKEafe3gihn..."
    }
  ]
}
```

```
    }
  ],
  "response_A": "The first image contains one cat and the second image contains one
dog",
  "response_B": "The first image has one animal and the second has one animal",
}
```

要使用您的自定义数据集，请按以下必填字段修改您的评估配方，且不要更改其中任何内容：

```
evaluation:
  task: mm_llm_judge
  strategy: judge
  metric: all
```

## 限制

- 每次评估只支持一个 .jsonl 文件。
- 该文件必须严格遵循定义的架构。
- Nova MM Judge 模型仅支持图像参考。
- Nova MM Judge 模型对于 Amazon Nova Lite 规格是一样的。
- 目前不支持自定义评估模型。
- 不支持 Amazon S3 图像 URI。
- 输入数据集应确保：排除 image 属性后，所有记录的上下文长度均小于 12k。

## Rubric Judge

Rubric Judge 是基于 Amazon Nova 2.0 Lite 构建的增强版 LLM-as-a-judge 评测模型。与仅提供偏好判定（A 优于 B、B 优于 A 或平局）的[原始评判模型](#)不同，Rubric Judge 可针对每个提示词动态生成自定义评估准则，并在多个维度上给出细粒度评分。

### 关键功能：

- 动态评估维度生成：根据输入提示自动生成相关的评估维度
- 加权评分：为各评估维度分配重要性权重，体现其相对优先级
- 细粒度评测：对每个维度以二分类（是/否）或量表（1-5 分）形式给出详细评分
- 质量指标：计算连续型质量得分（0-1 区间），量化不同响应之间的差异程度

模型生成的评估维度示例：

```
price_validation:
  description: "The response includes validation to ensure price is a positive value."
  type: "scale"
  weight: 0.3
```

模型会依据所有生成的评估维度对两组响应进行评估，再利用各维度得分得出最终偏好判定。

主题

- [配方配置](#)
- [输入数据集格式](#)
- [评估输出](#)
- [推理模型支持](#)

配方配置

Rubric Judge 配方

在配方中通过设置 `task: rubric_llm_judge` 来启用 Rubric Judge：

```
run:
  name: nova-eval-job-name # [MODIFIABLE] Unique
  identifier for your evaluation job
  model_type: amazon.nova-2-lite-v1:0:256k # [FIXED] Rubric Judge model
  type
  model_name_or_path: "nova-lite-2/prod" # [FIXED] Path to model
  checkpoint or identifier
  replicas: 1 # [MODIFIABLE] Number of
  replicas for SageMaker Training job
  data_s3_path: "" # [FIXED] Leave empty for
  SageMaker Training job
  output_s3_path: "" # [FIXED] Leave empty for
  SageMaker Training job

evaluation:
  task: rubric_llm_judge # [FIXED] Evaluation task -
  enables Rubric Judge
  strategy: judge # [FIXED] Evaluation strategy
  metric: all # [FIXED] Metric calculation
  method
```

```

inference:
  max_new_tokens: 12000 # [MODIFIABLE] Maximum tokens
  to generate
  top_k: -1 # [MODIFIABLE] Top-k sampling
  parameter
  top_p: 1.0 # [MODIFIABLE] Nucleus sampling
  parameter
  temperature: 0 # [MODIFIABLE] Sampling
  temperature (0 = deterministic)

```

## 原始 LLM-as-a-judge 配方 ( 供比较 )

### 原始评判模型使用 task: llm\_judge :

```

run:
  name: eval-job-name # [MODIFIABLE] Unique
  identifier for your evaluation job
  model_type: amazon.nova-micro-v1:0:128k # [FIXED] Model type
  model_name_or_path: "nova-micro/prod" # [FIXED] Path to model
  checkpoint or identifier
  replicas: 1 # [MODIFIABLE] Number of
  replicas for SageMaker Training job
  data_s3_path: "" # [FIXED] Leave empty for
  SageMaker Training job
  output_s3_path: "" # [FIXED] Leave empty for
  SageMaker Training job

evaluation:
  task: llm_judge # [FIXED] Original judge task
  strategy: judge # [FIXED] Evaluation strategy
  metric: all # [FIXED] Metric calculation
  method

inference:
  max_new_tokens: 12000 # [MODIFIABLE] Maximum tokens
  to generate
  top_k: -1 # [MODIFIABLE] Top-k sampling
  parameter
  top_p: 1.0 # [MODIFIABLE] Nucleus sampling
  parameter
  temperature: 0 # [MODIFIABLE] Sampling
  temperature (0 = deterministic)

```

## 输入数据集格式

输入数据集格式与[原始评判模型](#)完全一致：

必填字段：

- `prompt`：字符串类型，包含输入提示与指令
- `response_A`：字符串类型，基准模型的输出
- `response_B`：字符串类型，自定义模型的输出

示例数据集 ( JSONL 格式 )：

```
{"prompt": "What is the most effective way to combat climate change?", "response_A": "The most effective way to combat climate change is through a combination of transitioning to renewable energy sources and implementing strict carbon pricing policies. This creates economic incentives for businesses to reduce emissions while promoting clean energy adoption.", "response_B": "We should focus on renewable energy. Solar and wind power are good. People should drive electric cars. Companies need to pollute less."}
{"prompt": "Explain how a computer's CPU works", "response_A": "CPU is like brain of computer. It does math and makes computer work fast. Has lots of tiny parts inside.", "response_B": "A CPU (Central Processing Unit) functions through a fetch-execute cycle, where instructions are retrieved from memory, decoded, and executed through its arithmetic logic unit (ALU). It coordinates with cache memory and registers to process data efficiently using binary operations."}
{"prompt": "How does photosynthesis work?", "response_A": "Plants do photosynthesis to make food. They use sunlight and water. It happens in leaves.", "response_B": "Photosynthesis is a complex biochemical process where plants convert light energy into chemical energy. They utilize chlorophyll to absorb sunlight, combining CO2 and water to produce glucose and oxygen through a series of chemical reactions in chloroplasts."}
```

格式要求：

- 每条记录必须是单行 JSON 对象
- 各行记录以换行符分隔
- 严格遵循示例中所示的字段命名规则

## 评估输出

### 输出结构

与原始评判模型相比，Rubric Judge 可生成更完善的评估指标：

```
{
  "config_general": {
    "lighteval_sha": "string",
    "num_fewshot_seeds": "int",
    "max_samples": "int | null",
    "job_id": "int",
    "start_time": "float",
    "end_time": "float",
    "total_evaluation_time_seconds": "string",
    "model_name": "string",
    "model_sha": "string",
    "model_dtype": "string | null",
    "model_size": "string"
  },
  "results": {
    "custom|rubric_llm_judge_judge|0": {
      "a_scores": "float",
      "a_scores_stderr": "float",
      "b_scores": "float",
      "b_scores_stderr": "float",
      "ties": "float",
      "ties_stderr": "float",
      "inference_error": "float",
      "inference_error_stderr": "float",
      "score": "float",
      "score_stderr": "float",
      "weighted_score_A": "float",
      "weighted_score_A_stderr": "float",
      "weighted_score_B": "float",
      "weighted_score_B_stderr": "float",
      "score_margin": "float",
      "score_margin_stderr": "float",
      "winrate": "float",
      "lower_rate": "float",
      "upper_rate": "float"
    }
  },
  "versions": {
```

```

    "custom|rubric_llm_judge_judge|0": "int"
  }
}

```

## Rubric Judge 新增指标

以下六项指标为 Rubric Judge 独有，可提供细粒度的质量评测：

| 指标                      | 说明                                                                                                                                       |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| weighted_score_A        | 基于模型生成的所有评估维度，对 response_A 计算的归一化加权平均质量得分。得分按维度重要性加权，并归一化到 0-1 区间（分值越高，质量越好）                                                             |
| weighted_score_A_stderr | weighted_score_A 的均值标准误，用于表示统计不确定性                                                                                                       |
| weighted_score_B        | 基于模型生成的所有评估维度，对 response_B 计算的归一化加权平均质量得分。得分按维度重要性加权，并归一化到 0-1 区间（分值越高，质量越好）                                                             |
| weighted_score_B_stderr | weighted_score_B 的均值标准误，用于表示统计不确定性                                                                                                       |
| score_margin            | 加权分数差值（计算公式： $\text{weighted\_score\_A} - \text{weighted\_score\_B}$ ）。范围：-1.0 至 1.0。正值表示 response_A 更优；负值表示 response_B 更优；接近 0 表示两者质量相近 |
| score_margin_stderr     | score_margin 的均值标准误，用于表示质量差异测量结果的不确定性                                                                                                    |

## 加权分数指标说明

用途：加权分数提供连续型质量评估，作为二分类偏好判定的补充，便于更深入地分析模型性能。

与原始评判模型的主要区别：

- 原始评判模型：仅输出离散偏好结果 ( A 优于 B、B 优于 A、A 与 B 持平 )
- Rubric Judge：基于自定义评估维度，同时输出偏好结果与连续型质量得分 ( 0-1 区间 )

score\_margin 解读：

- $\text{score\_margin} = -0.128$ ：表示 response\_B 得分比 response\_A 高 12.8 个百分点
- $|\text{score\_margin}| < 0.1$ ：质量差异极小 ( 判定接近 )
- $|\text{score\_margin}| > 0.2$ ：质量差异明显 ( 判定置信度高 )

使用案例：

- 模型优化：定位模型表现不足的具体方面
- 质量量化：不仅评估胜负关系，更衡量性能差距的大小
- 置信度评测：区分接近判定与明确的质量差异

#### Important

最终判定仍以评判模型的显式偏好标签为准，以保留整体推理能力，并通过正向/反向评估有效缓解位置偏差。加权分数仅作为可观测性工具，不能替代主判定结果。

计算方法

加权分数通过以下流程计算：

- 提取评估维度数据：解析评判模型的 YAML 输出，提取各维度得分与权重
- 得分归一化：
  - 量表型维度 ( 1-5 分 )：通过公式  $(\text{score} - 1) / 4$  归一化到 0-1 区间
  - 二分型维度 ( 是/否 )：转换为 1.0/0.0
- 应用权重：将每条归一化得分乘以对应维度的权重
- 汇总得分：对每条响应的所有加权分数求和
- 计算差值：计算  $\text{score\_margin} = \text{weighted\_score\_A} - \text{weighted\_score\_B}$

示例：如果若 response\_A 加权总分为 0.65，response\_B 为 0.78，则 score\_margin = -0.13，表示在所有加权维度下，response\_B 的质量高出 13 个百分点。

## 推理模型支持

推理模型支持功能可借助具备推理能力的 Amazon Nova 模型开展评测，这类模型会在生成最终响应前执行显式的内部推理。该功能通过 reasoning\_effort 参数实现 API 级别的控制，可动态启用/禁用推理功能，有助于提升复杂分析类任务的响应质量。

支持的模型：

- amazon.nova-2-lite-v1:0:256k

## 配方配置

在配方的 inference 模块中添加 reasoning\_effort 参数，即可启用推理功能：

```
run:
  name: eval-job-name # [MODIFIABLE] Unique
  identifier for your evaluation job
  model_type: amazon.nova-2-lite-v1:0:256k # [FIXED] Must be a reasoning-
  supported model
  model_name_or_path: nova-lite-2/prod # [FIXED] Path to model
  checkpoint or identifier
  replicas: 1 # [MODIFIABLE] Number of
  replicas for SageMaker Training job
  data_s3_path: "" # [MODIFIABLE] Leave empty for
  SageMaker Training job; optional for SageMaker SageMaker HyperPod job
  output_s3_path: "" # [MODIFIABLE] Output path for
  SageMaker SageMaker HyperPod job (not compatible with SageMaker Training jobs)

evaluation:
  task: mmlu # [MODIFIABLE] Evaluation task
  strategy: generate # [MODIFIABLE] Evaluation
  strategy
  metric: all # [MODIFIABLE] Metric
  calculation method

inference:
  reasoning_effort: high # [MODIFIABLE] Enables
  reasoning mode; options: low/medium/high or null to disable
  max_new_tokens: 200 # [MODIFIABLE] Maximum tokens
  to generate
```

```

top_k: 50 # [MODIFIABLE] Top-k sampling
parameter
top_p: 1.0 # [MODIFIABLE] Nucleus
sampling parameter
temperature: 0 # [MODIFIABLE] Sampling
temperature (0 = deterministic)

```

## reasoning\_effort 参数使用说明

reasoning\_effort 参数用于控制具备推理能力模型的推理行为。

先决条件：

- 模型兼容性：仅当 model\_type 指定了具备推理能力的模型（当前为 amazon.nova-2-lite-v1:0:256k）时，才需设置 reasoning\_effort
- 错误处理：对不受支持的模型使用 reasoning\_effort 将会失败，并返回 ConfigValidationError: "Reasoning mode is enabled but model '{model\_type}' does not support reasoning. Please use a reasoning-capable model or disable reasoning mode." 错误

可用选项：

| Option    | 行为        | 词元限制           | 使用案例                |
|-----------|-----------|----------------|---------------------|
| null (默认) | 禁用推理模式    | 不适用            | 无需推理开销的标准评估         |
| low       | 启用推理并施加约束 | 内部推理占用 4000 词元 | 需要简洁推理的场景；兼顾速度与成本优化 |
| high      | 启用推理且无约束  | 内部推理无词元上限      | 需要深度分析与分步推理的复杂问题    |

## 何时启用推理

满足以下场景时，建议使用推理模式（**low**、**medium** 或 **high**）：

- 复杂问题求解任务（数学、逻辑谜题、编程）

- 需要中间推理过程的多步分析类问题
- 详细推导或分步思考可提升准确率的任务
- 优先保证响应质量而非速度的场景

满足以下场景时，建议使用非推理模式（不设置该参数）：

- 简单问答或事实查询
- 创意写作类任务
- 对响应速度要求极高的场景
- 需要排除推理开销的性能基准测试
- 推理无法提升效果、需优化成本的场景

### 问题排查

错误：“Reasoning mode is enabled but model does not support reasoning”

原因：reasoning\_effort 参数被设为非空值，但指定的 model\_type 支持推理。

解决方法：

- 确认模型类型为 amazon.nova-2-lite-v1:0:256k
- 若使用其他模型，或切换为具备推理能力的模型，或从配方中移除 reasoning\_effort 参数

### 启动评估作业

下面提供了建议的评估实例类型和模型类型配置：

```
# Install Dependencies (Helm - https://helm.sh/docs/intro/install/)
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
rm -f ./get_helm.sh

# Install the SageMaker HyperPod CLI
git clone --recurse-submodules https://github.com/aws/sagemaker-hyperpod-cli.git
git checkout -b release_v2
cd sagemaker-hyperpod-cli
pip install .
```

```

# Verify the installation
hyperpod --help

# Connect to a SageMaker HyperPod Cluster
hyperpod connect-cluster --cluster-name cluster-name

# Submit the Job using the recipe for eval
# Namespace by default should be kubeflow
hyperpod start-job [--namespace namespace] --recipe evaluation/nova/
nova_micro_p5_48xl_general_text_benchmark_eval --override-parameters \
'{
  "instance_type": "p5d.48xlarge",
  "container": "708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-evaluation-repo:SM-
HP-Eval-V2-latest",
  "recipes.run.name": custom-run-name,
  "recipes.run.model_type": model_type,
  "recipes.run.model_name_or_path" " model name or finetune checkpoint s3uri,
  "recipes.run.data_s3_path": s3 for input data only for genqa and llm_judge, must be
full S3 path that include filename,
}'

# List jobs
hyperpod list-jobs [--namespace namespace] [--all-namespaces]

# Getting Job details
hyperpod get-job --job-name job-name [--namespace namespace] [--verbose]

# Listing Pods
hyperpod list-pods --job-name job-name --namespace namespace

# Cancel Job
hyperpod cancel-job --job-name job-name [--namespace namespace]

```

您还应该能够通过 Amazon EKS 集群控制台查看作业状态。

## 访问和分析评估结果

成功完成评估作业后，您可以参考本节中的信息访问并分析结果。根据配方中定义的 `output_s3_path`（例如 `s3://output_path/`），输出结构如下所示：

```
job_name/
```

```
### eval-result/  
#   ### results_[timestamp].json  
#   ### inference_output.jsonl (only present for gen_qa)  
#   ### details/  
#       ### model/  
#           ### execution-date-time/  
#               ###details_task_name_#_datetime.parquet  
### tensorboard-results/  
    ### eval/  
        ### events.out.tfevents.[timestamp]
```

指标结果存储在指定的 S3 输出位置 `s3://output_path/job_name/eval-result/result-timestamp.json`。

Tensorboard 结果存储在 S3 路径 `s3://output_path/job_name/eval-tensorboard-result/eval/event.out.tfevents.epoch+ip` 中。

除 `llm_judge` 和 `strong_reject` 之外的所有推理输出都存储在 S3 路径中：`s3://output_path/job_name/eval-result/details/model/taskname.parquet`。

对于 `gen_qa`，`inference_output.jsonl` 文件针对每个 JSON 对象均包含以下字段：

- `prompt`：提交给模型的最终提示
- `推理`：模型的原始推理输出
- `gold`：来自输入数据集的目标响应
- `metadata`：来自输入数据集的元数据字符串（如果提供）

要在 Tensorboard 中直观显示您的评估指标，请完成以下步骤：

1. 导航到 SageMaker AI Tensorboard。
2. 选择 S3 文件夹。
3. 添加您的 S3 文件夹路径，例如 `s3://output_path/job-name/eval-tensorboard-result/eval`。
4. 等待同步完成。

时间序列、标量和文本可视化均可用。

我们建议您遵循以下最佳实操：

- 按模型和基准测试类型整理输出路径。
- 保持一致的命名约定以便于跟踪。
- 将解压缩的结果保存在安全的位置。
- 监控 TensorBoard 同步状态以成功加载数据。

您可以在 CloudWatch 日志组 `/aws/sagemaker/Clusters/cluster-id` 中找到 SageMaker HyperPod 作业错误日志。

### 对数概率输出格式

在推理设置中配置 `top_logprobs` 后，评估输出会在 parquet 文件中包含词元级别的对数概率。每个词元位置都会以字典形式，返回排名靠前的候选词元及其对数概率，结构如下：

```
{
  "Ġint": {"logprob_value": -17.8125, "decoded_value": " int"},
  "Ġthe": {"logprob_value": -2.345, "decoded_value": " the"}
}
```

每个词元条目包含：

- `logprob_value`：词元的对数概率值
- `decoded_value`：词元经解码后人类可读的字符串形式

原始分词器词元作为字典键以确保唯一性，`decoded_value` 则提供可读形式。

### RFT 评测

#### Note

仅当您为 Amazon Nova Forge 客户时，方可在自有 AWS 环境中通过远程奖励函数开展评测。

#### Important

`rl_env` 配置字段仅用于评测，不用于训练。训练期间，需要使用 `reward_lambda_arn` (单轮) 或带有 `rollout.delegate: true` (多轮) 的 BYOO 基础设施来配置奖励函数。

## 什么是 RFT 评测？

RFT 评测支持您在强化学习微调的训练前、训练中及训练后，通过自定义奖励函数评测模型性能。与采用预定义指标的标准评测不同，RFT 评测支持通过 Lambda 函数自定义成功标准，并依据您的具体需求对模型输出进行评分。

## 为什么要使用 RFT 进行评测？

评测对于判断强化微调过程是否实现以下目标至关重要：

- 提升模型与您特定使用案例及人类价值偏好的对齐程度
- 维持或提升模型在核心任务上的能力
- 避免产生非预期副作用，例如事实准确性下降、内容冗余度增加或其他任务性能退化
- 满足由奖励函数定义的自定义成功标准

## 何时使用 RFT 评测

在以下场景中使用 RFT 评测：

- RFT 训练之前：在评测数据集上建立基线指标
- RFT 训练期间：使用中间检查点监控训练进度
- RFT 训练之后：验证最终模型是否符合您的要求
- 模型对比：采用一致的奖励标准对多个模型版本进行评测

### Note

如需领域专属的自定义评估指标，请使用 RFT 评测。对于通用评测（准确率、困惑度、BLEU），请使用标准评测方法。

## 主题

- [数据格式要求](#)
- [准备评测配方](#)
- [预设奖励函数](#)
- [创建奖励函数](#)
- [IAM 权限](#)

- [执行评测作业](#)
- [了解评测结果](#)

## 数据格式要求

### 输入数据结构

RFT 评测输入数据必须遵循 OpenAI 强化微调格式。每个样本都是一个 JSON 对象，其中包含：

- `messages`：包含 `system` 和 `user` 角色的对话轮次数组
- 其他可选元数据，例如 `reference_answer`

### 数据格式示例

以下示例显示了所需格式：

```
{
  "messages": [
    {
      "role": "user",
      "content": [
        {
          "type": "text",
          "text": "Solve for x. Return only JSON like {\"x\": <number>}. Equation: 2x + 5 = 13"
        }
      ]
    }
  ],
  "reference_answer": {
    "x": 4
  }
}
```

## 目前的局限性

以下限制适用于 RFT 评测：

- 纯文本模式：不支持多模态输入（图像、音频、视频）
- 单轮对话：仅支持单条用户消息（不支持多轮对话）

- JSON 格式：输入数据必须为 JSONL 格式（每行一个 JSON 对象）
- 模型输出：根据指定模型生成的补全进行评测

## 准备评测配方

### 示例配方配置

以下示例展示了一个完整的 RFT 评测配方：

```
run:
  name: nova-lite-rft-eval-job
  model_type: amazon.nova-lite-v1:0:300k
  model_name_or_path: s3://escrow_bucket/model_location      # [MODIFIABLE] S3 path to
  your model or model identifier
  replicas: 1  # [MODIFIABLE] For SageMaker
  Training jobs only; fixed for SageMaker HyperPod jobs
  data_s3_path: ""  # [REQUIRED FOR HYPERPOD]
  Leave empty for SageMaker Training jobs
  output_s3_path: ""                                       # [REQUIRED] Output artifact
  S3 path for evaluation results

evaluation:
  task: rft_eval   # [FIXED] Do not modify
  strategy: rft_eval                                       # [FIXED] Do not modify
  metric: all   # [FIXED] Do not modify

# Inference Configuration
inference:
  max_new_tokens: 8196                                     # [MODIFIABLE] Maximum tokens
  to generate
  top_k: -1  # [MODIFIABLE] Top-k sampling
  parameter
  top_p: 1.0   # [MODIFIABLE] Nucleus
  sampling parameter
  temperature: 0   # [MODIFIABLE] Sampling
  temperature (0 = deterministic)
  top_logprobs: 0

# Evaluation Environment Configuration (NOT used in training)
rl_env:
  reward_lambda_arn: arn:aws:lambda:<region>:<account_id>:function:<reward-function-
  name>
```

## 预设奖励函数

两个预设的奖励函数 ( `prime_code` 和 `prime_math` ) 可用作 Lambda 层，以便与 RFT Lambda 函数集成。

### 概述

这些预设函数提供开箱即用的评测能力，适用于：

- `prime_code`：代码生成和正确性评测
- `prime_math`：数学推理和问题求解评测

### 快速设置

要使用预设奖励函数，请执行以下操作：

1. 从 [nova-custom-eval-sdk 版本](#) 页面下载 Lambda 层
2. 使用 AWS CLI 发布 Lambda 层：

```
aws lambda publish-layer-version \  
  --layer-name preset-function-layer \  
  --description "Preset reward function layer with dependencies" \  
  --zip-file fileb://universal_reward_layer.zip \  
  --compatible-runtimes python3.9 python3.10 python3.11 python3.12 \  
  --compatible-architectures x86_64 arm64
```

3. 在 AWS 管理控制台中将该层添加到 Lambda 函数 ( 从自定义层中选择 `preset-function-layer`，并添加 `AWSSDKPandas-Python312` 以获取 `numpy` 依赖项 )
4. 在 Lambda 代码中导入并使用：

```
from prime_code import compute_score # For code evaluation  
from prime_math import compute_score # For math evaluation
```

### `prime_code` 函数

用途：通过针对测试用例执行代码并衡量正确性来评测 Python 代码生成任务。

评测输入数据集格式示例：

```

{"messages":[{"role":"user","content":"Write a function that returns the sum of
two numbers."}], "reference_answer":{"inputs":["3\n5", "10\n-2", "0\n0"], "outputs":
["8", "8", "0"]}}
{"messages":[{"role":"user","content":"Write a function to check if a number
is even."}], "reference_answer":{"inputs":["4", "7", "0", "-2"], "outputs":
["True", "False", "True", "True"]}}

```

主要特征：

- 从 Markdown 代码块中自动提取代码
- 函数检测与基于调用的测试
- 带超时保护的测试用例执行
- 语法验证与编译检查
- 包含回溯信息的详细错误报告

prime\_math 函数

用途：依托符号计算能力，评测模型的数学推理与问题求解水平。

输入格式：

```

{"messages":[{"role":"user","content":"What is the derivative of x^2 +
3x?."}], "reference_answer":"2*x + 3"}

```

主要特征：

- 使用 SymPy 进行符号计算评测
- 支持多种答案格式 ( LaTeX、纯文本、符号 )
- 数学等价性检查
- 表达式归一化与化简

最佳实践

使用预设奖励函数时，请遵循以下最佳实践：

- 在测试用例中使用正确的数据类型 ( 整数与字符串、布尔值与 True )
- 在代码问题中提供清晰的函数签名

- 在测试输入中包括边缘情况 ( 零、负数、空输入 )
- 在参考答案中统一设置数学表达式的格式
- 在部署前使用样本数据测试奖励函数

## 创建奖励函数

### Lambda ARN

必须遵循以下格式来表示 Lambda ARN :

```
"arn:aws:lambda:*:*:function:*SageMaker*"
```

如果 Lambda 不遵循此命名方案，作业将失败并显示以下错误：

```
[ERROR] Unexpected error: lambda_arn must contain one of: ['SageMaker', 'sagemaker', 'Sagemaker'] when running on SMHP platform (Key: lambda_arn)
```

### Lambda 函数结构

Lambda 函数接收批量的模型输出并返回奖励分数。以下是一个实现示例：

```
from typing import List, Any
import json
import re
from dataclasses import asdict, dataclass

@dataclass
class MetricResult:
    """Individual metric result."""
    name: str
    value: float
    type: str

@dataclass
class RewardOutput:
    """Reward service output."""
    id: str
    aggregate_reward_score: float
    metrics_list: List[MetricResult]
```

```
def lambda_handler(event, context):
    """ Main lambda handler """
    return lambda_grader(event)

def lambda_grader(samples: list[dict]) -> list[dict]:
    """ Core grader function """
    scores: List[RewardOutput] = []

    for sample in samples:
        print("Sample: ", json.dumps(sample, indent=2))

        # Extract components
        idx = sample.get("id", "no id")
        if not idx or idx == "no id":
            print(f"ID is None/empty for sample: {sample}")

        ground_truth = sample.get("reference_answer")

        if "messages" not in sample:
            print(f"Messages is None/empty for id: {idx}")
            continue

        if ground_truth is None:
            print(f"No answer found in ground truth for id: {idx}")
            continue

        # Get model's response (last turn is assistant turn)
        last_message = sample["messages"][-1]

        if last_message["role"] != "nova_assistant":
            print(f>Last message is not from assistant for id: {idx}")
            continue

        if "content" not in last_message:
            print(f"Completion text is empty for id: {idx}")
            continue

        model_text = last_message["content"]

        # --- Actual scoring logic (lexical overlap) ---
        ground_truth_text = _extract_ground_truth_text(ground_truth)
```

```
# Calculate main score and individual metrics
overlap_score = _lexical_overlap_score(model_text, ground_truth_text)

# Create two separate metrics as in the first implementation
accuracy_score = overlap_score # Use overlap as accuracy
fluency_score = _calculate_fluency(model_text) # New function for fluency

# Create individual metrics
metrics_list = [
    MetricResult(name="accuracy", value=accuracy_score, type="Metric"),
    MetricResult(name="fluency", value=fluency_score, type="Reward")
]

ro = RewardOutput(
    id=idx,
    aggregate_reward_score=overlap_score,
    metrics_list=metrics_list
)

print(f"Response for id: {idx} is {ro}")
scores.append(ro)

# Convert to dict format
result = []
for score in scores:
    result.append({
        "id": score.id,
        "aggregate_reward_score": score.aggregate_reward_score,
        "metrics_list": [asdict(metric) for metric in score.metrics_list]
    })

return result

def _extract_ground_truth_text(ground_truth: Any) -> str:
    """
    Turn the `ground_truth` field into a plain string.
    """
    if isinstance(ground_truth, str):
        return ground_truth

    if isinstance(ground_truth, dict):
        # Common patterns: { "explanation": "...", "answer": "..."}

```

```
    if "explanation" in ground_truth and isinstance(ground_truth["explanation"],
str):
        return ground_truth["explanation"]
    if "answer" in ground_truth and isinstance(ground_truth["answer"], str):
        return ground_truth["answer"]
    # Fallback: stringify the whole dict
    return json.dumps(ground_truth, ensure_ascii=False)

# Fallback: stringify anything else
return str(ground_truth)

def _tokenize(text: str) -> List[str]:
    # Very simple tokenizer: lowercase + alphanumeric word chunks
    return re.findall(r"\w+", text.lower())

def _lexical_overlap_score(model_text: str, ground_truth_text: str) -> float:
    """
    Simple lexical overlap score in [0, 1]:
    score = |tokens(model) ∩ tokens(gt)| / |tokens(gt)|
    """
    gt_tokens = _tokenize(ground_truth_text)
    model_tokens = _tokenize(model_text)

    if not gt_tokens:
        return 0.0

    gt_set = set(gt_tokens)
    model_set = set(model_tokens)
    common = gt_set & model_set

    return len(common) / len(gt_set)

def _calculate_fluency(text: str) -> float:
    """
    Calculate a simple fluency score based on:
    - Average word length
    - Text length
    - Sentence structure

    Returns a score between 0 and 1.
    """
```

```
# Simple implementation - could be enhanced with more sophisticated NLP
words = _tokenize(text)

if not words:
    return 0.0

# Average word length normalized to [0,1] range
# Assumption: average English word is ~5 chars, so normalize around that
avg_word_len = sum(len(word) for word in words) / len(words)
word_len_score = min(avg_word_len / 10, 1.0)

# Text length score - favor reasonable length responses
ideal_length = 100 # words
length_score = min(len(words) / ideal_length, 1.0)

# Simple sentence structure check (periods, question marks, etc.)
sentence_count = len(re.findall(r'[.!?]+', text)) + 1
sentence_ratio = min(sentence_count / (len(words) / 15), 1.0)

# Combine scores
fluency_score = (word_len_score + length_score + sentence_ratio) / 3

return fluency_score
```

## Lambda 请求格式

Lambda 函数接收以下格式的数据：

```
[
  {
    "id": "sample-001",
    "messages": [
      {
        "role": "user",
        "content": [
          {
            "type": "text",
            "text": "Do you have a dedicated security team?"
          }
        ]
      },
      {
        "role": "nova_assistant",
        "content": [
```

```

    {
      "type": "text",
      "text": "As an AI developed by Company, I don't have a dedicated security
team in the traditional sense. However, the development and deployment of AI systems
like me involve extensive security measures, including data encryption, user privacy
protection, and other safeguards to ensure safe and responsible use."
    }
  ]
},
"reference_answer": {
  "compliant": "No",
  "explanation": "As an AI developed by Company, I do not have a traditional
security team. However, the deployment involves stringent safety measures, such as
encryption and privacy safeguards."
}
}
]

```

### Note

消息结构包含嵌套的 `content` 数组，与输入数据格式相匹配。角色为 `nova_assistant` 的最后一条消息包含模型生成的响应。

## Lambda 响应格式

Lambda 函数必须按以下格式返回数据：

```

[
  {
    "id": "sample-001",
    "aggregate_reward_score": 0.75,
    "metrics_list": [
      {
        "name": "accuracy",
        "value": 0.85,
        "type": "Metric"
      },
      {
        "name": "fluency",
        "value": 0.90,

```

```

        "type": "Reward"
    }
]
}
]

```

响应字段：

- `id`：必须与输入样本 ID 匹配
- `aggregate_reward_score`：总体得分（通常为 0.0 到 1.0）
- `metrics_list`：各项指标组成的数组，包含以下字段：
  - `name`：指标标识符（例如 `accuracy`、`fluency`）
  - `value`：指标得分（通常为 0.0 到 1.0）
  - `type`：Metric（用于报告）或 Reward（用于训练）

## IAM 权限

### 所需的权限

SageMaker AI 执行角色必须具有调用 Lambda 函数的权限。将此策略添加到 SageMaker AI 执行角色：

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction"
      ],
      "Resource": "arn:aws:lambda:region:account-id:function:function-name"
    }
  ]
}

```

## Lambda 执行角色

Lambda 函数的执行角色需要基本的 Lambda 执行权限：

```

{

```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogGroup",
      "logs:CreateLogStream",
      "logs:PutLogEvents"
    ],
    "Resource": "arn:aws:logs:*:*:*"
  }
]
```

其他权限：如果 Lambda 函数访问其他 AWS 服务（例如用于参考数据的 Amazon S3、用于日志记录的 DynamoDB），请将这些权限添加到 Lambda 执行角色。

## 执行评测作业

### 1. 准备数据

- 根据数据格式要求对评测数据进行格式化
- 将 JSONL 文件上传到 Amazon S3：s3://your-bucket/eval-data/eval\_data.jsonl

### 2. 配置配方

使用配置更新示例配方：

- 将 model\_name\_or\_path 设置为模型路径
- 将 lambda\_arn 设置为奖励函数 ARN
- 将 output\_s3\_path 设置为目标输出位置
- 根据需要调整 inference 参数

将配方另存为 rft\_eval\_recipe.yaml

### 3. 运行评测

使用提供的笔记本执行评测作业：[Nova 模型评测笔记本](#)

### 4. 监控进度

通过以下方式监控评测作业：

- SageMaker AI 控制台：检查作业状态与日志
- CloudWatch Logs：查看详细执行日志
- Lambda 日志：调试奖励函数相关问题

## 了解评测结果

### 输出格式

评测作业会以 JSONL 格式将结果输出到指定的 Amazon S3 位置。每行包含一个样本的评测结果：

```
{
  "id": "sample-001",
  "aggregate_reward_score": 0.75,
  "metrics_list": [
    {
      "name": "accuracy",
      "value": 0.85,
      "type": "Metric"
    },
    {
      "name": "fluency",
      "value": 0.90,
      "type": "Reward"
    }
  ]
}
```

### Note

RFT 评测作业输出格式与 Lambda 响应格式完全一致。评测服务会原样透传 Lambda 函数的响应，确保奖励计算结果与最终输出保持一致。

## 解析结果

### 奖励总分：

- 取值范围：通常为 0.0（最差）到 1.0（最佳），具体取决于实现逻辑
- 用途：以单一数值概括整体性能

- 应用场景：模型对比、跟踪训练过程中的效果提升

各项指标：

- 指标类型：用于分析的参考性指标
- 奖励类型：RFT 训练过程中使用的指标
- 结果解读：数值越高通常表示性能越好（除非设计了反向指标）

性能基准

何种表现为“良好”取决于实际使用案例：

| 分数区间      | 评估结果 | 处理建议        |
|-----------|------|-------------|
| 0.8 – 1.0 | 优秀   | 模型可直接部署     |
| 0.6 – 0.8 | 良好   | 可进行小幅优化     |
| 0.4 – 0.6 | 一般   | 需大幅优化提升     |
| 0.0 – 0.4 | 较差   | 检查训练数据和奖励函数 |

#### Important

以上仅为通用准则。请根据业务需求、基线模型性能、领域特定约束以及进一步训练的成本效益分析，自行定义阈值。

## 使用 MLflow 监控 HyperPod

您可借助 MLflow 跟踪并监控 SageMaker HyperPod 上的训练作业。请按如下步骤搭建 MLflow，并将其接入训练配方。

### 创建 MLflow 应用程序

#### AWS CLI 命令示例

```
aws sagemaker-mlflow create-mlflow-app \
```

```
--name <app-name> \  
--artifact-store-uri <s3-bucket-name> \  
--role-arn <role-arn> \  
--region <region-name>
```

## 示例输出

```
{  
  "Arn": "arn:aws:sagemaker:us-east-1:111122223333:mlflow-app/app-LGZE0Z2UY4NZ"  
}
```

## 生成预签名 URL

### AWS CLI 命令示例

```
aws sagemaker-mlflow create-presigned-mlflow-app-url \  
  --arn <app-arn> \  
  --region <region-name> \  
  --output text
```

## 示例输出

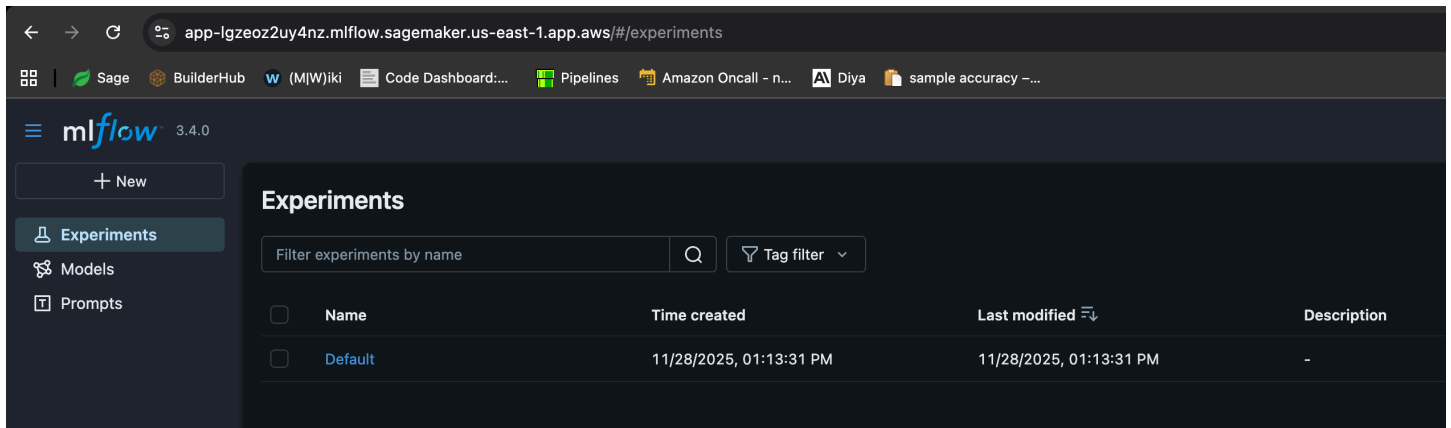
```
https://app-LGZE0Z2UY4NZ.mlflow.sagemaker.us-east-1.app.aws/auth?  
authToken=eyJhbGciOiJIUzI1NiJ9.eyJhdXRoVG9rZW5JZCI6IkxETVBPUyIsImZhc0NyZWRLbnRpYWxzIjoiQWdWNGhD
```

## 打开预签名 URL 查看应用程序

### 单击

```
https://app-LGZE0Z2UY4NZ.mlflow.sagemaker.us-east-1.app.aws/auth?  
authToken=eyJhbGciOiJIUzI1NiJ9.eyJhdXRoVG9rZW5JZCI6IkxETVBPUyIsImZhc0NyZWRLbnRpYWxzIjoiQWdWNGhD
```

## 视图

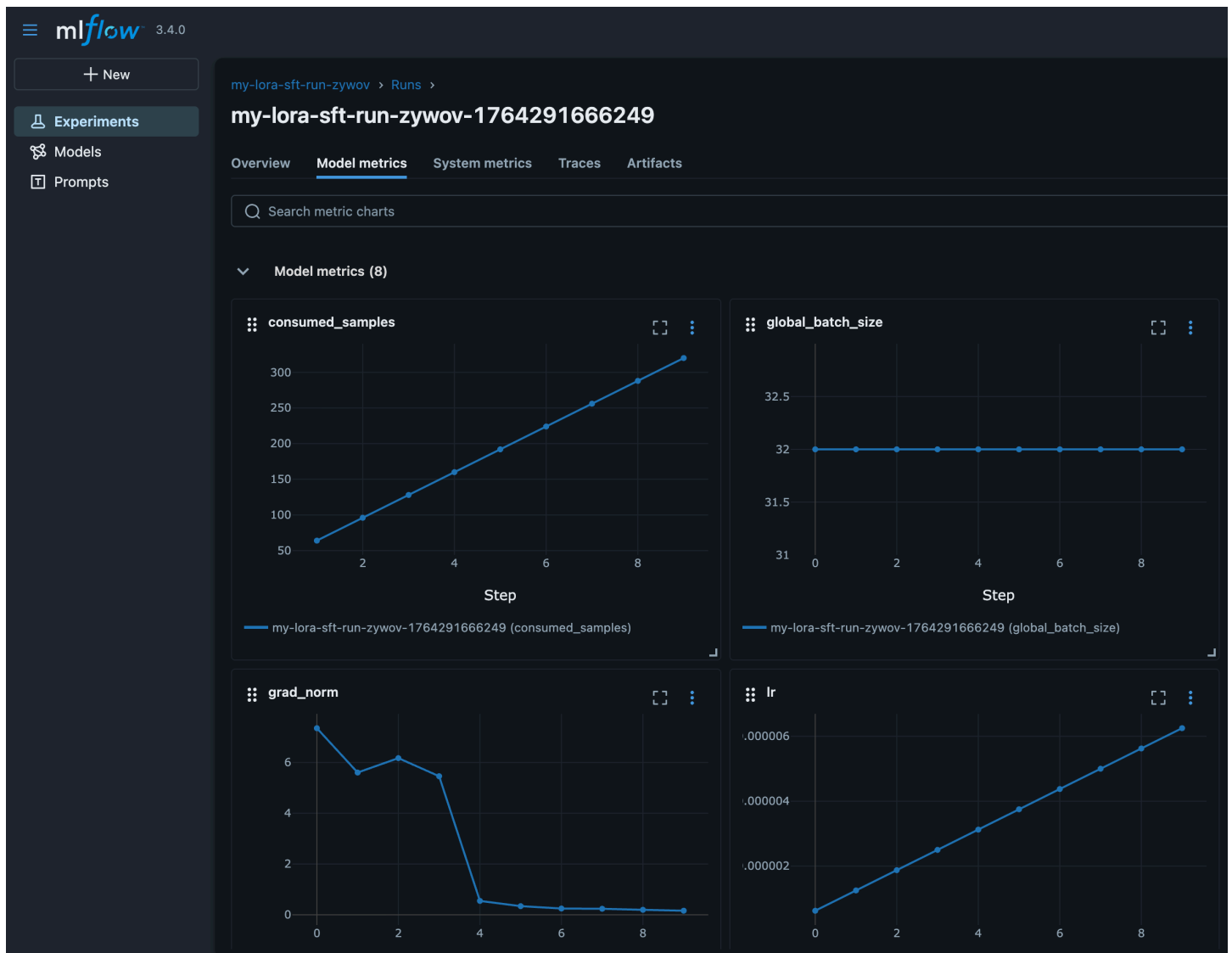


在 SageMaker HyperPod 配方的 run 代码块中进行配置

## 指南

```
run
  mlflow_tracking_uri: arn:aws:sagemaker:us-east-1:111122223333:mlflow-app/app-
  LGZE0Z2UY4NZ
```

## 视图



## 迭代训练

### 概述

迭代训练指通过多种训练方法，经多轮训练周期对模型进行反复微调的过程，具体包括训练、评估、分析错误、调整数据/目标/超参数等环节，且每一轮训练均基于上一轮的检查点启动。该方法可帮助您系统性定位模型的失效场景，整合针对特定短板的精选示例，并持续适配不断变化的业务需求。

相较于单次训练的优势：

- 精准优化：针对评估中发现的特定失效问题进行改进
- 自适应精调：应对数据分布变化或产品需求演进

- 风险缓解：增量验证优化效果，而非仅依赖一次长时间训练
- 数据高效：将数据采集聚焦在模型表现薄弱的环节
- 课程式训练：使用质量逐步提升的数据进行多轮训练

## 工作原理

### 检查点的位置与访问方式

每次训练作业完成后，系统会在训练配置中 `output_path` 参数指定的输出位置生成一个清单文件。

#### 访问检查点的步骤

- 进入 S3 中指定的 `output_path` 路径
- 下载并解压缩 `output.tar.gz` 文件
- 打开文件内的 `manifest.json` 文件
- 找到 `checkpoint_s3_bucket` 参数，该参数包含已训练模型的 S3 URI

#### manifest.json 结构示例

```
{
  "checkpoint_s3_bucket": "s3://customer-escrow-<account-number>-smtj-<unique-
  identifier>/<job-name>/stepID",
  ...
}
```

### 托管存储桶说明

由于 Amazon Nova 模型权重为专有资产，训练后的模型检查点会存储在 AWS 托管账户内的托管 S3 存储桶中，而非直接复制到您的账户。这些托管存储桶具有以下特点：

- 安全存储您的自定义模型权重
- 可被其他 AWS 服务引用（推理、评估及后续训练作业）
- 仅您的 AWS 账户可通过 IAM 权限访问
- 产生的标准 S3 存储费用计入您的账户（请参阅“成本注意事项”）

您可以在下一轮训练中，将托管存储桶路径作为 `model_name_or_path` 参数的值，以继续进行迭代训练。

## 使用检查点进行迭代训练

配置下一轮训练作业，作为历史检查点的基础模型：

```
run:
  name: "my-iterative-training-job"
  model_type: amazon.nova-2-lite-v1:0:256k
  model_name_or_path: "s3://customer-escrow-<account-number>-smtj-<unique-identifier>/<previous-job-name>"
  data_s3_path: s3://<bucket>/<data-file>.jsonl
  replicas: 4
```

## 建议使用迭代训练的场景

### 理想使用案例

满足以下条件时，建议使用迭代训练：

- 反馈闭环：能够收集真实场景下的失效案例，并系统性解决这些问题
- 动态环境：文档、API 或支持场景持续更新，需定期更新模型
- 评估体系完善：拥有可靠的基准测试和评估框架（见下方示例），可精准衡量模型改进效果
- 机器学习运维能力：有资源管理多轮训练周期及版本控制

### 完善的评估框架示例

- 带通过/失败阈值的自动化基准测试套件
- 包含评分者间信度指标的人工评估流程
- 覆盖边缘场景与对抗性输入的红队测试方案
- 可衡量生产环境影响的 A/B 测试基础设施

### 常见模式

SFT → RFT 流程：最常用的迭代模式：

- 先执行 SFT：通过示例演示教会模型解决问题的方法
- 再执行 RFT：借助奖励信号优化模型在更广问题域的表现

该流程对初始表现较差的模型至关重要：若未先通过 SFT 建立基础的问题解决能力，直接对准确率接近零的模型执行 RFT 便无法提升性能。

## 不建议使用迭代训练的场景

以下情况避免使用迭代训练：

- 任务稳定且定义明确：数据分布固定、需求稳定，模型性能已接近上限
- 简单分类问题：任务单一，单次训练即可满足需求
- 资源限制：缺乏专门的机器学习运维能力，无法管理多轮训练周期
- 收益有限：训练开销过大，而性能提升微乎其微，性价比不足

## 示例 workflows：SFT → RFT

本示例展示推理模型的一种通用迭代训练模式。

### 步骤 1：初始 SFT 训练

使用数据集配置并启动 SFT 训练作业：

```
run:
  name: "initial-sft-training"
  model_type: amazon.nova-2-lite-v1:0:256k
  model_name_or_path: "nova-lite-2/prod"
  data_s3_path: s3://<bucket>/sft-training-data.jsonl
  validation_data_s3_path: s3://<bucket>/sft-validation-data.jsonl
```

理由：SFT 提供额外的示例演示，将模型输出规范为预期的格式和风格，为模型建立基础能力。

训练完成后

- 记录训练作业中配置的 `output_path` 路径
- 从该路径下载 `output.tar.gz` 文件
- 提取并找到 `manifest.json`
- 复制 `checkpoint_s3_bucket` 值

### 步骤 2：基于 SFT 检查点的 RFT 训练

使用 SFT 检查点创建新的 RFT 训练作业：

```
run:
  name: "rft-on-sft-checkpoint"
  model_type: amazon.nova-2-lite-v1:0:256k
  model_name_or_path: "s3://customer-escrow-<account-number>-smtj-<unique-identifier>/<initial-sft-training>"
  data_s3_path: s3://<bucket>/rft-training-data.jsonl
  reward_lambda_arn: <your-reward-function-arn>
```

理由：RFT 训练以 SFT 为基础，让模型基于奖励函数形成更复杂的推理模式并完成性能优化。

### 步骤 3：评估与迭代

对 RFT 检查点执行评测以验证性能：

```
run:
  name: "evaluate-rft-checkpoint"
  model_type: amazon.nova-2-lite-v1:0:256k
  model_name_or_path: "s3://customer-escrow-<account-number>-smtj-<unique-identifier>/<rft-on-sft-checkpoint>"
  data_s3_path: s3://<bucket>/evaluation-data.jsonl
```

若未达到目标指标，调整数据或超参数后继续迭代训练。

#### Important

所有迭代过程中，训练技术（LoRA 对比全秩训练）必须保持一致：

- 若 SFT 采用 LoRA 训练，RFT 也必须使用 LoRA 训练
- 若 SFT 采用全秩训练，RFT 也必须使用全秩训练
- 不可在流程中途切换 LoRA 与全秩训练模式

#### Important

若在 Amazon 拥有的输出 S3 存储桶中使用 KMS 密钥进行加密，则后续所有迭代训练必须使用同一 KMS 密钥。

## 监控迭代进度

您可以通过为训练作业[设置 MLflow](#) 来跟踪各项指标。

### 创建 MLflow 应用程序

使用 Studio UI：如果您通过 Studio UI 创建训练作业，系统会自动创建默认 MLflow 应用程序，并在“高级选项”下默认选择该应用程序。

使用 CLI：如果您使用 CLI，则必须创建一个 MLflow 应用程序，并将其作为输入传递给训练作业 API 请求。

```
mlflow_app_name="<enter your MLflow app name>"
role_arn="<enter your role ARN>"
bucket_name="<enter your bucket name>"
region="<enter your region>"

mlflow_app_arn=$(aws sagemaker create-mlflow-app \
  --name $mlflow_app_name \
  --artifact-store-uri "s3://$bucket_name" \
  --role-arn $role_arn \
  --region $region)
```

### 访问 MLflow 应用程序

使用 CLI：创建预签名 URL 以访问 MLflow 应用程序 UI：

```
aws sagemaker create-presigned-mlflow-app-url \
  --arn $mlflow_app_arn \
  --region $region \
  --output text
```

使用 Studio UI：Studio UI 会展示存储在 MLflow 中的关键指标，并提供指向 MLflow 应用程序 UI 的链接。

### 要跟踪的关键指标

通过迭代监控以下指标，以评测模型优化效果并跟踪作业进度：

#### 针对 SFT

- 训练损失曲线
- 已处理样本数量及处理耗时
- 预留测试集上的性能准确率
- 格式合规性 ( 如有效 JSON 输出率 )
- 领域专属评估数据上的困惑度

### 针对 RFT

- 训练过程中的平均奖励分数
- 奖励分布 ( 高奖励响应占比 )
- 验证奖励趋势 ( 关注过拟合情况 )
- 任务专属成功率 ( 如代码执行通过率、数学题准确率 )

### 一般性问题

- 迭代间基准性能变化
- 代表性样本的人工评估分数
- 生产环境指标 ( 若采用迭代式部署 )

## 停止迭代的判定条件

满足以下任一情况即可停止迭代：

- 性能趋于平稳：继续训练无法显著提升目标指标
- 切换训练技术有效：若某一技术效果趋于平稳，可尝试切换 ( 如 SFT → RFT → SFT ) 以突破性能瓶颈
- 达到目标指标：已满足预设的成功标准
- 检测到性能退化：新迭代导致模型效果下降 ( 参见下文回滚流程 )

有关详细的评估流程，请参阅评估部分。

## 最佳实践

### 从小规模开始，逐步扩展规模

先使用最小数据集和单轮训练 ( epoch ) 验证方案，再逐步扩大规模。这有助于建立信心，并尽早发现问题。

### 设定明确的成功指标

在开始前定义定量与定性指标：

#### 不同使用案例的成功指标示例

- 问答任务：精确匹配准确率、F1 分数、人工偏好评分
- 代码生成：单元测试通过率、编译成功率、执行时间
- 推理任务：步骤准确率、最终答案正确率、奖励分数
- 内容生成：连贯性得分、事实准确性、风格符合度

### 实现自动化评估

设置自动化评估流程，在每轮训练后跟踪性能，支持快速迭代与客观对比。

### 保持严格的版本控制

为每一轮迭代记录以下信息：

- 数据集版本与修改内容
- 模型检查点位置
- 超参数变更
- 性能指标及变化量
- 定性观察结论

这有助于沉淀团队知识，并便于调试。

### 优先关注数据质量而非数量

分析上一轮的失败案例，添加有针对性的高质量示例，而非单纯扩大数据集规模。

## 规划迭代预算

典型迭代次数建议为 3–5 轮：

- 1–2 轮迭代：通常足以完成简单优化或最终精调
- 3–5 轮迭代：适用于需要多轮优化的复杂任务
- 5 轮以上迭代：可能出现边际收益递减，或需要更换方案

根据计算资源预算与性能提升幅度灵活调整。

## 具备回滚能力

若某轮迭代出现性能退化：

- 定位退化原因：对比各检查点的评估指标
- 回退到上一检查点：使用早期检查点的 S3 路径作为 `model_name_or_path`
- 调整训练方案：修改数据、超参数或训练技术后重试
- 记录失败原因：记载导致效果退化的因素，避免重蹈覆辙

## 回滚示例

```
run:
  name: "rollback-to-iteration-2"
  model_type: amazon.nova-2-lite-v1:0:256k
  # Use iteration 2 checkpoint instead of failed iteration 3
  model_name_or_path: "s3://customer-escrow-<account-number>-smtj-<unique-identifier>/
<iteration-2-job-name>"
```

## 成本注意事项

### 检查点存储

- 位置：检查点存储在托管存储桶中，将产生标准 S3 存储费用，计入您的 AWS 账户账单
- 保留：检查点将永久保留，除非手动显式删除
- 管理：可配置生命周期策略，对不再需要的旧检查点进行归档或删除

### 成本优化建议

- 验证新版迭代效果后，及时删除中间过程产生的检查点
- 将需长期保留的检查点归档至 S3 Glacier，降低存储成本
- 根据合规要求与实验需求，设置合理的保留策略

## 限制

### 模型系列一致性

进行迭代训练时，所有迭代轮次必须使用同一模型类型。

#### 初始训练

```
run:
  model_type: amazon.nova-2-lite-v1:0:256k
  model_name_or_path: "nova-lite-2/prod"
```

后续所有迭代必须使用相同的 `model_type`

```
run:
  model_type: amazon.nova-2-lite-v1:0:256k # Must match original
  model_name_or_path: "s3://customer-escrow-  
<account-number>-smtj-  
<unique-identifier>/  
<job-name>"
```

### 训练技术一致性

迭代过程中训练技术必须保持一致：

- 使用 LoRA 训练的模型，只能继续用 LoRA 进行迭代训练
- 使用全秩训练的模型，只能继续用全秩进行迭代训练

#### 迭代训练中 LoRA 适配器的工作机制

- 每一轮 LoRA 训练迭代都会生成新的适配器权重
- 新适配器会替换旧适配器（而非叠加）
- 基础模型保持冻结，仅更新适配器

## 训练技术兼容性矩阵

| 初始训练         | 可迭代的训练方式                  |
|--------------|---------------------------|
| SFT ( 全秩 )   | SFT ( 全秩 )、RFT ( 全秩 )     |
| SFT ( LoRA ) | SFT ( LoRA )、RFT ( LoRA ) |
| RFT ( 全秩 )   | RFT ( 全秩 )                |
| RFT ( LoRA ) | RFT ( LoRA )              |

### 作业启动前的兼容性验证

- 查看上一轮训练配方，确认模型类型与训练技术 ( LoRA/全秩 )
- 确保新配方与上述模型类型和技术完全一致
- 查看 manifest.json 文件，确认检查点路径准确无误

## 问题排查

错误：“Incompatible model training techniques detected”

原因：当前使用的训练技术 ( LoRA/全秩 ) 与检查点所用技术不一致。

解决方法：确保配方使用与原模型相同的训练技术：

- 若检查点基于 LoRA 训练，新配方需使用 LoRA
- 若检查点基于全秩训练，新配方需使用全秩

错误：“Base model for the job extracted from model\_name\_or\_path does not match model\_type”

原因：model\_type 中指定的模型类型，与检查点中的实际模型类型不一致。

解决方法：验证以下内容：

- 配方中的 model\_type 与原模型类型一致

- model\_name\_or\_path 中的检查点 S3 路径准确
- 使用的路径来自正确的 manifest.json 文件

### 正确配置示例

```
run:
  model_type: amazon.nova-2-lite-v1:0:256k # Must match checkpoint's model
  model_name_or_path: "s3://customer-escrow-<account-number>-smtj-<unique-identifier>/
<job-name>"
```

### 错误：“Model configuration not found”

原因：model\_name\_or\_path 中的 S3 路径无效或无法访问。

解决方法：

- 验证 S3 路径是否从 manifest.json 文件中正确复制
- 确保 IAM 角色拥有访问托管存储桶的权限
- 确认上一轮训练作业已成功完成
- 检查路径是否存在拼写错误

### 迭代后性能退化

现象：新训练迭代完成后，评估指标出现下降。

解决方法：

- 回滚：使用上一轮检查点作为基础模型
- 分析：检查失败迭代的训练日志及数据质量
- 调整：修改超参数（降低学习率）、提升数据质量或减少训练轮数（epoch）
- 重试：使用调整后的参数重新启动迭代训练

## SageMaker 推理

自定义 Amazon Nova 模型现已在 SageMaker 推理服务中推出。借助 SageMaker 上的 Amazon Nova，您可以对已训练好的自定义 Nova 模型执行预测（即推理）操作。SageMaker 提供多种 ML 基

基础设施和模型部署选项，以帮助满足您的所有 ML 推理需求。使用 SageMaker 推理，您能够实现模型部署弹性扩缩、在生产环境中更高效地管理模型，并降低运维负担。

SageMaker 支持多种推理方式，例如用于低延迟推理的实时端点，以及用于批量请求的异步端点。通过利用适合您使用案例的推理选项，您可以确保高效的模型部署和推理。有关 SageMaker 推理的更多信息，请[参阅部署模型用于推理](#)。

### Important

SageMaker 推理仅支持全秩自定义模型和已合并 LoRA 的模型。对于未合并 LoRA 的模型及基础模型，请使用 Amazon Bedrock。

## 功能

以下为 Amazon Nova 模型在 SageMaker 推理中支持的功能：

### 模型能力

- 文本生成

### 部署与扩缩

- 支持自定义实例选型的实时端点
- 自动扩缩：根据流量自动调整算力，优化成本与 GPU 利用率。有关更多信息，请[参阅自动扩缩 Amazon SageMaker 模型](#)。
- 支持流式 API，可实时生成词元

### 监控与优化

- 集成 Amazon CloudWatch，用于监控与提醒
- 通过 VPC 配置实现可用区域感知的时延优化

### 开发工具

- 支持 AWS CLI：有关更多信息，请[参阅 SageMaker AWS CLI 命令参考](#)。
- 支持通过 SDK 集成笔记本

## 支持的模型与实例

创建 SageMaker 推理端点时，可设置两个环境变量来配置部署：CONTEXT\_LENGTH 和 MAX\_CONCURRENCY。

- CONTEXT\_LENGTH：单个请求的词元最大总长（输入 + 输出）
- MAX\_CONCURRENCY：端点可处理的最大并发请求数

下表列出了支持的 Amazon Nova 模型、实例类型及对应配置。MAX\_CONCURRENCY 数值表示在每个 CONTEXT\_LENGTH 设置下所支持的最大并发数：

| 模型                   | 实例类型           | 支持的配置                                                                                                    |
|----------------------|----------------|----------------------------------------------------------------------------------------------------------|
| Amazon Nova<br>Micro | ml.g5.12xlarge | CONTEXT_LENGTH : 400<br>0 , MAX_CONCURRENCY : 32<br><br>CONTEXT_LENGTH : 800<br>0 , MAX_CONCURRENCY : 16 |
|                      | ml.g5.24xlarge | CONTEXT_LENGTH : 800<br>0 , MAX_CONCURRENCY : 32                                                         |
|                      | ml.g6.12xlarge | CONTEXT_LENGTH : 400<br>0 , MAX_CONCURRENCY : 32<br><br>CONTEXT_LENGTH : 800<br>0 , MAX_CONCURRENCY : 16 |
|                      |                | ml.g6.24xlarge                                                                                           |
|                      | ml.g6.48xlarge | CONTEXT_LENGTH : 800<br>0 , MAX_CONCURRENCY : 32                                                         |
|                      | ml.p5.48xlarge | CONTEXT_LENGTH : 800<br>0 , MAX_CONCURRENCY : 32<br><br>CONTEXT_LENGTH : 160<br>00 , MAX_CONCURRENCY : 2 |

| 模型               | 实例类型           | 支持的配置                                                                                                                                                |
|------------------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
|                  |                | CONTEXT_LENGTH : 24000 , MAX_CONCURRENCY : 1                                                                                                         |
| Amazon Nova Lite | ml.g6.48xlarge | CONTEXT_LENGTH : 4000 , MAX_CONCURRENCY : 32<br><br>CONTEXT_LENGTH : 8000 , MAX_CONCURRENCY : 16                                                     |
|                  | ml.p5.48xlarge | CONTEXT_LENGTH : 8000 , MAX_CONCURRENCY : 32<br><br>CONTEXT_LENGTH : 16000 , MAX_CONCURRENCY : 2<br><br>CONTEXT_LENGTH : 24000 , MAX_CONCURRENCY : 1 |
| Nova 2 Lite      | ml.p5.48xlarge | CONTEXT_LENGTH : 8000 , MAX_CONCURRENCY : 32<br><br>CONTEXT_LENGTH : 16000 , MAX_CONCURRENCY : 2<br><br>CONTEXT_LENGTH : 24000 , MAX_CONCURRENCY : 1 |

### Note

表格中显示的 MAX\_CONCURRENCY 数值是对应 CONTEXT\_LENGTH 设置的上限值。您可以在相同并发数下使用更短的上下文长度，但超出这些上限值将导致 SageMaker 端点创建失败。

以 Amazon Nova Micro 搭配 ml.g5.12xlarge 为例：

- CONTEXT\_LENGTH=2000 , MAX\_CONCURRENCY=32 → 有效
- CONTEXT\_LENGTH=8000 , MAX\_CONCURRENCY=32 → 无效 ( 上下文长度 8000 时并发上限为 16 )

- CONTEXT\_LENGTH=8000 , MAX\_CONCURRENCY=4 → 有效
- CONTEXT\_LENGTH=8000 , MAX\_CONCURRENCY=16 → 有效
- CONTEXT\_LENGTH=10000 → 无效 ( 该实例最大上下文长度为 8000 )

## 支持的 AWS 区域

下表列出了 Amazon Nova 模型可在 SageMaker 推理中使用的 AWS 区域：

| 区域名称             | 区域代码      | 可用性 |
|------------------|-----------|-----|
| 美国东部 ( 弗吉尼亚州北部 ) | us-east-1 | 可用  |
| 美国西部 ( 俄勒冈州 )    | us-west-2 | 可用  |

## 支持的区域与容器映像

下表按区域列出了 Amazon Nova 模型在 SageMaker 推理中使用的容器映像 URI。每个区域提供两种映像标签：版本化标签 (v1.0.0) 和最新标签 (SM-Inference-latest)。生产环境部署时，建议使用版本化标签。

| 区域        | 容器映像 URI                                                                             |
|-----------|--------------------------------------------------------------------------------------|
| us-east-1 | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-inference-repo:SM-Inference-latest |
| us-west-2 | 176779409107.dkr.ecr.us-west-2.amazonaws.com/nova-inference-repo:SM-Inference-latest |

## 最佳实践

有关在 SageMaker 上部署和管理模型的最佳实践，请参阅 [SageMaker 最佳实践](#)。

## 支持

如在 SageMaker 推理中使用 Amazon Nova 模型时遇到问题或需要支持，可通过控制台或 AWS 客户经理联系 AWS Support。

### 主题

- [入门](#)
- [API 参考](#)
- [评估在 SageMaker 推理上托管的模型](#)
- [在 Amazon SageMaker Inference 滥用检测中部署 Amazon Nova Forge 模型](#)

## 入门

本指南介绍如何在 SageMaker 实时端点上部署自定义 Amazon Nova 模型、配置推理参数，并调用模型进行测试。

### 先决条件

在 SageMaker 推理上部署 Amazon Nova 模型需满足以下先决条件：

- 创建 AWS 账户 账户：如尚无账户，请参阅[创建 AWS 账户](#)。
- 所需 IAM 权限：确保 IAM 用户或角色已附加以下托管策略：
  - AmazonSageMakerFullAccess
  - AmazonS3FullAccess
- 所需 SDK/CLI 版本：以下 SDK 版本已在 SageMaker 推理上通过 Amazon Nova 模型的测试与验证：
  - 适用于基于资源的 API 方式：SageMaker Python SDK v3.0.0+ (sagemaker>=3.0.0)
  - 适用于直接 API 调用：Boto3 版本 1.35.0+ (boto3>=1.35.0)。本指南中的示例均采用此方式。
- 增加服务配额：针对您计划用于 SageMaker Inference 端点（例如 ml.p5.48xlarge for endpoint usage）的机器学习实例类型，请求增加 Amazon SageMaker 服务配额。有关受支持实例类型的列表，请参阅 [the section called “支持的模型与实例”](#)。如需申请提升配额，请参阅[请求增加配额](#)。有关 SageMaker 实例配额的更多信息，请参阅 [SageMaker 端点和配额](#)。

### 步骤 1：配置 AWS 凭证

使用以下方法之一管理配置 AWS 凭证：

## 选项 1 : AWS CLI ( 建议 )

```
aws configure
```

提示后，输入 AWS 访问密钥 ID、私有密钥和默认区域名称。

## 选项 2 : AWS 凭证文件

创建或编辑 `~/.aws/credentials` :

```
[default]
aws_access_key_id = YOUR_ACCESS_KEY
aws_secret_access_key = YOUR_SECRET_KEY
```

## 选项 3 : 环境变量

```
export AWS_ACCESS_KEY_ID=your_access_key
export AWS_SECRET_ACCESS_KEY=your_secret_key
```

### Note

有关 AWS 凭证的更多信息，请参阅[配置与凭证文件设置](#)。

## 初始化 AWS 客户端

创建包含以下代码的 Python 脚本或笔记本文件，用于初始化 AWS SDK 并验证凭证：

```
import boto3

# AWS Configuration - Update these for your environment
REGION = "us-east-1" # Supported regions: us-east-1, us-west-2
AWS_ACCOUNT_ID = "YOUR_ACCOUNT_ID" # Replace with your AWS account ID

# Initialize AWS clients using default credential chain
sagemaker = boto3.client('sagemaker', region_name=REGION)
sts = boto3.client('sts')

# Verify credentials
try:
```

```
identity = sts.get_caller_identity()
print(f"Successfully authenticated to AWS Account: {identity['Account']}")

if identity['Account'] != AWS_ACCOUNT_ID:
    print(f"Warning: Connected to account {identity['Account']}, expected
{AWS_ACCOUNT_ID}")

except Exception as e:
    print(f"Failed to authenticate: {e}")
    print("Please verify your credentials are configured correctly.")
```

如果身份验证成功，您会看到输出结果中包含自己的 AWS 账户 ID 确认信息。

## 步骤 2：创建 SageMaker 执行角色

SageMaker 执行角色是一种 IAM 角色，用于授予 SageMaker 权限，使其能够代表您访问 AWS 资源（例如存储模型构件的 Amazon S3 存储桶、用于日志记录的 CloudWatch）。

### 创建执行角色

#### Note

创建 IAM 角色需要具备 `iam:CreateRole` 和 `iam:AttachRolePolicy` 权限。在继续操作前，确保自己的 IAM 用户或角色已拥有这些权限。

以下代码将创建一个具备部署 Amazon Nova 自定义模型所需权限的 IAM 角色：

```
import json

# Create SageMaker Execution Role
role_name = f"SageMakerInference-ExecutionRole-{AWS_ACCOUNT_ID}"

trust_policy = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {"Service": "sagemaker.amazonaws.com"},
            "Action": "sts:AssumeRole"
        }
    ]
}
```

```
    ]
}

iam = boto3.client('iam', region_name=REGION)

# Create the role
role_response = iam.create_role(
    RoleName=role_name,
    AssumeRolePolicyDocument=json.dumps(trust_policy),
    Description='SageMaker execution role with S3 and SageMaker access'
)

# Attach required policies
iam.attach_role_policy(
    RoleName=role_name,
    PolicyArn='arn:aws:iam::aws:policy/AmazonSageMakerFullAccess'
)

iam.attach_role_policy(
    RoleName=role_name,
    PolicyArn='arn:aws:iam::aws:policy/AmazonS3FullAccess'
)

SAGEMAKER_EXECUTION_ROLE_ARN = role_response['Role']['Arn']
print(f"Created SageMaker execution role: {SAGEMAKER_EXECUTION_ROLE_ARN}")
```

## 使用现有执行角色 ( 可选 )

如果已有 SageMaker 执行角色，可直接复用：

```
# Replace with your existing role ARN
SAGEMAKER_EXECUTION_ROLE_ARN = "arn:aws:iam::YOUR_ACCOUNT_ID:role/
YOUR_EXISTING_ROLE_NAME"
```

## 查找账户中现有的 SageMaker 角色：

```
iam = boto3.client('iam', region_name=REGION)
response = iam.list_roles()
sagemaker_roles = [role for role in response['Roles'] if 'SageMaker' in
    role['RoleName']]
for role in sagemaker_roles:
    print(f"{role['RoleName']}: {role['Arn']}")
```

**⚠ Important**

执行角色必须与 `sagemaker.amazonaws.com` 建立信任关系，且拥有访问 Amazon S3 和 SageMaker 资源的权限。

有关 SageMaker 执行角色的更多信息，请参阅 [SageMaker 角色](#)。

### 步骤 3：配置模型参数

为您的 Amazon Nova 模型配置部署参数。这些设置将控制模型行为、资源分配及推理特性。有关支持的实例类型，以及各实例对应的 `CONTEXT_LENGTH` 和 `MAX_CONCURRENCY` 取值，请参阅 [the section called “支持的模型与实例”](#)。

#### 必填参数

- `IMAGE`：Amazon Nova 推理容器的 Docker 容器映像 URI。该地址由 AWS 提供。
- `CONTEXT_LENGTH`：模型上下文长度。
- `MAX_CONCURRENCY`：每轮迭代的最大序列数；用于限制 GPU 上单批次可并发处理的独立用户请求（提示词）数量。取值范围：大于 0 的整数。

#### 可选生成参数

- `DEFAULT_TEMPERATURE`：控制生成内容的随机性。取值范围：0.0 到 2.0（0.0 = 确定性生成，数值越高随机性越强）。
- `DEFAULT_TOP_P`：核采样词元选择阈值。取值范围：1e-10 到 1.0。
- `DEFAULT_TOP_K`：将词元选择范围限制为概率最高的前 K 个词元。取值范围：大于等于 -1 的整数（-1 = 无限制）。
- `DEFAULT_MAX_NEW_TOKENS`：：响应中生成的最大词元数（即最大输出词元数）。取值范围：大于等于 1 的整数。
- `DEFAULT_LOGPROBS`：每个词元返回的对数概率数量。取值范围：1 到 20 的整数。

#### 配置部署

```
# AWS Configuration
REGION = "us-east-1" # Must match region from Step 1
```

```
# ECR Account mapping by region
ECR_ACCOUNT_MAP = {
    "us-east-1": "708977205387",
    "us-west-2": "176779409107"
}

# Container Image
IMAGE = f"{ECR_ACCOUNT_MAP[REGION]}.dkr.ecr.{REGION}.amazonaws.com/nova-inference-
repo:SM-Inference-latest"
print(f"IMAGE = {IMAGE}")

# Model Parameters
CONTEXT_LENGTH = "16000"          # Maximum total context length
MAX_CONCURRENCY = "2"            # Maximum concurrent sequences

# Optional: Default generation parameters (uncomment to use)
DEFAULT_TEMPERATURE = "0.0"      # Deterministic output
DEFAULT_TOP_P = "1.0"           # Consider all tokens
# DEFAULT_TOP_K = "50"          # Uncomment to limit to top 50 tokens
# DEFAULT_MAX_NEW_TOKENS = "2048" # Uncomment to set max output tokens
# DEFAULT_LOGPROBS = "1"        # Uncomment to enable log probabilities

# Build environment variables for the container
environment = {
    'CONTEXT_LENGTH': CONTEXT_LENGTH,
    'MAX_CONCURRENCY': MAX_CONCURRENCY,
}

# Add optional parameters if defined
if 'DEFAULT_TEMPERATURE' in globals():
    environment['DEFAULT_TEMPERATURE'] = DEFAULT_TEMPERATURE
if 'DEFAULT_TOP_P' in globals():
    environment['DEFAULT_TOP_P'] = DEFAULT_TOP_P
if 'DEFAULT_TOP_K' in globals():
    environment['DEFAULT_TOP_K'] = DEFAULT_TOP_K
if 'DEFAULT_MAX_NEW_TOKENS' in globals():
    environment['DEFAULT_MAX_NEW_TOKENS'] = DEFAULT_MAX_NEW_TOKENS
if 'DEFAULT_LOGPROBS' in globals():
    environment['DEFAULT_LOGPROBS'] = DEFAULT_LOGPROBS

print("Environment configuration:")
for key, value in environment.items():
    print(f" {key}: {value}")
```

## 配置部署专属参数

接下来为您的 Amazon Nova 模型部署配置专属参数，包括模型构件存储位置及实例类型选择。

### 设置部署标识符

```
# Deployment identifier - use a descriptive name for your use case
JOB_NAME = "my-nova-deployment"
```

### 指定模型构件存储位置

提供训练完成的 Amazon Nova 模型构件所在的 Amazon S3 URI。该地址应为模型训练或微调作业的输出位置。

```
# S3 location of your trained Nova model artifacts
# Replace with your model's S3 URI - must end with /
MODEL_S3_LOCATION = "s3://your-bucket-name/path/to/model/artifacts/"
```

### 选择模型变体与实例类型

```
# Configure model variant and instance type
TESTCASE = {
    "model": "lite2",          # Options: micro, lite, lite2
    "instance": "ml.p5.48xlarge" # Refer to "Supported models and instances" section
}

# Generate resource names
INSTANCE_TYPE = TESTCASE["instance"]
MODEL_NAME = JOB_NAME + "-" + TESTCASE["model"] + "-" + INSTANCE_TYPE.replace(".", "-")
ENDPOINT_CONFIG_NAME = MODEL_NAME + "-Config"
ENDPOINT_NAME = MODEL_NAME + "-Endpoint"

print(f"Model Name: {MODEL_NAME}")
print(f"Endpoint Config: {ENDPOINT_CONFIG_NAME}")
print(f"Endpoint Name: {ENDPOINT_NAME}")
```

### 命名规范

代码会自动为 AWS 资源生成统一的命名：

- 模型名称：{JOB\_NAME}-{model}-{instance-type}

- 端点配置：{MODEL\_NAME}-Config
- 端点名称：{MODEL\_NAME}-Endpoint

## 步骤 4：创建 SageMaker 模型与端点配置

本步骤中，您将创建两项核心资源：一是关联 Amazon Nova 模型构件的 SageMaker 模型对象，二是定义模型部署方式的端点配置。

**SageMaker 模型**：封装推理容器映像、模型构件存储位置及环境配置的模型对象。该资源可复用，能部署到多个端点。

**端点配置**：定义部署的基础设施设置，包括实例类型、实例数量、模型变体。通过该配置，您可将部署设置与模型本身分开管理。

### 创建 SageMaker 模型

以下代码将创建一个关联您 Amazon Nova 模型构件的 SageMaker 模型：

```
try:
    model_response = sagemaker.create_model(
        ModelName=MODEL_NAME,
        PrimaryContainer={
            'Image': IMAGE,
            'ModelDataSource': {
                'S3DataSource': {
                    'S3Uri': MODEL_S3_LOCATION,
                    'S3DataType': 'S3Prefix',
                    'CompressionType': 'None'
                }
            },
            'Environment': environment
        },
        ExecutionRoleArn=SAGEMAKER_EXECUTION_ROLE_ARN,
        EnableNetworkIsolation=True
    )
    print("Model created successfully!")
    print(f"Model ARN: {model_response['ModelArn']}")

except sagemaker.exceptions.ClientError as e:
    print(f"Error creating model: {e}")
```

关键参数：

- `ModelName` : 模型的唯一标识符
- `Image` : Amazon Nova 推理所用的 Docker 容器映像 URI
- `ModelDataSource` : 模型构件的 Amazon S3 存储位置
- `Environment` : 步骤 3 中配置的环境变量
- `ExecutionRoleArn` : 步骤 2 中创建的 IAM 角色 ARN
- `EnableNetworkIsolation` : 设为 `True` 可增强安全性 ( 禁止容器发起出站网络请求 )

## 创建端点配置

接下来，创建定义部署基础设施的端点配置：

```
# Create Endpoint Configuration
try:
    production_variant = {
        'VariantName': 'primary',
        'ModelName': MODEL_NAME,
        'InitialInstanceCount': 1,
        'InstanceType': INSTANCE_TYPE,
    }

    config_response = sagemaker.create_endpoint_config(
        EndpointConfigName=ENDPOINT_CONFIG_NAME,
        ProductionVariants=[production_variant]
    )
    print("Endpoint configuration created successfully!")
    print(f"Config ARN: {config_response['EndpointConfigArn']}")

except sagemaker.exceptions.ClientError as e:
    print(f"Error creating endpoint configuration: {e}")
```

关键参数：

- `VariantName` : 该模型变体的标识符 ( 单模型部署时使用“primary” )
- `ModelName` : 关联上文创建的模型
- `InitialInstanceCount` : 部署的实例数量 ( 初始可设为 1 , 后续则按需扩缩 )
- `InstanceType` : 步骤 3 中选定的机器学习实例类型

## 验证资源创建

您可验证资源是否创建成功：

```
# Describe the model
model_info = sagemaker.describe_model(ModelName=MODEL_NAME)
print(f"Model Status: {model_info['ModelName']} created")

# Describe the endpoint configuration
config_info =
    sagemaker.describe_endpoint_config(EndpointConfigName=ENDPOINT_CONFIG_NAME)
print(f"Endpoint Config Status: {config_info['EndpointConfigName']} created")
```

## 步骤 5：部署端点

下一步，您将通过创建 SageMaker 实时端点来部署 Amazon Nova 模型。该端点将托管您的模型，并提供安全的 HTTPS 端点用于发起推理请求。

端点创建通常需要 15–30 分钟，此过程中 AWS 会完成基础设施配置、模型构件下载及推理容器初始化。

### 创建端点

```
import time

try:
    endpoint_response = sagemaker.create_endpoint(
        EndpointName=ENDPOINT_NAME,
        EndpointConfigName=ENDPOINT_CONFIG_NAME
    )
    print("Endpoint creation initiated successfully!")
    print(f"Endpoint ARN: {endpoint_response['EndpointArn']}")
except Exception as e:
    print(f"Error creating endpoint: {e}")
```

### 监控端点创建进度

以下代码会轮询端点状态，直至部署完成：

```
# Monitor endpoint creation progress
print("Waiting for endpoint creation to complete...")
print("This typically takes 15-30 minutes...\n")

while True:
    try:
```

```
response = sagemaker.describe_endpoint(EndpointName=ENDPOINT_NAME)
status = response['EndpointStatus']

if status == 'Creating':
    print(f"# Status: {status} - Provisioning infrastructure and loading
model...")
elif status == 'InService':
    print(f"# Status: {status}")
    print("\nEndpoint creation completed successfully!")
    print(f"Endpoint Name: {ENDPOINT_NAME}")
    print(f"Endpoint ARN: {response['EndpointArn']}")
    break
elif status == 'Failed':
    print(f"# Status: {status}")
    print(f"Failure Reason: {response.get('FailureReason', 'Unknown')}")
    print("\nFull response:")
    print(response)
    break
else:
    print(f"Status: {status}")

except Exception as e:
    print(f"Error checking endpoint status: {e}")
    break

time.sleep(30) # Check every 30 seconds
```

## 验证端点就绪状态

当端点状态变为 InService 后，您可验证其配置信息：

```
# Get detailed endpoint information
endpoint_info = sagemaker.describe_endpoint(EndpointName=ENDPOINT_NAME)

print("\n=== Endpoint Details ===")
print(f"Endpoint Name: {endpoint_info['EndpointName']}")
print(f"Endpoint ARN: {endpoint_info['EndpointArn']}")
print(f"Status: {endpoint_info['EndpointStatus']}")
print(f"Creation Time: {endpoint_info['CreationTime']}")
print(f>Last Modified: {endpoint_info['LastModifiedTime']}")

# Get endpoint config for instance type details
endpoint_config_name = endpoint_info['EndpointConfigName']
```

```
endpoint_config =
    sagemaker.describe_endpoint_config(EndpointConfigName=endpoint_config_name)

# Display production variant details
for variant in endpoint_info['ProductionVariants']:
    print(f"\nProduction Variant: {variant['VariantName']}")
    print(f"  Current Instance Count: {variant['CurrentInstanceCount']}")
    print(f"  Desired Instance Count: {variant['DesiredInstanceCount']}")
    # Get instance type from endpoint config
    for config_variant in endpoint_config['ProductionVariants']:
        if config_variant['VariantName'] == variant['VariantName']:
            print(f"  Instance Type: {config_variant['InstanceType']}")
            break
```

## 排查端点创建失败问题

常见失败原因：

- 容量不足：请求的实例类型在您的区域暂不可用
  - 解决方法：尝试更换实例类型，或提交配额提升申请
- IAM 权限问题：执行角色缺少必要权限
  - 解决方法：验证该角色是否拥有访问 Amazon S3 模型构件的权限，以及 SageMaker 相关必要权限
- 未找到模型构件：Amazon S3 URI 错误或无法访问
  - 解决方法：验证 Amazon S3 URI 正确性，检查存储桶权限，并确认当前操作的区域正确
- 资源限额超限：账户的端点/实例数量超出服务限额
  - 解决方法：通过“服务配额”或 AWS Support 提交配额提升申请

### Note

若需删除创建失败的端点并重新部署：

```
sagemaker.delete_endpoint(EndpointName=ENDPOINT_NAME)
```

## 步骤 6：调用端点

当端点状态变为 InService 后，即可发送推理请求，通过 Amazon Nova 模型生成预测结果。SageMaker 支持两类端点调用方式：同步端点（含流式/非流式模式的实时调用）和异步端点（基于 Amazon S3 的批量处理）。

### 设置运行时客户端

创建带有合理超时设置的 SageMaker Runtime 客户端：

```
import json
import boto3
import botocore
from botocore.exceptions import ClientError

# Configure client with appropriate timeouts
config = botocore.config.Config(
    read_timeout=120,      # Maximum time to wait for response
    connect_timeout=10,   # Maximum time to establish connection
    retries={'max_attempts': 3} # Number of retry attempts
)

# Create SageMaker Runtime client
runtime_client = boto3.client('sagemaker-runtime', config=config, region_name=REGION)
```

### 编写通用推理函数

以下函数可同时处理流式与非流式请求：

```
def invoke_nova_endpoint(request_body):
    """
    Invoke Nova endpoint with automatic streaming detection.

    Args:
        request_body (dict): Request payload containing prompt and parameters

    Returns:
        dict: Response from the model (for non-streaming requests)
        None: For streaming requests (prints output directly)
    """
    body = json.dumps(request_body)
    is_streaming = request_body.get("stream", False)
```

```

try:
    print(f"Invoking endpoint ({'streaming' if is_streaming else 'non-streaming'})...")

    if is_streaming:
        response = runtime_client.invoke_endpoint_with_response_stream(
            EndpointName=ENDPOINT_NAME,
            ContentType='application/json',
            Body=body
        )

        event_stream = response['Body']
        for event in event_stream:
            if 'PayloadPart' in event:
                chunk = event['PayloadPart']
                if 'Bytes' in chunk:
                    data = chunk['Bytes'].decode()
                    print("Chunk:", data)
            else:
                # Non-streaming inference
                response = runtime_client.invoke_endpoint(
                    EndpointName=ENDPOINT_NAME,
                    ContentType='application/json',
                    Accept='application/json',
                    Body=body
                )

                response_body = response['Body'].read().decode('utf-8')
                result = json.loads(response_body)
                print("# Response received successfully")
                return result

except ClientError as e:
    error_code = e.response['Error']['Code']
    error_message = e.response['Error']['Message']
    print(f"# AWS Error: {error_code} - {error_message}")
except Exception as e:
    print(f"# Unexpected error: {str(e)}")

```

## 示例 1：非流式对话补全

采用对话格式实现多轮交互：

```
# Non-streaming chat request
```

```
chat_request = {
  "messages": [
    {"role": "user", "content": "Hello! How are you?"}
  ],
  "max_tokens": 100,
  "max_completion_tokens": 100, # Alternative to max_tokens
  "stream": False,
  "temperature": 0.7,
  "top_p": 0.9,
  "top_k": 50,
  "logprobs": True,
  "top_logprobs": 3,
  "reasoning_effort": "low", # Options: "low", "high"
  "allowed_token_ids": None, # List of allowed token IDs
  "truncate_prompt_tokens": None, # Truncate prompt to this many tokens
  "stream_options": None
}

response = invoke_nova_endpoint(chat_request)
```

示例响应：

```
{
  "id": "chatcpl-123456",
  "object": "chat.completion",
  "created": 1234567890,
  "model": "default",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "Hello! I'm doing well, thank you for asking. I'm here and ready to help you with any questions or tasks you might have. How can I assist you today?"
      },
      "logprobs": {
        "content": [
          {
            "token": "Hello",
            "logprob": -0.123,
            "top_logprobs": [
              {"token": "Hello", "logprob": -0.123},
```

```

        {"token": "Hi", "logprob": -2.456},
        {"token": "Hey", "logprob": -3.789}
    ]
}
# Additional tokens...
]
},
"finish_reason": "stop"
}
],
"usage": {
    "prompt_tokens": 12,
    "completion_tokens": 28,
    "total_tokens": 40
}
}

```

## 示例 2：简单文本补全

采用补全格式实现基础文本生成：

```

# Simple completion request
completion_request = {
    "prompt": "The capital of France is",
    "max_tokens": 50,
    "stream": False,
    "temperature": 0.0,
    "top_p": 1.0,
    "top_k": -1, # -1 means no limit
    "logprobs": 3, # Number of log probabilities to return
    "allowed_token_ids": None, # List of allowed token IDs
    "truncate_prompt_tokens": None, # Truncate prompt to this many tokens
    "stream_options": None
}

response = invoke_nova_endpoint(completion_request)

```

示例响应：

```

{
    "id": "cml-789012",
    "object": "text_completion",
    "created": 1234567890,

```

```

"model": "default",
"choices": [
  {
    "text": " Paris.",
    "index": 0,
    "logprobs": {
      "tokens": [" Paris", "."],
      "token_logprobs": [-0.001, -0.002],
      "top_logprobs": [
        {" Paris": -0.001, " London": -5.234, " Rome": -6.789},
        {".": -0.002, ",": -4.567, "!": -7.890}
      ]
    },
    "finish_reason": "stop"
  }
],
"usage": {
  "prompt_tokens": 6,
  "completion_tokens": 2,
  "total_tokens": 8
}
}

```

### 示例 3：流式对话补全

```

# Streaming chat request
streaming_request = {
  "messages": [
    {"role": "user", "content": "Tell me a short story about a robot"}
  ],
  "max_tokens": 200,
  "stream": True,
  "temperature": 0.7,
  "top_p": 0.95,
  "top_k": 40,
  "logprobs": True,
  "top_logprobs": 2,
  "reasoning_effort": "high", # For more detailed reasoning
  "stream_options": {"include_usage": True}
}

invoke_nova_endpoint(streaming_request)

```

## 流式传输输出示例：

```

Chunk: data: {"id":"chatcpl-029ca032-fa01-4868-80b7-
c4cb1af90fb9","object":"chat.completion.chunk","created":1772060532,"model":"default","choices":
[{"index":0,"delta":
{"role":"assistant","content":"","reasoning_content":null},"logprobs":null,"finish_reason":null}]}
Chunk: data: {"id":"chatcpl-029ca032-fa01-4868-80b7-
c4cb1af90fb9","object":"chat.completion.chunk","created":1772060532,"model":"default","choices":
[{"index":0,"delta":{"content":" Once","reasoning_content":null},"logprobs":
{"content":[{"token":"\u2581Once","logprob":-0.6078429222106934,"bytes":
[226,150,129,79,110,99,101],"top_logprobs":
[{"token":"\u2581Once","logprob":-0.6078429222106934,"bytes":
[226,150,129,79,110,99,101]},{token":"\u2581In","logprob":-0.7864127159118652,"bytes":
[226,150,129,73,110]}]}]},"finish_reason":null,"token_ids":null}]}
Chunk: data: {"id":"chatcpl-029ca032-fa01-4868-80b7-
c4cb1af90fb9","object":"chat.completion.chunk","created":1772060532,"model":"default","choices":
[{"index":0,"delta":{"content":" upon","reasoning_content":null},"logprobs":
{"content":[{"token":"\u2581upon","logprob":-0.0012345,"bytes":
[226,150,129,117,112,111,110],"top_logprobs":
[{"token":"\u2581upon","logprob":-0.0012345,"bytes":[226,150,129,117,112,111,110]}],
{"token":"\u2581a","logprob":-6.789,"bytes":
[226,150,129,97]}]}]},"finish_reason":null,"token_ids":null}]}
Chunk: data: {"id":"chatcpl-029ca032-fa01-4868-80b7-
c4cb1af90fb9","object":"chat.completion.chunk","created":1772060532,"model":"default","choices":
[{"index":0,"delta":{"content":" a","reasoning_content":null},"logprobs":
{"content":[{"token":"\u2581a","logprob":-0.0001234,"bytes":
[226,150,129,97],"top_logprobs":[{"token":"\u2581a","logprob":-0.0001234,"bytes":
[226,150,129,97]},{token":"\u2581time","logprob":-9.123,"bytes":
[226,150,129,116,105,109,101]}]}]},"finish_reason":null,"token_ids":null}]}
Chunk: data: {"id":"chatcpl-029ca032-fa01-4868-80b7-
c4cb1af90fb9","object":"chat.completion.chunk","created":1772060532,"model":"default","choices":
[{"index":0,"delta":{"content":" time","reasoning_content":null},"logprobs":
{"content":[{"token":"\u2581time","logprob":-0.0023456,"bytes":
[226,150,129,116,105,109,101],"top_logprobs":
[{"token":"\u2581time","logprob":-0.0023456,"bytes":[226,150,129,116,105,109,101]}],
{"token":"","logprob":-6.012,"bytes":
[44]}]}]},"finish_reason":null,"token_ids":null}]}

# Additional chunks...

Chunk: data: {"id":"chatcpl-029ca032-fa01-4868-80b7-
c4cb1af90fb9","object":"chat.completion.chunk","created":1772060532,"model":"default","choices":
[{"index":0,"delta":{},"logprobs":null,"finish_reason":"stop"}],"usage":
{"prompt_tokens":15,"completion_tokens":87,"total_tokens":102}}

```

```
Chunk: data: [DONE]
```

#### 示例 4：多模态对话补全

采用多模态格式处理图像与文本混合输入：

```
# Multimodal chat request (if supported by your model)
multimodal_request = {
  "messages": [
    {
      "role": "user",
      "content": [
        {"type": "text", "text": "What's in this image?"},
        {"type": "image_url", "image_url": {"url": "data:image/
jpeg;base64,..."}}
      ]
    }
  ],
  "max_tokens": 150,
  "temperature": 0.3,
  "top_p": 0.8,
  "stream": False
}

response = invoke_nova_endpoint(multimodal_request)
```

示例响应：

```
{
  "id": "chatcpl-345678",
  "object": "chat.completion",
  "created": 1234567890,
  "model": "default",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "The image shows..."
      },
      "finish_reason": "stop"
    }
  ],
}
```

```
"usage": {
  "prompt_tokens": 1250,
  "completion_tokens": 45,
  "total_tokens": 1295
}
```

## 步骤 7：清理资源（可选）

为避免产生不必要的费用，请删除本教程中创建的 AWS 资源。SageMaker 端点在运行期间会持续计费，即使您未主动发起推理请求。

### Important

删除资源是永久操作，无法撤销。继续操作之前，确认不再需要这些资源。

## 删除端点

```
import boto3

# Initialize SageMaker client
sagemaker = boto3.client('sagemaker', region_name=REGION)

try:
    print("Deleting endpoint...")
    sagemaker.delete_endpoint(EndpointName=ENDPOINT_NAME)
    print(f"# Endpoint '{ENDPOINT_NAME}' deletion initiated")
    print("Charges will stop once deletion completes (typically 2-5 minutes)")
except Exception as e:
    print(f"# Error deleting endpoint: {e}")
```

### Note

端点删除为异步操作。您可以监控删除状态：

```
import time

print("Monitoring endpoint deletion...")
while True:
    try:
        response = sagemaker.describe_endpoint(EndpointName=ENDPOINT_NAME)
```

```
status = response['EndpointStatus']
print(f"Status: {status}")
time.sleep(10)
except sagemaker.exceptions.ClientError as e:
    if e.response['Error']['Code'] == 'ValidationException':
        print("# Endpoint successfully deleted")
        break
    else:
        print(f"Error: {e}")
        break
```

## 删除现有端点配置

端点删除完成后，删除端点配置：

```
try:
    print("Deleting endpoint configuration...")
    sagemaker.delete_endpoint_config(EndpointConfigName=ENDPOINT_CONFIG_NAME)
    print(f"# Endpoint configuration '{ENDPOINT_CONFIG_NAME}' deleted")
except Exception as e:
    print(f"# Error deleting endpoint configuration: {e}")
```

## 删除模型

删除 SageMaker 模型对象：

```
try:
    print("Deleting model...")
    sagemaker.delete_model(ModelName=MODEL_NAME)
    print(f"# Model '{MODEL_NAME}' deleted")
except Exception as e:
    print(f"# Error deleting model: {e}")
```

## API 参考

SageMaker 上的 Amazon Nova 模型，使用标准 SageMaker Runtime API 进行推理。有关完整的 API 文档，请参阅[测试已部署模型](#)。

## 端点调用

SageMaker 上的 Amazon Nova 模型支持两种调用方式：

- 同步调用：使用 [InvokeEndpoint API](#) 处理实时、非流式推理请求。
- 流式调用：使用 [InvokeEndpointWithResponseStream API](#) 处理实时流式推理请求。

## 请求格式

Amazon Nova 模型支持三种请求格式：

### 对话补全格式

该格式用于对话交互：

```
{
  "messages": [
    {"role": "user", "content": "string"}
  ],
  "max_tokens": integer,
  "max_completion_tokens": integer,
  "stream": boolean,
  "temperature": float,
  "top_p": float,
  "top_k": integer,
  "logprobs": boolean,
  "top_logprobs": integer,
  "reasoning_effort": "low" | "high",
  "allowed_token_ids": [integer],
  "truncate_prompt_tokens": integer,
  "stream_options": {
    "include_usage": boolean
  }
}
```

### 文本补全格式

该格式用于简单文本生成：

```
{
  "prompt": "string",
  "max_tokens": integer,
  "stream": boolean,
  "temperature": float,
  "top_p": float,
```

```

"top_k": integer,
"logprobs": integer,
"allowed_token_ids": [integer],
"truncate_prompt_tokens": integer,
"stream_options": {
  "include_usage": boolean
}
}

```

## 多模态对话补全格式

该格式用于图像与文本混合输入：

```

{
  "messages": [
    {
      "role": "user",
      "content": [
        {"type": "text", "text": "What's in this image?"},
        {"type": "image_url", "image_url": {"url": "data:image/jpeg;base64,..."}}
      ]
    }
  ],
  "max_tokens": integer,
  "temperature": float,
  "top_p": float,
  "stream": boolean
}

```

## 请求参数

- **messages** ( 数组 )：用于对话补全格式。由包含 **role** 和 **content** 字段的对象数组组成。**content** 为字符串表示纯文本输入，为数组表示多模态输入。
- **prompt** ( 字符串 )：用于文本补全格式。用于生成内容的输入文本。
- **max\_tokens** ( 整数 )：响应中生成的最大词元数。取值范围： $\geq 1$ 。
- **max\_completion\_tokens** ( 整数 )：**max\_tokens** 的替代参数，用于对话补全。生成的最大补全词元数。
- **temperature** ( 浮点数 )：控制生成内容的随机性。取值范围：0.0 到 2.0 ( 0.0 = 确定性生成，2.0 = 最大随机性 )。
- **top\_p** ( 浮点数 )：核采样阈值。取值范围： $1e-10$  到 1.0。

- `top_k` ( 整数 ) : 将词元选择范围限制为概率最高的前 K 个词元。取值范围 : 大于等于 -1 ( -1 = 无限制 )。
- `stream` ( 布尔值 ) : 是否流式返回响应。true 为流式 , false 为非流式。
- `logprobs` ( 布尔值/整数 ) : 对话补全使用布尔值。文本补全使用整数 , 表示返回的对数概率数量。取值范围 : 1 到 20。
- `top_logprobs` ( 整数 ) : 返回对数概率的概率最高词元数量 ( 仅对话补全 )。
- `reasoning_effort` ( 字符串 ) : 推理强度等级。选项 : low、high ( 仅限 Nova 2 Lite 自定义模型的聊天补全 )。
- `allowed_token_ids` ( 数组 ) : 允许生成的词元 ID 列表。用于将输出限制为指定词元。
- `truncate_prompt_tokens` ( 整数 ) : 若提示词超出限制 , 则截断为该词元数量。
- `stream_options` ( 对象 ) : 流式响应选项。包含布尔值 `include_usage` , 用于在流式响应中包含词元用量信息。

## 响应格式

响应格式取决于调用方式与请求类型 :

对话补全响应 ( 非流式 )

适用于同步对话补全请求 :

```
{
  "id": "chatcmpl-123e4567-e89b-12d3-a456-426614174000",
  "object": "chat.completion",
  "created": 1677652288,
  "model": "nova-micro-custom",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "Hello! I'm doing well, thank you for asking. How can I help you today?",
        "refusal": null,
        "reasoning": null,
        "reasoning_content": null
      },
      "logprobs": {
        "content": [
```

```
{
  "token": "Hello",
  "logprob": -0.31725305,
  "bytes": [72, 101, 108, 108, 111],
  "top_logprobs": [
    {
      "token": "Hello",
      "logprob": -0.31725305,
      "bytes": [72, 101, 108, 108, 111]
    },
    {
      "token": "Hi",
      "logprob": -1.3190403,
      "bytes": [72, 105]
    }
  ]
},
"finish_reason": "stop",
"stop_reason": null,
"token_ids": [9906, 0, 358, 2157, 1049, 11, 1309, 345, 369, 6464, 13]
},
"usage": {
  "prompt_tokens": 9,
  "completion_tokens": 12,
  "total_tokens": 21,
  "prompt_tokens_details": {
    "cached_tokens": 0
  }
},
"prompt_token_ids": [9906, 0, 358]
}
```

## 文本补全响应 (非流式)

适用于同步文本补全请求：

```
{
  "id": "cml-123e4567-e89b-12d3-a456-426614174000",
  "object": "text_completion",
  "created": 1677652288,
  "model": "nova-micro-custom",
```

```
"choices": [
  {
    "index": 0,
    "text": "Paris, the capital and most populous city of France.",
    "logprobs": {
      "tokens": ["Paris", ",", " the", " capital"],
      "token_logprobs": [-0.31725305, -0.07918124, -0.12345678, -0.23456789],
      "top_logprobs": [
        {
          "Paris": -0.31725305,
          "London": -1.3190403,
          "Rome": -2.1234567
        },
        {
          ",": -0.07918124,
          " is": -1.2345678
        }
      ]
    },
    "finish_reason": "stop",
    "stop_reason": null,
    "prompt_token_ids": [464, 6864, 315, 4881, 374],
    "token_ids": [3915, 11, 279, 6864, 323, 1455, 95551, 3363, 315, 4881, 13]
  }
],
"usage": {
  "prompt_tokens": 5,
  "completion_tokens": 11,
  "total_tokens": 16,
  "prompt_tokens_details": {
    "cached_tokens": 0
  }
}
}
```

## 对话补全流式响应

适用于流式对话补全请求，响应以服务器发送事件 ( SSE ) 形式返回：

```
data: {
  "id": "chatcmpl-123e4567-e89b-12d3-a456-426614174000",
  "object": "chat.completion.chunk",
  "created": 1677652288,
  "model": "nova-micro-custom",
```

```
"choices": [
  {
    "index": 0,
    "delta": {
      "role": "assistant",
      "content": "Hello",
      "refusal": null,
      "reasoning": null,
      "reasoning_content": null
    },
    "logprobs": {
      "content": [
        {
          "token": "Hello",
          "logprob": -0.31725305,
          "bytes": [72, 101, 108, 108, 111],
          "top_logprobs": [
            {
              "token": "Hello",
              "logprob": -0.31725305,
              "bytes": [72, 101, 108, 108, 111]
            }
          ]
        }
      ]
    },
    "finish_reason": null,
    "stop_reason": null
  }
],
"usage": null,
"prompt_token_ids": null
}

data: {
  "id": "chatcmpl-123e4567-e89b-12d3-a456-426614174000",
  "object": "chat.completion.chunk",
  "created": 1677652288,
  "model": "nova-micro-custom",
  "choices": [
    {
      "index": 0,
      "delta": {
        "content": "! I'm"
```

```
    },
    "logprobs": null,
    "finish_reason": null,
    "stop_reason": null
  }
],
"usage": null
}

data: {
  "id": "chatcmpl-123e4567-e89b-12d3-a456-426614174000",
  "object": "chat.completion.chunk",
  "created": 1677652288,
  "model": "nova-micro-custom",
  "choices": [
    {
      "index": 0,
      "delta": {},
      "finish_reason": "stop",
      "stop_reason": null
    }
  ],
  "usage": {
    "prompt_tokens": 9,
    "completion_tokens": 12,
    "total_tokens": 21,
    "prompt_tokens_details": {
      "cached_tokens": 0
    }
  }
}

data: [DONE]
```

## 文本补全流式响应

适用于流式文本补全请求：

```
data: {
  "id": "cmpl-123e4567-e89b-12d3-a456-426614174000",
  "object": "text_completion",
  "created": 1677652288,
  "model": "nova-micro-custom",
  "choices": [
```

```
{
  "index": 0,
  "text": "Paris",
  "logprobs": {
    "tokens": ["Paris"],
    "token_logprobs": [-0.31725305],
    "top_logprobs": [
      {
        "Paris": -0.31725305,
        "London": -1.3190403
      }
    ]
  },
  "finish_reason": null,
  "stop_reason": null
}

data: {
  "id": "cml-123e4567-e89b-12d3-a456-426614174000",
  "object": "text_completion",
  "created": 1677652288,
  "model": "nova-micro-custom",
  "choices": [
    {
      "index": 0,
      "text": ", the capital",
      "logprobs": null,
      "finish_reason": null,
      "stop_reason": null
    }
  ],
  "usage": null
}

data: {
  "id": "cml-123e4567-e89b-12d3-a456-426614174000",
  "object": "text_completion",
  "created": 1677652288,
  "model": "nova-micro-custom",
  "choices": [
    {
```

```
    "index": 0,
    "text": "",
    "finish_reason": "stop",
    "stop_reason": null
  }
],
"usage": {
  "prompt_tokens": 5,
  "completion_tokens": 11,
  "total_tokens": 16
}
}
data: [DONE]
```

## 响应字段说明

- `id` : 补全结果的唯一标识符
- `object` : 返回对象类型 ( “chat.completion” “text\_completion” “chat.completion.chunk” )
- `created` : 生成补全结果的 Unix 时间戳
- `model` : 用于生成补全结果的模型
- `choices` : 补全结果数组
- `usage` : 词元用量信息，包含提示词词元、补全词元和总词元数
- `logprobs` : 词元的对数概率信息 ( 需主动请求 )
- `finish_reason` 模型停止生成的原因 ( “stop” “length” “content\_filter” )
- `delta` : 流式响应中的增量内容
- `reasoning` : 使用 `reasoning_effort` 时的推理内容
- `token_ids` : 生成文本对应的词元 ID 数组

如需完整的 API 文档，请参阅 [InvokeEndpoint API 参考](#) 和 [InvokeEndpointWithResponseStream API 参考](#)。

## 评估在 SageMaker 推理上托管的模型

本指南介绍如何使用开源评估框架 [Inspect AI](#)，对部署在 SageMaker 推理端点上的自定义 Amazon Nova 模型进行评估。

**Note**

有关实操演示，请参阅 [SageMaker Inspect AI 快速入门笔记本](#)。

## 概述

您可以使用人工智能研究社区的标准化基准，对部署在 SageMaker 端点上的自定义 Amazon Nova 模型进行评估。该方式支持：

- 大规模评估自定义 Amazon Nova 模型（微调、蒸馏或其他适配版本）
- 通过多端点实例并行推理执行评估
- 使用 MMLU、TruthfulQA、HumanEval 等基准对比模型性能
- 与现有的 SageMaker 基础设施集成

## 支持的模型

SageMaker 推理提供程序支持以下模型/端点类型：

- Amazon Nova 系列模型（Nova Micro、Nova Lite、Nova Lite 2）
- 通过 vLLM 或兼容 OpenAI 协议的推理服务器部署的模型
- 任何支持 OpenAI Chat Completions API 格式的端点

## 先决条件

在开始之前，请确保您满足以下条件：

- 拥有 AWS 账户，且该账户具备创建和调用 SageMaker 端点的权限
- 已通过 AWS CLI CLI、环境变量或 IAM 角色配置好 AWS 凭证
- Python 3.9 或更高版本

## 所需的 IAM 权限

您的 IAM 用户或角色需具备以下权限：

```
{  
  "Version": "2012-10-17",
```

```
"Statement": [  
  {  
    "Effect": "Allow",  
    "Action": [  
      "sagemaker:InvokeEndpoint",  
      "sagemaker:DescribeEndpoint"  
    ],  
    "Resource": "arn:aws:sagemaker:*:*:endpoint/*"  
  }  
]
```

## 步骤 1：部署 SageMaker 端点

运行评估前，需先部署承载目标模型的 SageMaker 推理端点。

有关使用 Amazon Nova 模型创建 SageMaker 推理端点的操作指引，请参阅[the section called “开始使用”](#)。

当端点状态变为 InService 后，请记录端点名称，以便在评估命令中使用。

## 步骤 2：安装评估依赖项

创建 Python 虚拟环境，并安装所需的依赖包。

```
# Create virtual environment  
python3.12 -m venv venv  
source venv/bin/activate  
  
# Install uv for faster package installation  
pip install uv  
  
# Install Inspect AI and evaluation benchmarks  
uv pip install inspect-ai inspect-evals  
  
# Install AWS dependencies  
uv pip install aioboto3 boto3 botocore openai
```

## 步骤 3：配置 AWS 凭证

选择以下任一身份验证方式：

选项 1：AWS CLI ( 建议 )

```
aws configure
```

提示后，输入 AWS 访问密钥 ID、秘密访问密钥和默认区域名称。

### 选项 2：环境变量

```
export AWS_ACCESS_KEY_ID=your_access_key
export AWS_SECRET_ACCESS_KEY=your_secret_key
export AWS_DEFAULT_REGION=us-west-2
```

### 选项 3：IAM 角色

若在 Amazon EC2 或 SageMaker 笔记本中运行，将自动使用实例的 IAM 角色。

### 验证凭证

```
import boto3

sts = boto3.client('sts')
identity = sts.get_caller_identity()
print(f"Account: {identity['Account']}")
print(f>User/Role: {identity['Arn']}")
```

### 步骤 4：安装 SageMaker 提供程序

SageMaker 提供程序用于让 Inspect AI 与您的 SageMaker 端点通信。[快速入门笔记本](#)中已简化该提供程序的安装流程。

### 步骤 5：下载评估基准

克隆 Inspect Evals 存储库以获取标准评估基准：

```
git clone https://github.com/UKGovernmentBEIS/inspect_evals.git
```

该存储库包含的评估基准包括：

- MMLU 和 mmlu-Pro ( 知识与推理 )
- TruthfulQA ( 真实性 )
- HumanEval ( 代码生成 )

- GSM8K ( 数学推理 )

## 步骤 6 : 运行评估

使用您的 SageMaker 端点运行评估 :

```
cd inspect_evals/src/inspect_evals/

inspect eval mmlu_pro/mmlu_pro.py \
  --model sagemaker/my-nova-endpoint \
  -M region_name=us-west-2 \
  --max-connections 256 \
  --max-retries 100 \
  --display plain
```

### 关键参数

| 参数                 | 默认值       | 说明                                                |
|--------------------|-----------|---------------------------------------------------|
| --max-connections  | 10        | 发送到端点的并行请求数。需随实例数量调整 ( 例如 : 10 个实例 x 25 = 250 ) 。 |
| --max-retries      | 3         | 失败请求的重试次数。大规模评估建议设为 50-100。                       |
| -M region_name     | us-east-1 | 端点部署所在的 AWS 区域。                                   |
| -M read_timeout    | 600       | 请求超时时间 ( 单位 : 秒 ) 。                               |
| -M connect_timeout | 60        | 连接超时时间 ( 单位 : 秒 ) 。                               |

### 调优建议

针对多实例端点 :

```
# 10-instance endpoint example
--max-connections 250 # ~25 connections per instance
--max-retries 100    # Handle transient errors
```

--max-connections 设置过高可能导致端点过载并触发节流 ; 设置过低则会造成资源利用率不足。

## 步骤 7：查看结果

启动 Inspect AI 可视化工具分析评估结果：

```
inspect view
```

该工具将展示以下内容：

- 整体得分与指标
- 含模型响应的单样本评估结果
- 错误分析与失败模式

## 管理终端节点

### 更新端点

如需使用新模型或新配置更新现有端点：

```
import boto3

sagemaker = boto3.client('sagemaker', region_name=REGION)

# Create new model and endpoint configuration
# Then update the endpoint
sagemaker.update_endpoint(
    EndpointName=EXISTING_ENDPOINT_NAME,
    EndpointConfigName=NEW_ENDPOINT_CONFIG_NAME
)
```

### 删除端点

```
sagemaker.delete_endpoint(EndpointName=ENDPOINT_NAME)
```

## 添加自定义评估基准

您可以通过以下工作流向 Inspect AI 添加新的评估基准：

1. 研究基准数据集格式与评估指标
2. 参考 `inspect_evals/` 中类似的实现

3. 创建任务文件，将数据集记录转换为 Inspect AI 样本
4. 实现对应的求解器与评分器
5. 使用小规模测试运行进行验证

示例任务结构：

```
from inspect_ai import Task, task
from inspect_ai.dataset import hf_dataset
from inspect_ai.scorer import choice
from inspect_ai.solver import multiple_choice

@task
def my_benchmark():
    return Task(
        dataset=hf_dataset("dataset_name", split="test"),
        solver=multiple_choice(),
        scorer=choice()
    )
```

## 问题排查

### 常见问题

#### 端点节流或超时

- 降低 `--max-connections`
- 提高 `--max-retries`
- 查看端点的 CloudWatch 指标，确认是否存在容量问题

#### 身份验证错误

- 确认 AWS 凭证配置正确
- 检查 IAM 权限是否包含 `sagemaker:InvokeEndpoint`

#### 模型错误

- 确认端点处于 InService 状态
- 检查模型是否支持 OpenAI Chat Completions API 格式

## 相关资源

- [Inspect AI 文档](#)
- [Inspect Evals 存储库](#)
- [SageMaker 开发人员指南](#)
- [署模型用于推理](#)
- [配置 AWS CLI](#)

## 在 Amazon SageMaker Inference 滥用检测中部署 Amazon Nova Forge 模型

AWS 致力于以负责任的方式使用人工智能。为防止潜在滥用，在 Amazon SageMaker Inference 中部署 Amazon Nova Forge 模型时，SageMaker Inference 将启用自动化滥用检测机制，用于识别可能违反 AWS [可接受使用政策](#)（AUP）和服务条款（包括[负责任的人工智能政策](#)）相关规定的行为。

我们的滥用检测机制是完全自动化的，因此无需人工审核或者访问用户输入或模型输出。

自动化滥用检测包括：

- **对内容进行分类：**我们使用分类器来检测用户输入和模型输出中的有害内容（例如煽动暴力的内容）。分类器是一种处理模型输入和输出，并指定危害类型和置信度级别的算法。我们可能会在 Amazon Nova Forge 模型使用过程中运行这些分类器。分类过程是自动化的，不涉及对用户输入或模型输出进行人工审核。
- **识别模式：**我们使用分类器指标来识别潜在的违规行为和反复出现的行为。我们可能会编译和分享匿名的分类器指标。Amazon SageMaker Inference 不会存储用户输入或模型输出。
- **检测并屏蔽儿童性虐待素材（CSAM）：**您需对自己（及终端用户）上传到 Amazon SageMaker Inference 的内容承担责任，并确保相关内容不含非法图像。为阻止 CSAM 的传播，当在 Amazon SageMaker Inference 中部署 Amazon Nova Forge 模型时，SageMaker Inference 可能会使用自动化滥用检测机制（例如哈希匹配技术或分类器）来检测明显的 CSAM。如果 Amazon SageMaker Inference 在您的图片输入中检测到明显的 CSAM，Amazon SageMaker Inference 将拦截该请求，并且您将收到一条自动发出的错误消息。Amazon SageMaker Inference 还可能向美国国家失踪与受虐儿童中心（NCMEC）或相关机构提交报告。我们高度重视 CSAM 问题，并将持续改进我们的检测、拦截和报告机制。您可能需要按照相关法律采取其他措施，并对这些行为负责。

自动化滥用检测机制识别出潜在的违规行为之后，我们可能会要求您说明自己如何使用 Amazon SageMaker Inference，以及如何遵守我们的服务条款。如果您未作出回应、不愿意或无法遵守这些条

款或策略，AWS 可能会暂停您对 Amazon SageMaker Inference 的访问。如果我们的自动化测试检测到模型响应未遵守我们的条款和政策，您可能仍需要为失败的推理作业付费。

如有其他疑问，请联系 AWS Support。有关更多信息，请参阅 [Amazon SageMaker 常见问题](#)。

## Amazon Bedrock 推理

在完成 Amazon Nova 模型的训练与测试后，即可将其部署到 Amazon Bedrock 以实现生产级规模的推理。部署流程包括：通过 CreateCustomModel API 创建 Amazon Bedrock 模型，将模型构件从托管的 Amazon S3 存储桶导出到该模型；待模型状态变为 ACTIVE 后，配置支持按需推理或预置吞吐量推理的端点。

在 SageMaker 中创建自定义模型后，即可使用 CreateCustomModel API 将其从 SageMaker 托管暂存区部署到 Amazon Bedrock 以运行推理。然后，您可以使用 CreateCustomModelDeployment 为参数高效微调 (PEFT) 模型创建 OD 推理端点或设置预置吞吐量推理。您还可以为全秩自定义模型设置预置吞吐量推理。

您也可以使用 Amazon Nova Forge SDK 部署自定义的 Amazon Nova 模型。Amazon Nova Forge SDK 提供简化的操作流程，支持从训练作业或 S3 模型检查点中提取相关信息，并将其发布到 Amazon Bedrock。有关更多信息，请参阅 [Amazon Nova Forge SDK](#)。

有关为自定义模型设置 Amazon Bedrock 推理的详细步骤，请参阅 [将自定义 Amazon Nova 模型部署到 Amazon Bedrock](#)。

以下章节将详细介绍自定义模型上的按需推理。

### 部署自定义模型进行按需型推理

通过模型自定义作业（微调、蒸馏或持续预训练）成功创建自定义模型后，可以为模型设置按需型推理。

要为自定义模型设置按需型推理，请使用自定义模型部署来部署模型。部署自定义模型后，请使用部署的 Amazon 资源名称 (ARN) 作为 InvokeModel 或 Converse API 操作中的 modelId 参数。您可以通过 Amazon Bedrock 功能（例如演示平台、代理和知识库），将已部署的模型用于按需型推理。

#### 主题

- [支持的模型](#)
- [部署自定义模型](#)
- [使用部署进行按需型推理](#)
- [删除自定义模型部署](#)

## 支持的模型

您可以为以下模型设置按需型推理：

- Amazon Nova Canvas
- Amazon Nova Lite
- Amazon Nova Micro
- Amazon Nova Pro

## 部署自定义模型

您可以使用 Amazon Bedrock 控制台、AWS Command Line Interface 或 AWS SDK 部署自定义模型。有关使用部署进行推理的信息，请参阅[使用部署进行按需型推理](#)。

### 主题

- [部署自定义模型 \(控制台\)](#)
- [部署自定义模型 \(AWS Command Line Interface\)](#)
- [部署自定义模型 \(AWS SDK\)](#)

### 部署自定义模型 (控制台)

您可以从自定义模型页面中部署自定义模型，如下所示。您也可以从按需自定义模型页面中部署具有相同字段的模型。要找到此页面，请在导航窗格的推理和评估中选择按需自定义模型。

### 部署自定义模型

1. 使用[具有 Amazon Bedrock 权限的 IAM 角色](#)登录 AWS 管理控制台，然后通过 <https://console.aws.amazon.com/nova/> 打开 Amazon Bedrock 控制台。
2. 在左侧导航窗格的基础模型下，选择自定义模型。
3. 在模型选项卡中，选择想要部署的模型对应的单选按钮。
4. 选择设置推理，然后选择按需部署。
5. 在部署详细信息页面上，提供以下信息：
  - 部署名称 (必填)：输入部署的唯一名称。
  - 描述 (可选)：为部署描述输入描述。

- 标签 ( 可选 ) : 为成本分配和资源管理添加标签。
6. 选择创建。当状态显示 Completed 时，表示自定义模型已做好按需推理的准备。有关使用自定义模型的更多信息，请参阅[使用部署进行按需型推理](#)。

### 部署自定义模型 (AWS Command Line Interface)

要使用 AWS Command Line Interface 部署自定义模型进行按需型推理，请使用带有自定义模型的 Amazon 资源名称 (ARN) 的 create-custom-model-deployment 命令。此命令使用 [CreateCustomModelDeployment](#) API 操作。它返回部署的 ARN，您可以在提出推理请求时将其用作 modelId。有关使用部署进行推理的信息，请参阅[使用部署进行按需型推理](#)。

```
aws bedrock create-custom-model-deployment \
--model-deployment-name "Unique name" \
--model-arn "Custom Model ARN" \
--description "Deployment description" \
--tags '[
  {
    "key": "Environment",
    "value": "Production"
  },
  {
    "key": "Team",
    "value": "ML-Engineering"
  },
  {
    "key": "Project",
    "value": "CustomerSupport"
  }
]' \
--client-request-token "unique-deployment-token" \
--region region
```

### 部署自定义模型 (AWS SDK)

要部署自定义模型进行按需型推理，请使用带有自定义模型的 Amazon 资源名称 (ARN) 的 [CreateCustomModelDeployment](#) API 操作。响应返回部署的 ARN，您可以在提出推理请求时将其用作 modelId。有关使用部署进行推理的信息，请参阅[使用部署进行按需型推理](#)。

以下代码演示了如何使用 SDK for Python (Boto3) 部署自定义模型。

```
def create_custom_model_deployment(bedrock_client):
```

```
"""Create a custom model deployment
Args:
    bedrock_client: A boto3 Bedrock client for making API calls

Returns:
    str: The ARN of the created custom model deployment

Raises:
    Exception: If there is an error creating the deployment
"""

try:
    response = bedrock_client.create_custom_model_deployment(
        modelDeploymentName="Unique deployment name",
        modelArn="Custom Model ARN",
        description="Deployment description",
        tags=[
            {'key': 'Environment', 'value': 'Production'},
            {'key': 'Team', 'value': 'ML-Engineering'},
            {'key': 'Project', 'value': 'CustomerSupport'}
        ],
        clientRequestToken=f"deployment-{uuid.uuid4()}"
    )

    deployment_arn = response['customModelDeploymentArn']
    print(f"Deployment created: {deployment_arn}")
    return deployment_arn

except Exception as e:
    print(f"Error creating deployment: {str(e)}")
    raise
```

## 使用部署进行按需型推理

部署自定义模型进行按需型推理后，可通过提出推理请求，使用模型来生成响应。对于 `InvokeModel` 或 `Converse` 操作，请使用部署的 Amazon 资源名称 (ARN) 作为 `modelId`。

有关提出推理请求的信息，请参阅以下主题：

- [使用模型推理提交提示并生成响应](#)
- [运行模型推理的先决条件](#)
- [使用 API 提交提示并生成响应](#)

## 删除自定义模型部署

在模型用于按需型推理之后，可以删除部署。删除部署后，就无法将其用于按需型推理，但底层的自定义模型不会删除。

您可以使用 Amazon Bedrock 控制台、AWS Command Line Interface 或 AWS SDK 删除自定义模型部署。

### Important

自定义模型部署删除后不可逆。请确保不再需要部署后再进行删除。如果需要再次使用自定义模型进行按需型推理，则必须创建新的部署。

### 主题

- [删除自定义模型部署 \(控制台\)](#)
- [删除自定义模型部署 \(AWS Command Line Interface\)](#)
- [删除自定义模型部署 \(AWS SDK\)](#)

### 删除自定义模型部署 (控制台)

#### 删除自定义模型部署

1. 在导航窗格中，在推理和评估下选择按需自定义模型。
2. 选择想要删除的自定义模型部署。
3. 选择删除。
4. 在确认对话框中，键入部署名称来确认删除。
5. 选择删除以确认。

在删除过程中，部署状态将更改为 Deleting。一旦完成后，部署将从列表中删除。

### 删除自定义模型部署 (AWS Command Line Interface)

要使用 AWS Command Line Interface 删除自定义模型部署，请使用带有部署标识符的 `delete-custom-model-deployment` 命令。

```
aws bedrock delete-custom-model-deployment \  
--custom-model-deployment-identifier "deployment-arn-or-name" \  

```

```
--region region
```

## 删除自定义模型部署 (AWS SDK)

要通过编程删除自定义模型部署，请使用带有部署的 Amazon 资源名称 (ARN) 或名称的 [DeleteCustomModelDeployment](#) API 操作。以下代码演示了如何使用 SDK for Python (Boto3) 删除自定义模型部署。

```
def delete_custom_model_deployment(bedrock_client):
    """Delete a custom model deployment

    Args:
        bedrock_client: A boto3 Bedrock client for making API calls

    Returns:
        dict: The response from the delete operation

    Raises:
        Exception: If there is an error deleting the deployment
    """

    try:
        response = bedrock_client.delete_custom_model_deployment(
            customModelDeploymentIdentifier="Deployment identifier"
        )

        print(f"Deployment deletion initiated")
        return response

    except Exception as e:
        print(f"Error deleting deployment: {str(e)}")
        raise
```

## 自定义模型中的按需型推理

通过按需型 (OD) 推理，您可以在自定义 Amazon Nova 模型上运行推理，而无需维护预置吞吐量端点。这有助于优化成本，并有效地进行扩展。使用按需型推理时，您将根据使用量付费，使用量以输入和输出的 Token 数计量。

### 兼容性要求

应满足以下兼容性要求：

- Amazon Nova Pro、Lite 及 Micro 的自定义理解模型支持 OD 推理。Nova 自定义内容生成模型不支持 OD 推理。
- 2025 年 7 月 16 日后训练的 Amazon Nova 自定义理解模型支持 OD 推理。2025 年 7 月 16 日前训练的自定义模型与 OD 推理不兼容。
- Amazon Bedrock 自定义：使用 Amazon Bedrock 自定义进行自定义的模型，以及使用 Amazon Bedrock 从教师式模型中蒸馏的学生式模型都支持 OD 推理。
- SageMaker AI 自定义：对于在 SageMaker AI 中进行自定义的模型，只有模型托管在 Amazon Bedrock 上时才支持 OD 推理功能，且仅限于参数高效微调 (PEFT) 模型。这包括直接偏好优化 + PEFT。全秩微调模型不支持 OD 推理。

## 模型训练和推理

2025 年 7 月 16 日后，当使用 PEFT 在 Amazon Bedrock 或 SageMaker AI 上训练新的自定义 Amazon Nova Pro、Lite 或 Micro 模型时，该模型将自动与预置的按需型推理选项兼容。您可以在部署模型时选择首选的推理方法。

要在 2025 年 7 月 16 日后训练的模型中使用 OD 推理，需完成以下步骤：

1. 使用 [Amazon Bedrock 自定义 API](#) 或 [SageMaker AI 定制 API](#) 创建新的微调任务。
2. 使用 [CreateCustomModel API](#) 将新训练的模型部署至 Amazon Bedrock。
3. 使用 CustomModelDeployment API 进行部署，从而进行按需型推理。

## 速率限制

以下的每分钟请求数 (RPM) 以及每分钟 Token 数 (TPM) 限制适用于按需型推理请求：

| Base Model for Custom Model | RPM per Custom Model Deployment | TPM per Custom Model Deployment |
|-----------------------------|---------------------------------|---------------------------------|
| Nova 2 Lite                 | 2,000                           | 4,000,000                       |

要详细了解 Amazon Nova 的可用配额，请参阅[Amazon Nova 的配额](#)。

## 延迟

调用基础模型与适配器时，端到端延迟（即首词元响应时间 TTFT）预计会有 20% 到 55% 的差异。确切的延迟值因模型大小而异，并与行业标准一致。

## 自定义 Amazon Nova 模型的限制

Amazon Nova 自定义功能不支持 SageMaker 上的以下功能。

- 通过 SSH 进入实例以查找指标

由于安全控制措施已就绪，因此，您无法通过 SSH 进入训练 algo-1 实例中的主节点来查找内存统计数据或 NVIDIA 统计数据并验证训练步骤。

- 支持 SageMaker 训练器 SDK

SageMaker 训练器目前无法用于启动训练作业。如今，您只能通过估算器 API 启动训练作业。

- SageMaker 训练作业无法使用暖池

由于安全控制措施已就绪，因此，SageMaker 暖池无法用于将实例保留在暖池中直到失效为止。

- 自定义模型合并

目前不支持合并多个模型。也就是说，无法创建多个 LoRA 适配器并使用基本模型执行多重合并操作。

- 支持的可观测性工具

[TensorBoard](#) 是唯一支持的用于查看 SageMaker 训练作业指标的可观测性工具。目前尚不支持 [MLFlow](#) 或 [WandB](#)。有关在 SageMaker 中使用 TensorBoard 的更多信息，请参阅 [SageMaker 中的 TensorBoard](#)。

# Amazon Nova Forge

Amazon Nova Forge 是一项首创服务，让组织能够以最简单、最具成本效益的方式使用 Nova 构建自己的前沿模型。

Amazon Nova Forge 引入了“开放训练”模型的概念，使组织能够访问各种早期模型检查点，并在模型训练的每个阶段将专有数据与 Amazon 精选的数据集相结合。这使得模型能够最大限度地从专有数据中学习，同时降低遗忘推理等基础技能的风险。

Nova Forge 提供以下关键功能：

- 访问模型开发各个阶段的检查点，并在新的 Nova 模型广泛使用之前加以利用
- 将专有数据与 Amazon Nova 精选的训练数据相结合
- 在环境中使用奖励函数执行强化学习
- 使用一键式配方，通过可视化工作流程或命令行界面优化 Nova 的构建过程
- 使用负责任的人工智能工具包，在训练过程中使模型符合 Amazon Nova 负责任的人工智能指南，并在推理过程中实施运行时控制以调节模型响应。

## 先决条件

主题

- [订阅 Nova Forge](#)
- [其他先决条件](#)

## 订阅 Nova Forge

要请求访问 Amazon Nova Forge 服务，请将以下标签添加到控制台 IAM 角色：键为 `forge-subscription`，值为 `true`。将此标签添加到角色后，请前往 SageMaker AI 控制台 >“模型训练和自定义”，然后单击“Nova Forge”。在此页面上，您将找到有关服务、定价信息和功能的详细信息。您可以请求订阅，然后从此页面管理订阅。

1. 该角色应具有调用 `api ListAttachedRolePolicy` 的权限，并且响应应包含 `AdministratorAccess` 或 `AmazonSageMakerFullAccess` 策略。
2. 该角色应具有调用 `api ListRoleTags` 的权限，并且响应标签应包含 `tag.key=forge-subscription`。

## 其他先决条件

此外，还应确保满足以下先决条件：

1. [一般先决条件](#)
2. 用户已完成其他步骤：将受限实例组（RIG）添加到 SageMaker HyperPod 集群（要完成此操作，请按照[此处](#)的步骤进行操作）

## Nova Forge 访问与设置

要开始使用 Nova Forge，请遵循以下两个步骤：

- 步骤 1：订阅 Nova Forge
- 步骤 2：设置 HyperPod 基础设施

## 获取 Nova Forge 文档

如需获取 Nova Forge 文档，请按照以下步骤操作：

```
mkdir NovaForgeHyperpodCLI
cd NovaForgeHyperpodCLI
aws s3 cp s3://nova-forge-c7363-206080352451-us-east-1/v1/ ./ --recursive
pip install -e .
```

## 步骤 1：订阅 Nova Forge

快速摘要：

1. 确认您拥有 Amazon Web Services 账户的管理员访问权限。
2. 导航到 SageMaker AI 控制台并请求访问 Nova Forge。
3. 订阅请求获批后，等待 Nova 团队通过电子邮件发送确认信息。
4. 使用 `forge-subscription` 标签标记执行角色。此标签是访问 Nova Forge 功能和检查点所必需的。将以下标签添加到执行角色：
  - 键：`forge-subscription`
  - 值：`true`

## 详细指南

要订阅 Nova Forge 并有效使用自定义服务，Amazon Web Services 客户必须拥有其 Amazon Web Services 账户的管理员访问权限，或者由其管理员授予其管理员访问权限。本文档概述了执行以下操作所需的步骤：

- 获取管理员访问权限
- 设置用于订阅 Nova Forge 的策略
- 访问自定义配方
- 配置自定义
- 监控 workflow
- 评估自定义模型检查点

## 选项 A

流程 1：账户用户必须联系账户管理员以请求以下操作：

- 通过 IAM 将 `forge-subscription` 标签添加到账户（有关步骤，请参阅附录 A）。
- 通过 IAM 添加 `ListRoleTags` 和 `ListAttachedRolePolicies` 权限（有关步骤，请参阅附录 B）。

**Required setup for requesting subscription**  
To request Nova Forge subscription, your IAM role must have the "forge-subscription" tag attached and the "ListRoleTags" and "ListAttachedRolePolicies" policies added.

**Nova Forge**  
Choose from multiple Nova model checkpoints to introduce your data at the optimal stage, mix Amazon-curated training data from each stage with your data, integrate your proprietary tools as reward functions, customize the responsible AI controls, and build your own custom frontier model.

**Subscription**  
Request subscription to access Nova Forge resources. Once approved, your payer account will be eligible for a 1-month free trial. **Infrastructure costs will apply during the free trial.** See Nova Forge pricing details below.

**Status**  
 Not subscribed

## 选项 B

流程：账户用户必须联系账户管理员以请求获得账户的管理员访问权限。

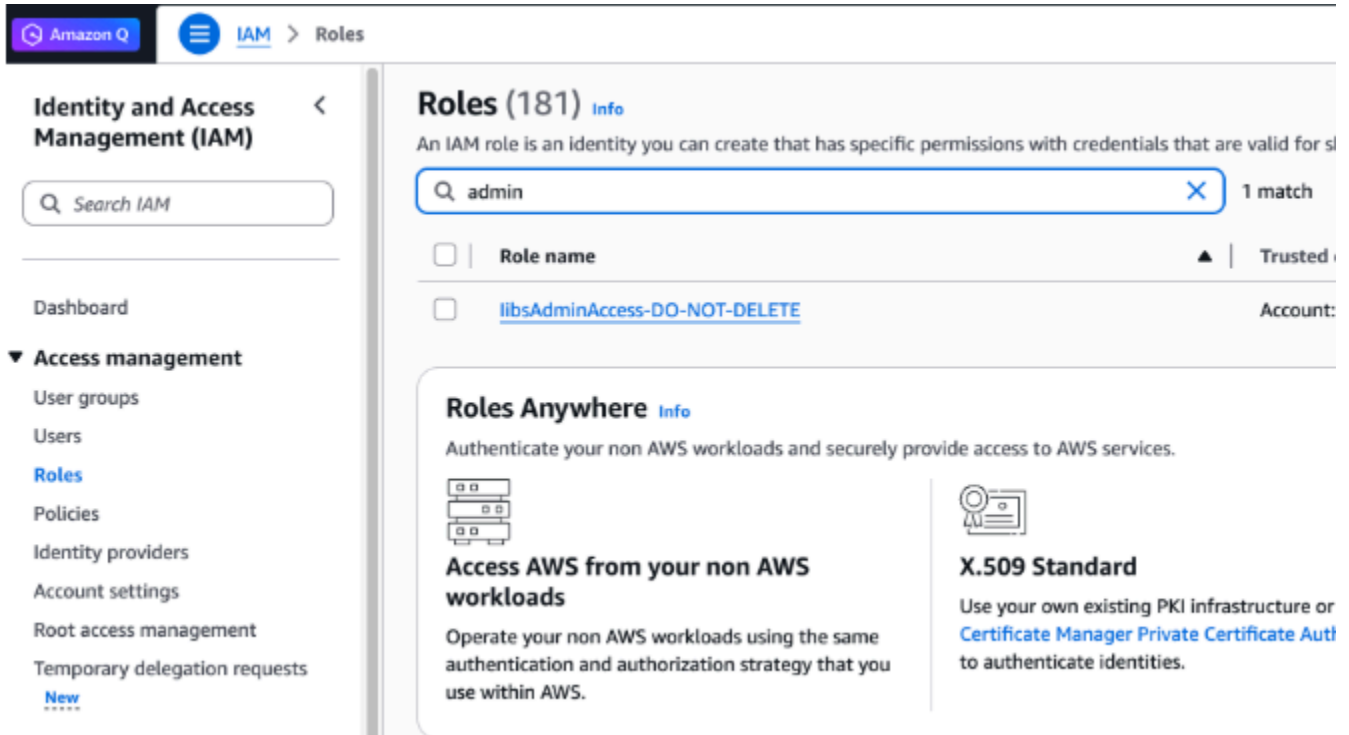
- 获得管理员访问权限后，请按照流程 2 中的步骤进行操作。

## 流程 2：具有管理员访问权限的 Amazon Web Services 账户

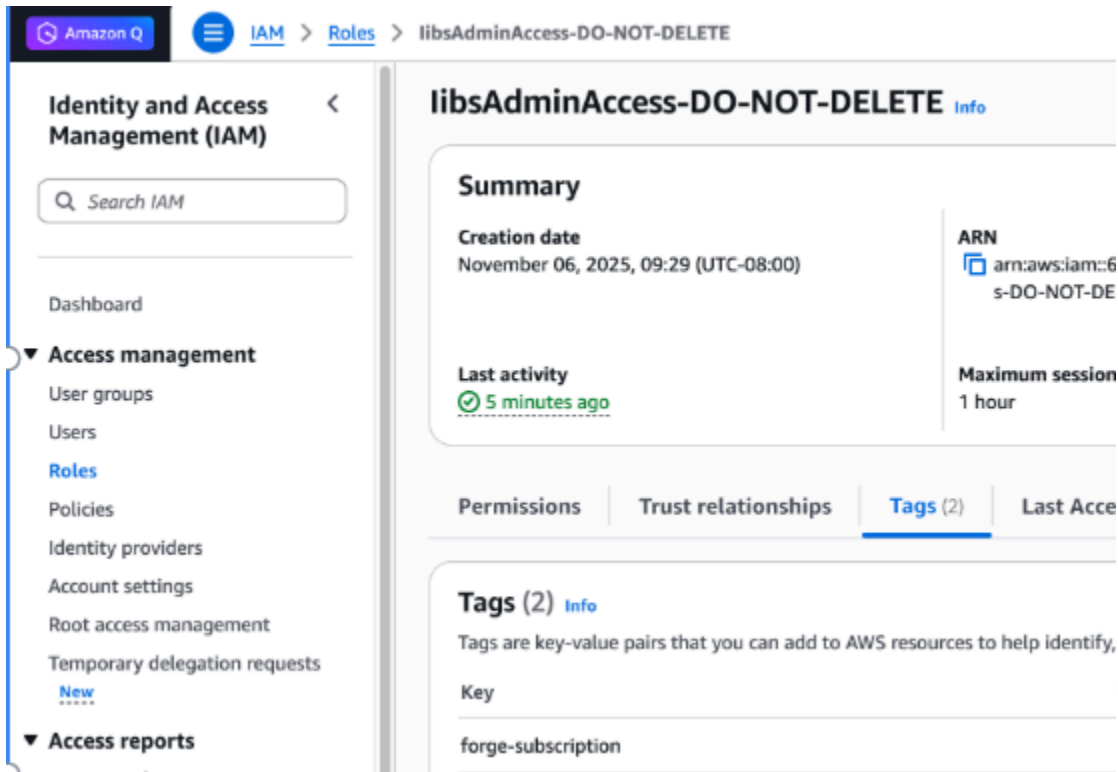
- 通过 IAM 将 forge-subscription 标签添加到账户。有关步骤，请参阅附录 A

### 附录 A. 将 forge-subscription 策略添加到 Amazon Web Services 账户

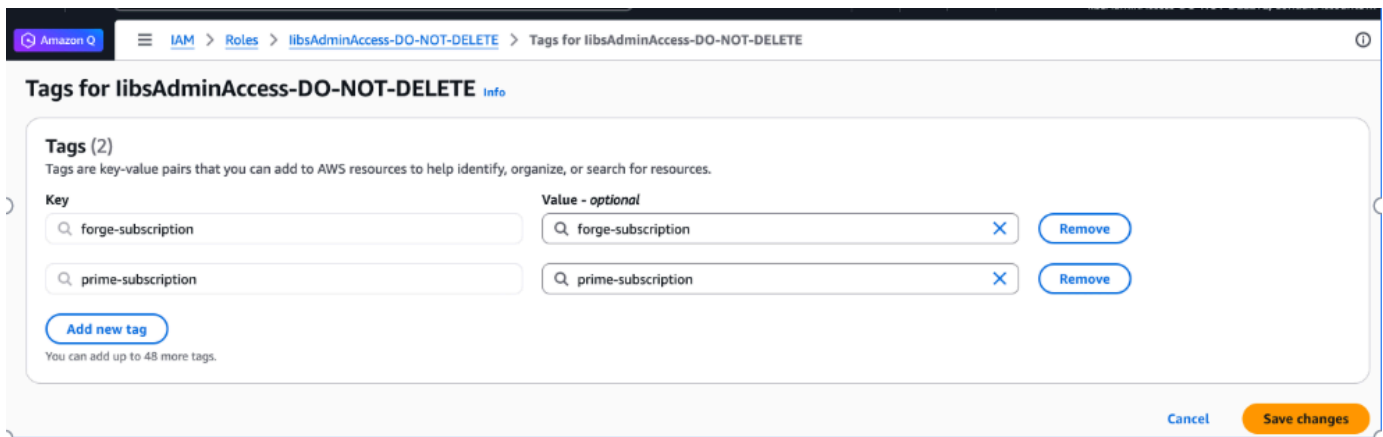
1. 转到 Amazon Web Services IAM 控制面板。单击左侧的“角色”。搜索管理员并单击管理员角色



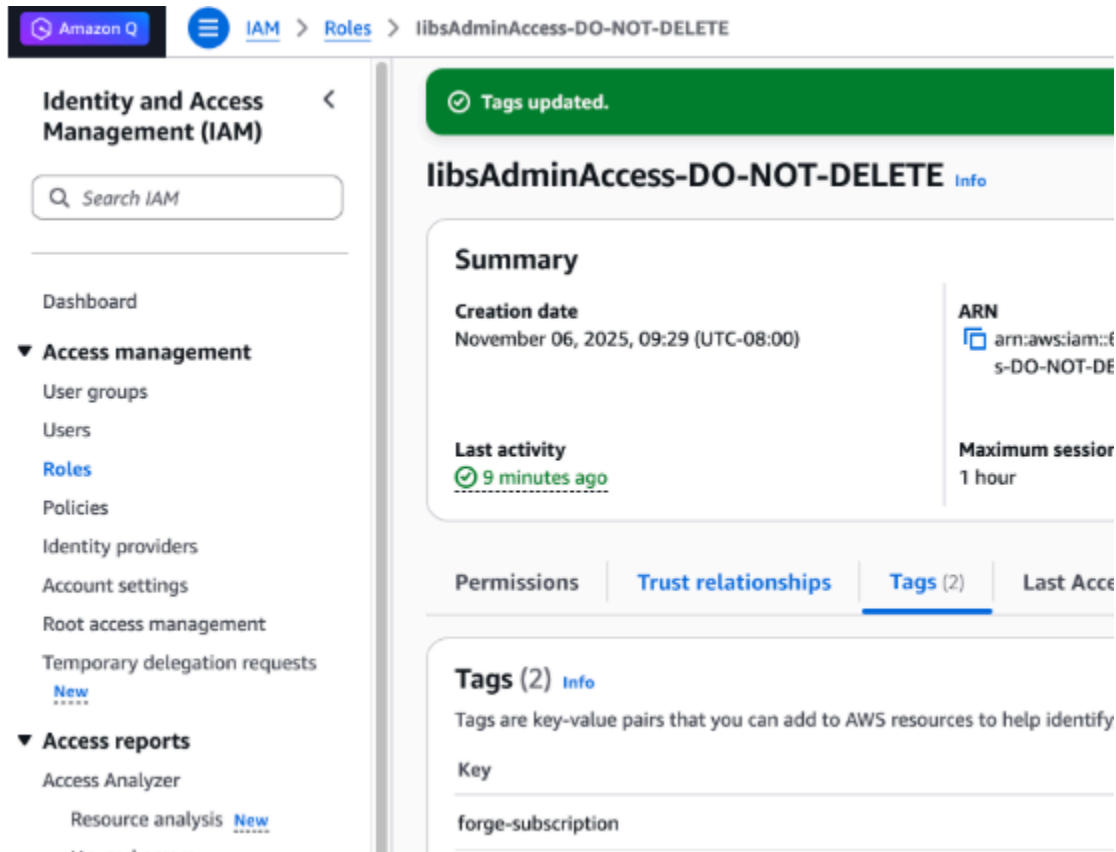
2. 选择 <AssumedRoleToUse> ( 例如 libsAdminAccess )。单击“标签”选项卡。



3. 单击“管理”选项卡。添加新标签。在“键”下输入“forge-subscription”，然后单击“保存更改”



4. 确保“标签”部分显示 forge-subscription 作为键。



## 附录 B. 由管理员将 ListRoleTags 和 ListAttachedPolicies 策略添加到非管理员角色的 Amazon Web Services 账户

1. 转到 Amazon Web Services IAM 控制面板。单击左侧的“角色”。搜索 <AssumedRoleToUse> (例如 ForgeAccessRole) 并单击该 <AssumedRoleToUse> (例如 ForgeAccessRole) 角色。

**Roles (30)** [Info](#)

An IAM role is an identity you can create that has specific permissions with credentials that are valid for s

Q forge X 1 match

| <input type="checkbox"/> | Role name                       | ▲ | Trusted  |
|--------------------------|---------------------------------|---|----------|
| <input type="checkbox"/> | <a href="#">ForgeAccessRole</a> |   | Account: |

**Roles Anywhere** [Info](#)

Authenticate your non AWS workloads and securely provide access to AWS services.

- 单击 <AssumedRoleToUse> ( 例如 ForgeAccessRole ) 角色并选择“标签”。添加类型为“forge-subscription”的新标签

**ForgeAccessRole** [Info](#)

**Summary**

Creation date  
December 09, 2025, 21:31 (UTC-05:00)

Last activity  
✔ 11 minutes ago

Permissions | Trust relationships | **Tags (1)** | Last Accessed

**Tags (1)** [Info](#)

Tags are key-value pairs that you can add to AWS resources to help identify, organ

| Key                |
|--------------------|
| forge-subscription |

- 在“权限”下，添加新权限：“添加权限”→“创建内联策略”→“添加以下所列策略”

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```

    "Sid": "VisualEditor0",
    "Effect": "Allow",
    "Action": [
        "iam:ListRoleTags",
        "iam:ListAttachedRolePolicies"
    ],
    "Resource": "*"
  }
]
}

```

**ForgeAccessRole** Info

**Summary**

Creation date  
December 09, 2025, 21:31 (UTC-05:00)

ARN  
arn:aws:iam:::role/forge-access-role

Last activity  
15 minutes ago

Maximum session duration  
1 hour

**Permissions** | Trust relationships | Tags (1) | Last Accessed | Revoke session

**Permissions policies (2)** Info

You can attach up to 10 managed policies.

Search

- Policy name
- AmazonSageMakerFullAccess
- forge-access-role

**forge-access-role**

```

1- {
2-   "Version": "2012-10-17",
3-   "Statement": [
4-     {
5-       "Sid": "VisualEditor0",
6-       "Effect": "Allow",
7-       "Action": [
8-         "iam:ListRoleTags",
9-         "iam:ListAttachedRolePolicies"
10-      ],
11-       "Resource": "*"
12-     }
13-   ]
14- }

```

## 步骤 2：设置 HyperPod 基础设施

按照[讲习会说明](#)，设置必要的基础设施，配置启用 Forge 功能的环境。

## 内容审核设置

如果需要访问 Nova Forge，则 Amazon Nova Lite 1.0 和 Pro 1.0 模型支持可自定义的内容审核设置 (CCMS)。CCMS 允许调整内容审核控制，以符合特定的业务需求，同时保持必要的负责任的人工智能保护措施。要确定业务模型是否适合使用 CCMS，请联系 AWS 客户经理。

有关使用自定义模型配置和使用 CCMS 的更多信息，请参阅[负责任的人工智能工具包和内容审核](#)一节。

## 持续预训练与中期训练

### Note

订阅后将提供详细文档

Nova Forge CPT 提供了超越标准 CPT 的高级功能，包括访问中间检查点以及与 Nova 预训练语料库进行数据混合。这些功能可以实现更高效的领域适应性，并更好地保留模型的通用能力。

### 什么是中间检查点？为什么需要它们？

中间检查点是 Amazon Nova 模型在预训练的不同阶段、在模型达到最终生产就绪状态之前保存的快照。在模型开发过程中，Amazon Nova 会经历多个训练阶段：使用恒定学习率进行初始预训练、学习率衰减、上下文扩展训练，以及最后的指令遵循对齐和安全训练。对于 CPT，中间检查点通常比最终的生产检查点更可取，因为它们更具可塑性，更易于领域适应。生产检查点经过了大量的指令遵循对齐和安全训练，这优化了模型的通用对话能力，但可能使其在 CPT 期间难以学习新的领域特定模式。相比之下，部分预训练和完全预训练的纯文本检查点保留了模型的预训练特征。它们尚未被强烈引导至特定行为模式，使其成为更高效的领域适应起点。当执行大规模 CPT (>100 亿个词元) 时，从中间检查点开始通常可以加快收敛速度、提高训练稳定性以及更高效地获取领域知识。然而，执行小规模 CPT (<100 亿个词元) 时，或者需要保留指令遵循能力时，生产检查点可能更合适，因为它允许在保持模型对话能力的同时进行领域适应。

CPT 需要多个中间检查点，因为它们提供了不同级别的模型可塑性，这会影响模型吸收新领域知识的效率。最终的生产检查点经过了大量的指令遵循对齐和安全训练，这优化了模型的通用对话能力，但也使其难以学习新的领域特定模式。换句话说，检查点通过训练后步骤得到了强化。相比之下，早期的检查点保留了模型的预训练特征，没有被强烈引导至特定行为，因此更具可塑性，更容易接受领域适应。

为实现最佳训练效率，我们提供了多个中间检查点。

## 有哪些可用的检查点？

### Nova 2.0

有三个 Amazon Nova Lite 2.0 检查点。

- PRE-TRAINED – [nova-lite-2/pretraining-text-RD]：这是在 Amazon Nova 预训练的恒定学习率和衰减阶段（使用数万亿个词元进行训练）之后的检查点。
- MID-TRAINED – [nova-lite-2/pretraining-text-CE]：此检查点允许以比预训练更保守的学习率引入中等数量的非结构化数据，从而吸收领域特定知识，同时避免灾难性遗忘。
- POST-TRAINED – [nova-lite-2/prod]：这是模型经过所有预训练和训练后步骤后完全对齐的最终检查点。

下表详细说明了预训练和中期训练的不同条件。

| 数据类型                          | 执行     | 使用检查点 |
|-------------------------------|--------|-------|
| 大规模的非结构化原始领域数据（文档、日志、文章、代码等）  | 持续预训练  | 预训练   |
| 大规模的非结构化原始领域数据（文档、日志、文章、代码等）  | 中期训练   | 预训练   |
| 小规模的非结构化原始数据；结构化推理轨迹/CoT 数据   | 中期训练   | 中期训练  |
| 结构化演示（高质量输入-输出对、精选任务指令、多回合对话） | 全面微调   | 中期训练  |
| 结构化演示（高质量输入-输出对、精选任务指令、多回合对话） | 参数高效微调 | 训练后   |

## 应该使用哪个检查点？

部分预训练的纯文本检查点和完全预训练的纯文本检查点通常收敛更快，并且领域适应所需的训练步骤更少。然而，它们没有经过指令调整，需要经过训练后的步骤才能执行有用的任务并遵循指令。GA 检查点可能需要更多步骤来适应，但为小规模实验提供了更安全的起点，即使在 CPT 训练后也能保持其部分训练后能力。

通常，对于大型训练数据集（>100 亿个词元），从部分预训练的纯文本检查点或完全预训练的纯文本检查点开始，可以提高训练的效率和稳定性，因为模型的知识库将大幅修改。对于小型数据集（<100 亿个词元），使用 GA 检查点可以在适应领域的同时保留指令遵循能力。

## 如何为 Nova 2.0 使用数据混合？

使用新的领域数据执行 CPT 时，将新数据与模型在预训练阶段之前使用过的部分数据混合非常有益。将旧数据与新的领域数据混合可以解决两个问题：

- **遗忘控制**：通过保留模型现有的技能和知识来防止灾难性遗忘。如果不进行数据混合，仅在狭窄领域数据上进行训练会导致模型覆盖其通用能力。例如，仅使用法律文件训练的模型可能会失去编码或进行数学运算的能力。混合通用领域数据集可以在获取新域知识的同时保留这些通用技能。
- **优化稳定性**：通过锚定模型的内部表示来保持训练稳定性。CPT 期间，模型已学习的特征会修改，而数据混合提供了来自不同来源的梯度，可以顺利地引导这种适应过程。如果没有数据混合，在狭窄分布上训练可能会导致梯度不稳定，即模型表示变化过于剧烈，从而导致训练发散、损失峰值或现有能力崩溃。这就是稳定性与可塑性的权衡：模型需要具备足够的可塑性来学习新的领域知识，同时也要足够稳定，不至于破坏已有的知识。

### Nova CPT 数据混合能力

访问 Amazon Nova 预训练数据和检查点是 Amazon Nova CPT 自定义服务的核心功能之一。Amazon Nova CPT 自定义服务可以轻松将领域数据与 Amazon Nova 的预训练语料库混合。此外，可以更改特定 Amazon Nova 数据类别（例如代码、数学、推理等）的采样比率，并控制其比例以补充领域数据。这可以在使模型适应特定领域的同时，强化与使用案例相符的能力。

### 寻找最佳混合比率

Amazon Nova 数据与领域数据的最佳比率取决于数据集的领域、复杂性、规模、质量以及保持通用能力的重要性。这个比率需要通过实验来发现。以下是决定混合多少 Amazon Nova 数据的实验框架。

选择具有代表性的领域数据子集（例如 50 亿个词元），并在所有实验运行中保持此数量不变。

运行小规模 CPT 实验，仅改变混合的 Amazon Nova 数据量：

- 不混合：100% 领域数据 → 仅 50 亿领域数据 ( 总计 50 亿 )
- 轻度混合：90% 领域数据 → 50 亿领域数据 + 约 5.6 亿 Amazon Nova 数据 ( 总计约 55.6 亿 )
- 中等混合：70% 领域数据 → 50 亿领域数据 + 约 21.4 亿 Amazon Nova 数据 ( 总计约 71.4 亿 )
- 高度混合：50% 领域数据 → 50 亿领域数据 + 50 亿 Amazon Nova 数据 ( 总计 100 亿 )

在领域内基准测试和通用领域基准测试上评估每个检查点。同时评估起始检查点 ( 任何训练开始前的 Amazon Nova 检查点 )。

- 客户领域性能在不同运行中是否大致保持不变？通常应该如此，因为每次运行都会显示相同数量的领域词元。如果领域性能随着混合量的增加而改善，说明 Amazon Nova 数据提供了有益的正则化。
- 随着混合量的增加，通用基准测试分数是否会提高？
  - 预期行为是，随着添加更多 Amazon Nova 数据，通用能力应单调提升。
  - 衡量多个通用基准测试：MMLU ( 常识 )、HumanEval ( 编码 )、GSM8K ( 数学 ) 或感兴趣的特定基准测试。
- 选择能够在保持领域性能的同时，为使用案例提供可接受的通用能力的混合比率。同时要考虑使用更多数据混合进行训练的额外成本。

确定最佳混合比率后，使用具有所选混合比率的完整领域数据集运行全面的 CPT。

## 剖析数据混合类别

下面，我们将剖析数据混合中的每个可用类别，以帮助您就哪些数据类别最适合纳入整体数据混合做出最佳决策。

### 如何启用数据混合

将 `data_mixing` 部分添加到配方，并在数据集类别中设置相应的百分比分布。`nova_data` 百分比总和必须为 100。

#### 包含数据混合的 Nova 2.0 配置

```
# Note:
# This recipe can run on p5.48xlarge

# Run config
display_name: "Nova Lite Pretrain on P5 GPU"
versions: ["2.0"]
```

```
instance_types: ["ml.p5.48xlarge"]

run:
  name: "my-cpt-run"      # A descriptive name for your training job
  model_type: "amazon.nova-2-lite-v1:0:256k" # Model variant specification, do not
change
  model_name_or_path: "nova-lite-2/prod" # Base model path, do not change
  replicas: 8             # Number of compute instances for training, allowed values are 4,
8, 16, 32
  data_s3_path: ""       # Customer data paths
  validation_data_s3_path: "" # Customer validation data paths
  output_s3_path: ""     # Output artifact path, SageMaker HyperPod job-specific
configuration - not compatible with standard SageMaker Training jobs

## Training specific configs
training_config:
  task_type: cpt
  max_length: 8192          # Maximum context window size (tokens)
  global_batch_size: 64    # Global batch size, allowed values are 32, 64, 128,
256.

  trainer:
    max_steps: 10          # The number of training steps to run total
    val_check_interval: 10 # The number of steps between running validation
    limit_val_batches: 2   # Batches of the validation set to use each trigger

  model:
    hidden_dropout: 0.0    # Dropout for hidden states, must be between 0.0 and
1.0
    attention_dropout: 0.0 # Dropout for attention weights, must be between 0.0
and 1.0

  optim:
    optimizer: adam
    lr: 1e-5               # Learning rate
    name: distributed_fused_adam # Optimizer algorithm, do not change
    adam_w_mode: true     # Enable AdamW mode
    eps: 1e-06            # Epsilon for numerical stability
    weight_decay: 0.0     # L2 regularization strength, must be between 0.0 and
1.0
    adam_beta1: 0.9       # Beta1 for Adam optimizer
    adam_beta2: 0.95     # Beta2 for Adam optimizer
    sched:
      warmup_steps: 10    # Learning rate warmup steps
```

```

constant_steps: 0          # Steps at constant learning rate
min_lr: 1e-6              # Minimum learning rate, must be lower than lr

data_mixing:
  dataset_catalog: cpt_text_lite
  sources:
    nova_data: # percent inputs for Nova data must sum to 100%; use 0% if you want to
exclude a data grouping
    agents: 20
    business-and-finance: 4
    scientific: 10
    code: 5
    factual-and-news: 5
    longform-text: 6
    health-and-medicine: 1
    humanities-and-education: 1
    legal: 1
    math: 9
    additional-languages: 15
    social-and-personal-interest: 11
    entertainment: 0.5
    reasoning: 10
    other: 0.5
    tables: 1
    customer_data: # percent input of customer data. 100 = use only customer data, 0 =
use only the nova_data mix above
    percent: 25

```

## 这些类别的含义

注意：Nova 2.0 包含了 Nova 1.0 中没有的、专门针对推理的类别（例如 reasoning-code、reasoning-math、reasoning-instruction-following）。

类别与信息标签摘要：

| 类别名称     | 详细信息                               |
|----------|------------------------------------|
| agents   | 侧重于人工智能系统中自主决策、任务完成和以目标为导向的行为的训练数据 |
| baseline | 侧重于一般理解、基本沟通和核心语言能力的基础语言数据         |

| 类别名称                            | 详细信息                           |
|---------------------------------|--------------------------------|
| chat                            | 展示自然对话流程、上下文维护和恰当社交互动的对话交流     |
| code                            | 来自各种编程语言和平台的编程源代码、文档和技术讨论。     |
| factuality                      | 侧重于准确性、来源验证和真实性评测的参考资料和经过验证的信息 |
| identity                        | 侧重于一致的角色特征、价值观和互动风格的人格框架和行为模式  |
| long-context                    | 侧重于在长篇幅交流中保持连贯性和相关性的长文本和复杂叙述   |
| math                            | 数学内容，包括教科书、问题、解决方案和数学讨论。       |
| rai                             | 强调伦理人工智能原则、安全考虑和负责任技术部署的案例和场景  |
| instruction-following           | 基于不同级别的用户提示和指令精确执行任务的示例        |
| stem                            | 涵盖科学、技术、工程和数学的技术内容，包括问题解决和理论概念 |
| planning                        | 展示战略思维、逐步任务细分和高效资源分配的序列        |
| reasoning-chat                  | 侧重于逻辑讨论和结构化对话流程的分析性对话场景        |
| reasoning-code                  | 侧重于系统性解决方案开发的编程挑战和算法问题         |
| reasoning-factuality            | 侧重于关键评估和验证过程的信息评估场景            |
| reasoning-instruction-following | 侧重于系统解释和有条不紊执行的复杂任务分析          |
| reasoning-math                  | 侧重于逻辑进程和解决策略的数学问题解决场景          |
| reasoning-planning              | 侧重于实现目标的系统性方法的战略决策场景           |
| reasoning-rag                   | 侧重于上下文理解和相关应用的信息检索和综合场景        |

| 类别名称           | 详细信息                              |
|----------------|-----------------------------------|
| reasoning-rai  | 侧重于系统评估人工智能安全性和公平性的道德决策场景         |
| reasoning-stem | 侧重于系统性分析和解决方案开发的科学问题解决场景          |
| rag            | 将检索到的外部知识与生成的响应有效结合，以提供准确上下文信息的示例 |
| translation    | 多语言内容对，在保留上下文、语气和文化细微差别的同时展示准确的翻译 |

## 参数指南

- `dataset_catalog`：目前唯一的值是 `cpt_text_lite`，直到我们启用多模态训练。
- `nova_data`：混合时，Nova 数据各个类别的百分比。它们之和应为 1.0。
- `customer_data`：混合到 Nova 数据中的客户数据的百分比。

训练中使用的词元总数可以通过  $\text{max\_length} * \text{global\_batch\_size} * \text{max\_steps}$  计算得出

## 限制

目前的 CPT 仅支持文本数据，不支持任何客户的多模态数据集。

# 监督式微调

## 介绍

监督式微调使用包含目标任务输入-输出对的数据集。换句话说，您可以提供提示（问题、指令等）示例以及正确或所需响应，并基于这些示例继续训练模型。调整模型权重以最大限度地减少监督损失，通常是其预测与目标响应词元之间的交叉熵。

## 何时使用 SFT？

当您拥有定义明确的任务且清楚预期输出时，SFT 最为适用。如果您能明确表述“给定 X 输入，正确/所需输出是 Y”，并且能收集到此类 X-Y 映射的示例，那么监督式微调是理想选择。SFT 表现出色的一些场景包括：

- 结构化或复杂的分类任务：例如，将内部文档或合同分为多个自定义类别。借助 SFT，模型学习这些特定类别的效果远超仅靠提示。
- 具有已知答案的问答或转换任务：例如，微调模型以回答公司知识库中的问题，或者在格式之间转换数据，其中每个输入都有对应的正确答案。
- 格式和风格一致性：如果需要模型始终以特定格式或语气做出响应，则可以根据正确格式/语气的示例进行微调。例如，在展示特定品牌语气或风格的提示-响应对上进行训练，可以使模型学会以该风格生成输出。同样，指令遵循行为通常也是通过在精选的良好助手行为示例上进行 SFT 来初步习得。

当您可以指定正确行为是什么样子时，SFT 是教授 LLM 新技能或行为最直接的方式。它利用模型现有的语言理解能力，并将其聚焦于任务上。如果差距在于知识而非行为，则不应使用 SFT；它无法让模型学习新的事实、术语或近期事件。在这种情况下，建议优先考虑对大规模领域内语料库进行持续预训练，或在推理时通过检索增强生成来引入外部知识。当您可以衡量质量但无法标记单个正确答案时，使用可验证的奖励或将 LLM 作为评判工具的强化微调可能比 SFT 更为可取。

根据任务复杂性以及未经微调的 Nova 模型性能，为每个任务规划数千到数万个演示样本，其中数据质量、一致性和多样性远比原始数量更为重要。

## 何时使用参数高效微调，何时使用全秩 SFT？

Nova 自定义配方使您能够执行参数高效微调（尤其是 LoRA）或全秩 SFT。如果您需要直接、具有成本效益的模型更新，或者数据量很少，建议选择参数高效微调方法，这样您只需训练小型适配器，而保留大部分主干网络不变（全秩 SFT 会更新所有模型参数）。

## 适用于 SFT 的数据混合

数据混合允许您将自定义训练数据集与 Nova 的专有训练数据相结合。此功能适用于 Nova 1.0 和 Nova 2.0 模型。

**Nova 专有数据类型：**Nova 支持文本和多模态 SFT 数据类型。它分为多个数据类别，每个类别都包含与该类别相关的多种任务。

**Nova 专有数据类别：**文本数据集包括多个类别，例如：自主决策、任务完成、以目标为导向的数据集（agents）、推理和非推理类的精确任务执行数据集（reasoning-instruction-following、instruction-following）、展示战略思维和逐步任务细分（planning）的序列、负责任的人工智能（rai）、长上下文、事实、数学、词干等等。同样，多模态数据集包括视频、屏幕截图、图表等。

数据混合功能允许您将自己的微调训练样本与用于微调 Nova 的 Nova 数据集样本进行混合。这可以防止在自定义训练上过拟合以及 Nova 能力的“灾难性遗忘”，或者在从新的预训练检查点开始训练时帮助您构建能力。

要混合 Nova 数据，只需在配方 YAML 文件的 `training_config` 部分下添加 `data_mixing` 数据块。文本数据混合块与多模态数据混合块的内容不同。请参考相应的配方。

## 支持的模型

- Nova 2.0 Lite

## 支持的模态

- 文本
- 多模态

## YAML 配置示例

### Nova 2.0 配置示例

```
run:
  name: my-lora-sft-run
  model_type: amazon.nova-2-lite-v1:0:256k
  model_name_or_path: nova-lite-2/prod
  data_s3_path: s3://my-bucket-name/train.jsonl
  replicas: 4
  output_s3_path: s3://my-bucket-name/outputs/
  mlflow_tracking_uri: ""
  mlflow_experiment_name: "my-lora-sft-experiment"
  mlflow_run_name: "my-lora-sft-run"

training_config:
  max_steps: 100
  save_steps: 10
  save_top_k: 5
  max_length: 32768
  global_batch_size: 32
  reasoning_enabled: true
  lr_scheduler:
```

```
warmup_steps: 15
min_lr: 1e-6
optim_config:
  lr: 1e-5
  weight_decay: 0.0
  adam_beta1: 0.9
  adam_beta2: 0.95
peft:
  peft_scheme: "lora"
  lora_tuning:
    alpha: 64
    lora_plus_lr_ratio: 64.0
```

## Nova 2.0 文本数据混合

```
data_mixing:
  dataset_catalog: sft_1p5_text_chat
  sources:
    customer_data:
      percent: 50
    nova_data:
      agents: 1
      baseline: 10
      chat: 0.5
      code: 10
      factuality: 0.1
      identity: 1
      long-context: 1
      math: 2
      rai: 1
      instruction-following: 13
      stem: 0.5
      planning: 10
      reasoning-chat: 0.5
      reasoning-code: 0.5
      reasoning-factuality: 0.5
      reasoning-instruction-following: 45
      reasoning-math: 0.5
      reasoning-planning: 0.5
      reasoning-rag: 0.4
      reasoning-rai: 0.5
      reasoning-stem: 0.4
```

```
rag: 1
translation: 0.1
```

## Nova 2.0 多模态数据混合

```
data_mixing:
  dataset_catalog: sft_1p5_mm_chat
  sources:
    customer_data:
      percent: 50
    nova_data:
      charts: 1
      chat: 38
      code: 20
      docs: 3
      general: 2
      grounding: 1
      rag: 4
      screenshot: 4
      text: 8
      translation: 4
      video: 15
```

## 模型检查点

### Nova 2.0 检查点

- PRE-TRAINED [nova-lite-2/pretraining-text-RD] : 在恒定学习率和衰减阶段 ( 使用数万亿个词元进行训练 ) 之后的检查点。[第 2 阶段的结果]
- MID-TRAINED [nova-lite-2/pretraining-text-CE] : 允许拥有中等数量非结构化数据的客户以比预训练更保守的学习率引入其数据，吸收领域特定知识，同时避免灾难性遗忘。[第 3 阶段的结果]
- FINAL [nova-lite-2/prod] : 完成所有预训练和训练后步骤后完全对齐的最终检查点。[第 4 阶段的结果]

### 训练阶段：

- 第 1 阶段：PT Ckpt，使用恒定学习率进行初始预训练

- 第 2 阶段：PT Ckpt，学习率衰减
- 第 3 阶段：PT Ckpt，上下文扩展训练
- 第 4 阶段：指令遵循对齐和安全训练

## 训练方法

### 训练方法选择指南

| 数据类型                          | 数据量         | 执行         | 使用检查点         |
|-------------------------------|-------------|------------|---------------|
| 大规模的非结构化原始领域数据（文档、日志、文章、代码等）  | 1 万亿+ 个词元   | 持续预训练      | 恒定学习率（CLR）结束时 |
| 大规模的非结构化原始领域数据                | 1000 亿+ 个词元 | 中期训练       | CLR 结束        |
| 少量非结构化原始数据；结构化推理轨迹/CoT 数据     | 10 亿+ 个词元   | 中期训练       | Nova 基础模型     |
| 结构化演示（高质量输入-输出对、精选任务指令、多回合对话） | 1000+ 个示例   | 监督式微调（SFT） | Nova 基础模型     |

## 开始之前的先决条件

- 我们假设您已设置 SMHP 集群，该集群包含具有活跃容量的受限实例组（RIG）。如果没有，请参阅[此处](#)完成 SMHP 集群和 RIG 设置 [[文档链接](#)、[讲习会链接](#)]
- 需要使用 p5.48xlarge EC2 实例来执行此配方。有效执行此配方所需的最小实例数量如下：
  - Nova Lite 2.0 – 4 个 p5.48xlarge
- 按照[此处](#)提供的说明，安装特定于 Forge 的 SageMaker HyperPod CLI
- 使用 `hyperpod get-clusters` 确认您可以连接到集群
  - 请注意，此命令将列出账户中的所有 SMHP 集群。

- 确认训练数据（以及可选的验证数据）位于 SMHP 集群的执行角色可以访问的 S3 存储桶中。有关数据准备，请参阅下一节。
- 已完成设置。如果尚未完成设置，请按照以下[指南](#)进行操作。
- 验证：完成设置后，确认您可以成功运行以下命令

```
aws sagemaker describe-cluster --cluster-name <cluster-name> --region <region>

hyperpod connect-cluster --cluster-name cluster-name
```

## 成功实现 SFT 的系统性方法

- 数据准备：遵循既定指南，创建、清理或重新格式化数据集，使其符合所需结构。确保输入、输出和辅助信息（例如推理轨迹或元数据）正确对齐和格式化。
- 训练配置：定义模型的训练方式。使用 Amazon SageMaker HyperPod 时，此配置是在 YAML 配方文件中编写的，包括：
  - 数据来源路径（训练和验证数据集）
  - 关键超参数（训练步骤数、学习率、批次大小）
  - 可选组件（分布式训练参数等）
  - 数据混合设置（定义客户数据和 Nova 数据类别的比例）
- 优化 SFT 超参数：我们推荐的 SFT 配方参数值是一个很好的起点，也是一个稳健的选择。如果想针对使用案例进一步优化它们，请使用不同的参数组合进行多次 SFT 运行，然后选择最佳组合。您可以按照自己选择的超参数优化方法来选择参数组合。一种简单的方法是，在保持其他参数为默认值的同时，改变一个参数的值（默认值\*0.5，默认值，默认值\*2），对每个要优化的参数重复此操作，并根据需要进行迭代。LoRa 最相关的参数是学习率、alpha（缩放参数）、用于训练的 epoch 数和预热步骤数；对于全秩微调，最相关的参数则主要是学习率、用于训练的 epoch 数和预热步骤数。

## 实验排序和数据混合

- 如果您只有一组任务的 SFT 数据（训练/开发/测试），并且只关心这些任务的测试性能
  - 在 [FINAL] Nova 检查点上执行不启用数据混合的 SFT。使用默认的 SFT 超参数，并可选择根据使用案例进行优化。监控验证指标和/或针对较大数据集评估中间检查点。

- 如果您只有一组任务的 SFT 数据（训练/开发/测试），并且关心这些任务的测试性能以及感兴趣领域的通用基准测试性能
  - 首先，在预训练检查点（PRE-TRAINED 或 MID-TRAINED 检查点，而非 FINAL 检查点）上，执行启用 Nova 数据混合的 SFT。使用中间检查点可以让模型更好地将自定义数据与 Nova 的专有数据整合，同时保持强大的通用能力。
  - 改变混合中 Nova 数据的比例（例如 10%、25%、50%、75%）以及选择补充使用案例的 Nova 数据类别（例如，如果您关心通用指令遵循能力，则选择 instruction following 类别），运行较短的 SFT 训练。监控验证指标，评估混合是否有助于提高通用基准测试的性能。选择能实现任务性能和通用性能最佳组合的训练混合比例和检查点。根据不同的使用案例，可以使用强化微调（RFT）进一步提升任务和通用性能。

## 为 SFT 准备数据集

Nova 2.0：使用 Converse API 格式 <https://docs.aws.amazon.com/bedrock/latest/userguide/conversation-inference-call.html>。Nova 2.0 数据格式可以包含其他推理字段：[https://docs.aws.amazon.com/bedrock/latest/APIReference/API\\_runtime\\_ReasoningContentBlock.html](https://docs.aws.amazon.com/bedrock/latest/APIReference/API_runtime_ReasoningContentBlock.html)

推理内容记录了模型在生成最终答案之前的中间思考步骤。在 assistant 回复中，使用 reasoningContent 字段以包含推理轨迹。请为推理内容使用纯文本，除非任务特别要求，否则避免使用 <thinking> 和 </thinking> 之类的标记标签。

## 评估方法

### 先决条件

- 来自训练作业的 manifest.json 文件的检查点 S3 URI（适用于经过训练的模型）
- 以正确格式上传到 S3 的评估数据集
- 用于评估结果的输出 S3 路径

开箱即用的基准测试：使用开箱即用的基准测试来验证通用任务的性能。有关更多详细信息，请单击此处：<https://docs.aws.amazon.com/sagemaker/latest/dg/nova-hp-evaluate.html>

### 自带数据

您也可以通过将数据格式化为以下格式，然后使用下述容器来提供自定义数据，如果需要，还可以获取推理结果以及用于校准的对数概率。

为每个任务创建具有以下结构的 jsonl：

```
{
  "metadata": "{key:4, category:'apple'}",
  "system": "arithmetic-patterns, please answer the following with no other words: ",
  "query": "What is the next number in this series? 1, 2, 4, 8, 16, ?",
  "response": "32"
}
```

评估作业在推理阶段生成的输出将具有以下结构：

```
{
  "prompt": "[{'role': 'system', 'content': 'arithmetic-patterns, please answer the following with no other words: '}, {'role': 'user', 'content': 'What is the next number in this series? 1, 2, 4, 8, 16, ?'}]",
  "inference": "['32']",
  "gold": "32",
  "metadata": "{key:4, category:'apple'}"
}
```

字段描述：

- `prompt`：发送给模型的格式化输入内容
- `inference`：模型生成的响应
- `gold`：输入数据集中的预期正确答案，即输入中的响应字段
- `metadata`：从输入中传递的可选元数据

## 准备评估配置

启动评估作业的命令。使用 `--override-parameters` 修改配方中的任何条目。

```
hyperpod start-job -n kubeflow \
  --recipe evaluation/nova/nova_micro_p5_48x1_bring_your_own_dataset_eval \
  --override-parameters '{
    "instance_type": "p5.48xlarge",
    "container": "708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-evaluation-repo:SM-HP-Eval-latest",
```

```
"recipes.run.name": "<your-eval-job-name>",
"recipes.run.model_name_or_path": "<checkpoint-s3-uri>",
"recipes.run.output_s3_path": "s3://<your-bucket>/eval-results/",
"recipes.run.data_s3_path": "s3://<your-bucket>/eval-data.jsonl"
}'
```

## 最佳实践

- 优先考虑数据质量而不是数量：高质量、多样化且具有代表性的训练数据，远比大量低质量数据更具价值。
- 包含推理型指令遵循类别：使用数据混合时，应包含“reasoning-instruction-following”类别，以保持跨任务的强大通用性能。
- 使用默认学习率：从默认学习率开始（LoRA 为  $1e-5$ ，全秩 SFT 为  $5e-6$ ），仅在需要时根据验证指标进行调整。
- 平衡 Nova 数据混合比例：最多混合 50% 的 Nova 数据，以实现延迟和性能的最佳平衡。更高的百分比可能会提高通用能力，但也会增加训练时间。
- 监控验证指标：在训练期间定期评估中间检查点，以便及早发现过拟合或性能下降。
- 在代表性数据集上测试：确保评估数据集准确代表生产使用案例，以便进行有意义的性能评估。

## 准备训练作业配置

### 超参数

除数据混合外的全套超参数：

```
## Run config
run:
  name: my-lora-sft-run
  model_type: amazon.nova-2-lite-v1:0:256k
  model_name_or_path: nova-lite-2/prod
  data_s3_path: s3://my-bucket-name/train.jsonl # SageMaker HyperPod (SMHP) only and
not compatible with SageMaker Training jobs. Note replace my-bucket-name with your
real bucket name for SMHP job
  replicas: 4 # Number of compute instances for training, allowed
values are 4, 8, 16, 32
```

```

output_s3_path: s3://my-bucket-name/outputs/          # Output artifact path
(Hyperpod job-specific; not compatible with standard SageMaker Training jobs). Note
replace my-bucket-name with your real bucket name for SMHP job

## MLFlow configs
mlflow_tracking_uri: "" # Required for MLFlow
mlflow_experiment_name: "my-lora-sft-experiment" # Optional for MLFlow. Note: leave
this field non-empty
mlflow_run_name: "my-lora-sft-run" # Optional for MLFlow. Note: leave this field non-
empty

training_config:
  max_steps: 100          # Maximum training steps. Minimal is 4.
  save_steps: 10 # This parameter suggests after how many training steps the
checkpoints will be saved. Should be less than or equal to max_steps(please override
this value with a numerical value equal or less than max_steps value; min: 4)
  save_top_k: 5          # Keep top K best checkpoints. Note supported only
for SageMaker HyperPod jobs. Minimal is 1.
  max_length: 32768      # Sequence length (options: 8192, 16384, 32768
[default], 65536)
  global_batch_size: 32  # Global batch size (options: 32, 64, 128)
  reasoning_enabled: true # If data has reasoningContent, set to true;
otherwise False

  lr_scheduler:
    warmup_steps: 15     # Learning rate warmup steps. Recommend 15% of
max_steps
    min_lr: 1e-6         # Minimum learning rate, must be between 0.0 and 1.0

  optim_config:         # Optimizer settings
    lr: 1e-5             # Learning rate, must be between 0.0 and 1.0
    weight_decay: 0.0    # L2 regularization strength, must be between 0.0
and 1.0
    adam_beta1: 0.9      # Exponential decay rate for first-moment estimates,
must be between 0.0 and 1.0
    adam_beta2: 0.95    # Exponential decay rate for second-moment
estimates, must be between 0.0 and 1.0

  peft:                 # Parameter-efficient fine-tuning (LoRA)
    peft_scheme: "lora" # Enable LoRA for PEFT
    lora_tuning:
      alpha: 64          # Scaling factor for LoRA weights ( options: 32, 64,
96, 128, 160, 192),

```

```
lora_plus_lr_ratio: 64.0 # LoRA+ learning rate scaling factor (0.0-100.0)
```

LoRa 最相关的参数是学习率、alpha ( 缩放参数 )、用于训练的 epoch 数和预热步骤数；对于全秩微调，最相关的参数则主要是学习率、用于训练的 epoch 数和预热步骤数。配方中已预填充推荐的默认值。

## 设置数据混合块

将 data\_mixing 部分添加到配方，并在数据集类别中设置相应的百分比分布。

下面介绍每个可用的 Nova 数据类别。

### 包含数据混合的 Nova 2.0 配置

```
data_mixing:
  dataset_catalog: sft_1p5_text_chat # Nova text dataset catalog
  sources:
    customer_data:
      percent: 50 # Percent of overall mix to draw from customer data
    nova_data: # The remainder will be drawn from Nova data. The
categories below must add to 100
    agents: 1 # autonomous decision-making, task completion, goal-
oriented behavior in AI systems
    baseline: 10 # [New in Nova 1.5]
    chat: 0.5 # Conversational exchanges demonstrating natural
dialogue flow
    code: 10 # Programming examples and solutions spanning
multiple languages
    factuality: 0.1 # [New in Nova 1.5]
    identity: 1 # [New in Nova 1.5]
    long-context: 1 # [New in Nova 1.5]
    math: 2 # [New in Nova 1.5]
    rai: 1 # ethical AI principles, safety considerations, and
responsible technology deployment
    instruction-following: 13 # precise task execution based on varying levels of
user prompts and directives
    stem: 0.5 # Technical content covering science, technology,
engineering, and mathematics
    planning: 10 # Sequences demonstrating strategic thinking and
step-by-step task breakdown
    reasoning-chat: 0.5
```

```

reasoning-code: 0.5
reasoning-factuality: 0.5
reasoning-instruction-following: 45
reasoning-math: 0.5
reasoning-planning: 0.5
reasoning-rag: 0.4
reasoning-rai: 0.5
reasoning-stem: 0.4
rag: 1 # combining retrieved external knowledge with
generated responses
translation: 0.1

```

这些类别的含义是什么？

## Nova 2.0 文本数据类别

| 类别名称         | 详细信息                               |
|--------------|------------------------------------|
| 代理           | 侧重于人工智能系统中自主决策、任务完成和以目标为导向的行为的训练数据 |
| 基准           | 侧重于一般理解、基本沟通和核心语言能力的基础语言数据         |
| 聊天           | 展示自然对话流程、上下文维护和恰当社交互动的对话交流         |
| 代码           | 来自各种编程语言和平台的编程源代码、文档和技术讨论。         |
| factuality   | 侧重于准确性、来源验证和真实性评测的参考资料和经过验证的信息     |
| 身份           | 侧重于一致的角色特征、价值观和互动风格的人格框架和行为模式      |
| long-context | 侧重于在长篇幅交流中保持连贯性和相关性的长文本和复杂叙述       |
| math         | 数学内容，包括教科书、问题、解决方案和数学讨论。           |

| 类别名称                            | 详细信息                              |
|---------------------------------|-----------------------------------|
| rai                             | 强调伦理人工智能原则、安全考虑和负责任技术部署的案例和场景     |
| 指令遵循                            | 基于不同级别的用户提示和指令精确执行任务的示例           |
| 词干                              | 涵盖科学、技术、工程和数学的技术内容，包括问题解决和理论概念    |
| 计划                              | 展示战略思维、逐步任务细分和高效资源分配的序列           |
| reasoning-chat                  | 侧重于逻辑讨论和结构化对话流程的分析性对话场景           |
| reasoning-code                  | 侧重于系统性解决方案开发的编程挑战和算法问题            |
| reasoning-factuality            | 侧重于关键评估和验证过程的信息评估场景               |
| reasoning-instruction-following | 侧重于系统解释和有条不紊执行的复杂任务分析             |
| reasoning-math                  | 侧重于逻辑进程和解决策略的数学问题解决场景             |
| reasoning-planning              | 侧重于实现目标的系统性方法的战略决策场景              |
| reasoning-rag                   | 侧重于上下文理解和相关应用的信息检索和综合场景           |
| reasoning-rai                   | 侧重于系统评估人工智能安全性和公平性的道德决策场景         |
| reasoning-stem                  | 侧重于系统性分析和解决方案开发的科学问题解决场景          |
| rag                             | 将检索到的外部知识与生成的响应有效结合，以提供准确上下文信息的示例 |

| 类别名称 | 详细信息                              |
|------|-----------------------------------|
| 转换   | 多语言内容对，在保留上下文、语气和文化细微差别的同时展示准确的翻译 |

## 多模态数据混合 ( Nova 2.0 )

```

data_mixing:
  dataset_catalog: sft_1p5_mm_chat      # Nova text dataset catalog
  sources:
    customer_data:
      percent: 50                        # Percent of overall mix to draw from customer data
    nova_data:                            # The remainder will be drawn from Nova data. The
categories below must add to 100
  charts: 1
  chat: 38
  code: 20
  docs: 3
  general: 2
  grounding: 1
  rag: 4
  screenshot: 4
  text: 8
  translation: 4
  video: 15

```

注意：Nova 2.0 包含 Nova 1.0 中不支持的视频数据类别支持。

这些类别的含义是什么？

### Nova 2.0 多模态数据类别

| 类别名称 | 详细信息                                           |
|------|------------------------------------------------|
| 图表   | 图形、饼图、条形图、折线图和其他统计可视化的视觉表示与描述，以帮助模型有效理解和传达定量信息 |

| 类别名称       | 详细信息                                                           |
|------------|----------------------------------------------------------------|
| 聊天         | 与视觉元素配对的对话数据，侧重于上下文对话理解和基于图像的交互                                |
| 代码         | 侧重于可视化代码解释、IDE 屏幕截图和技术图表的编程接口和开发环境                             |
| docs       | 以文档为中心的数据，将文本、图像、布局和格式相结合，训练模型理解和处理各种文档类型及结构，以帮助理解 PDF 内容识别等概念 |
| general    | 侧重于对图像、图形和随附的描述性文本进行广泛理解的多样化视觉-文本内容                            |
| Grounding  | 侧重于将语言概念与现实世界的视觉表示联系起来的视觉参考材料和标记图像                             |
| rag        | 多模态检索示例，展示如何有效结合和引用视觉及文本外部知识，以生成准确的上下文响应                       |
| screenshot | 侧重于理解软件界面和数字交互的应用程序界面截图和数字显示图像                                 |
| 文本         | 从仅文本 SFT Nova 数据集类别创建的上下文文本数据均衡池，以提供通用能力                       |
| 转换         | 侧重于图像中文本的多语言解释和文化视觉元素的跨语言视觉内容                                  |
| 视频         | 侧重于时间理解和顺序视觉叙述理解的基于动作的视觉内容                                     |

## 如何启动作业

如果您只需要获取启动首次 SFT 运行的基本细节，也可以参考 README。

容器信息：

## 容器信息和启动命令

| 模型       | 方法 | 子类别           | 映像 URI                                                                                          | Hyperpod 启动程序命令                                                                                                                                                                                                                                                                   |
|----------|----|---------------|-------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Nova 2.0 | 微调 | SFT 文本        | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-HP-SFT-V2-latest            | hyperpod start-job \ -n kubeflow \ --recipe fine-tuning/nova/nova_2_0/nova_lite/SFT/nova_lite_2_0_p5_gpu_sft \ --override-parameters '{ "instance_type": "ml.p5.48xlarge", "container": "708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-HP-SFT-V2-latest" }' |
| Nova 2.0 | 微调 | SFT 文本 + 数据混合 | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-HP-SFT-V2-DATAMIXING-latest | hyperpod start-job \ -n kubeflow \ --recipe fine-tuning/nova/forge/nova_2_0/nova_lite/SFT/nova_lite_2_0_p5_gpu_sft_ext_with_datamix \ --override-                                                                                                                                 |

| 模型 | 方法 | 子类别 | 映像 URI | Hyperpod 启动程序命令                                                                                                                                                                       |
|----|----|-----|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    |    |     |        | <pre>parameters { "instance _type": "ml.p5.48 xlarge", "container": "70897720 5387.dkr.ecr.us- east-1.amaz onaws.com/ nova-fine-tune- repo:SM-HP- SFT-V2-DA TAMIXING- latest" }</pre> |

| 模型       | 方法 | 子类别    | 映像 URI                                                                               | Hyperpod 启动程序命令                                                                                                                                                                                                                                                                   |
|----------|----|--------|--------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Nova 2.0 | 微调 | SFT MM | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-HP-SFT-V2-latest | hyperpod start-job \ -n kubeflow \ --recipe fine-tuning/nova/nova_2_0/nova_lite/SFT/nova_lite_2_0_p5_gpu_sft \ --override-parameters '{ "instance_type": "ml.p5.48xlarge", "container": "708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-HP-SFT-V2-latest" }' |

| 模型       | 方法 | 子类别           | 映像 URI                                                                                          | Hyperpod 启动程序命令                                                                                                                                                                                                                                                                                                    |
|----------|----|---------------|-------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Nova 2.0 | 微调 | SFT MM + 数据混合 | 708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-HP-SFT-V2-DATAMIXING-latest | hyperpod start-job \ -n kubeflow \ --recipe fine-tuning/nova/forge/nova_2_0/nova_lite/SFT/nova_lite_2_0_p5_gpu_sft_mm_with_datamix \ --override-parameters '{ "instance_type": "ml.p5.48xlarge", "container": "708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-HP-SFT-V2-DATAMIXING-latest" }' |

完成所有设置后，从 sagemaker-hyperpod-cli 存储库的根目录开始，导航到默认的 nova sft 配方文件夹：

- cd /src/hyperpod\_cli/sagemaker\_hyperpod\_recipes/recipes\_collection/recipes/training/nova/
- 在这里，可以根据所选的基础模型，选择运行 Nova 1 还是 Nova 2 方案。

对于 Nova 2.0 SFT：

- 如果您想使用常规的 SFT 作业，应该能在此文件夹下看到一个配方
  - `cd /src/hyperpod_cli/sagemaker_hyperpod_recipes/recipes_collection/recipes/fine-tuning/nova_2_0/nova_lite/SFT`，然后，您应该能在此文件夹下看到一个名为 `nova_lite_2_0_p5x8_gpu_sft.yaml` 的配方
- 如果想使用启用数据混合的 SFT 作业，可以导航到 SFT Forge 配方文件夹
  - `cd /src/hyperpod_cli/sagemaker_hyperpod_recipes/recipes_collection/recipes/fine-tuning/nova/forge/nova_2_0/nova_lite/SFT`，您应该能在此文件夹下看到一个名为 `nova_lite_2_0_p5x8_gpu_sft_with_datamix.yaml` 的配方
- 编辑配方中作业所需的部分，例如 `name`、`data_s3_path`、`validation_s3_path`、`output_s3_path` 和 `max_steps`。由于我们执行的是 SFT，`epoch` 的概念在此不适用。

数据混合配置看起来类似，但在底部会有一个额外的数据混合部分，类似于这样

```
data_mixing:
  dataset_catalog: sft_text_lite
  sources:
    nova_data: # percent inputs for Nova data must sum to 100%; use 0% if you want to
exclude a data grouping
    agents: 20
    business-and-finance: 20
    scientific: 20
    code: 20
    factual-and-news: 20
    longform-text: 0
    health-and-medicine: 0
    humanities-and-education: 0
    legal: 0
    math: 0
    additional-languages: 0
    social-and-personal-interest: 0
    entertainment: 0
    reasoning: 0
    other: 0
    tables: 0
    customer_data: # percent input of customer data. 100 = use only customer data, 0 =
use only the nova_data mix above
    percent: 25
```

这里有两个顶级数据类别：

- `nova_data` : 这是实际的数据混合，可进一步细分为更多类别。它们的总和必须为 100%
  - 这些类别的完整细分（包括词元计数）可在下方找到
- `customer_data` : 这是您在 YAML 顶部 `data_s3_path` 键中引用的训练数据。此处提供的百分比决定了 `nova_data` 在最终混合中的百分比。例如，在上述百分比选择中，训练期间我们将使用 25% 的 `customer_data` 和 75% 的 `nova_data`，其中 15% 将为 `agents`，15% 将为 `business-and-finance`，15% 将为 `scientific`，15% 将为 `code`，以及 15% 将为 `factual-and-news`。

提示：再次运行 `pip install-e`，就可以提交作业了！

我们将在此覆盖几个参数以使用数据混合：

```
hyperpod start-job \
  -n kubeflow \
  --recipe fine-tuning/nova/forge/nova_2_0/nova_lite/SFT/
nova_lite_2_0_p5x8_gpu_sft_with_datamix \
  --override-parameters '{
  "instance_type": "ml.p5.48xlarge",
  "recipes.run.name": "nova-sft-datamixing",
  "container": "708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-HP-
SFT-V2-Datamix",
  "recipes.run.data_s3_path": "s3://sft-data/sft_train_data.jsonl",
  "recipes.run.validation_data_s3_path": "s3://sft-data/sft_val_data.jsonl",
  "recipes.run.output_s3_path": "s3://sft-data/output/
}'
```

输出应包含作业名称，如下所示：

```
# MY Desktop # % hyperpod start-job \
  -n kubeflow \
  --recipe training/nova/forge/nova_2_0/nova_lite/sft/
nova_lite_2_0_p5x8_gpu_pretrain_with_datamix \
  --override-parameters '{
  "instance_type": "ml.p5.48xlarge",
  "recipes.run.name": "nova-sft-datamixing",
  "container": "708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-fine-tune-repo:SM-HP-
SFT-V2-Datamix",
  "recipes.run.data_s3_path": "s3://sft-data/sft_train_data.jsonl",
  "recipes.run.validation_data_s3_path": "3://sft-data/sft_val_data.jsonl",
  "recipes.run.output_s3_path": "3://sft-data/output/
```

```
}'
```

输出将如下所示：

```
Final command: python3 /local/home/my/Downloads/sagemaker-hyperpod-cli/
src/hyperpod_cli/sagemaker_hyperpod_recipes/main.py recipes=training/nova/
nova_micro_p5x8_gpu_pretrain cluster_type=k8s cluster=k8s base_results_dir=/local/home/
niphaded/Downloads/sagemaker-hyperpod-cli/results cluster.pullPolicy="IfNotPresent"
  cluster.restartPolicy="OnFailure" cluster.namespace="kubeflow"
  instance_type="p5d.48xlarge" container="900867814919.dkr.ecr.us-east-1.amazonaws.com/
nova-fine-tune-repo:sft-datamix-rig-final"
Prepared output directory at /local/home/my/Downloads/sagemaker-hyperpod-cli/results/
my-sft-run-wzdyn/k8s_templates
Found credentials in shared credentials file: ~/.aws/credentials
Helm script created at /local/home/my/Downloads/sagemaker-hyperpod-cli/results/my-sft-
run-wzdyn/niphaded-sft-run-wzdyn_launch.sh
Running Helm script: /local/home/my/Downloads/sagemaker-hyperpod-cli/results/my-sft-
run-wzdyn/niphaded-sft-run-wzdyn_launch.sh

NAME: my-sft-run-wzdyn
LAST DEPLOYED: Tue Aug 26 16:21:06 2025
NAMESPACE: kubeflow
STATUS: deployed
REVISION: 1
TEST SUITE: None
Launcher successfully generated: /local/home/my/Downloads/sagemaker-hyperpod-cli/src/
hyperpod_cli/sagemaker_hyperpod_recipes/launcher/nova/k8s_templates/SFT

{
  "Console URL": "https://us-east-1.console.aws.amazon.com/sagemaker/home?region=us-
east-1#/cluster-management/hyperpod-eks-ga-0703"
}
```

您可以使用 `hyperpod list-pods -n kubeflow --job-name my-sft-run-wzdyn` 查看作业状态

```
hyperpod list-pods -n kubeflow --job-name my-sft-run-wzdyn
{
  "pods": [
    {
      "PodName": "my-sft-run-wzdyn-master-0",
      "Namespace": "kubeflow",
```

```

    "Status": "Pending",
    "CreationTime": "2025-08-26 16:21:06+00:00"
  },
  {
    "PodName": "my-sft-run-wzdyn-worker-0",
    "Namespace": "kubeflow",
    "Status": "Pending",
    "CreationTime": "2025-08-26 16:21:06+00:00"
  }
]
}

```

或者直接使用 `kubectl` 命令查找它们。

例如，

```
kubectl get pods -o wide -w -n kubeflow | (head -n1 ; grep my-sft-run)
```

| NAME                      | RESTARTS | AGE           | IP                           | NODE | READY  | STATUS    | NOMINATED | NODE |
|---------------------------|----------|---------------|------------------------------|------|--------|-----------|-----------|------|
| my-sft-run-5suc8-master-0 |          |               |                              |      | 0/1    | Completed | 0         |      |
|                           | 3h23m    | 172.31.32.132 | hyperpod-i-00b3d8a1bf25714e4 |      | <none> |           | <none>    |      |
| my-sft-run-5suc8-worker-0 |          |               |                              |      | 0/1    | Completed | 0         |      |
|                           | 3h23m    | 172.31.44.196 | hyperpod-i-0aa7ccfc2bd26b2a0 |      | <none> |           | <none>    |      |
| my-sft-run-5suc8-worker-1 |          |               |                              |      | 0/1    | Completed | 0         |      |
|                           | 3h23m    | 172.31.46.84  | hyperpod-i-026df6406a7b7e55c |      | <none> |           | <none>    |      |
| my-sft-run-5suc8-worker-2 |          |               |                              |      | 0/1    | Completed | 0         |      |
|                           | 3h23m    | 172.31.28.68  | hyperpod-i-0802e850f903f28f1 |      | <none> |           | <none>    |      |

专业提示：务必始终使用 `-o wide` 标志，因为作业运行的 EKS 节点将帮助您在 AWS UI 中更快地找到日志

## 如何监控作业

您可以通过以下三种方式之一查看日志：

### a) 使用 CloudWatch

日志位于 Amazon Web Services 账户（包含 HyperPod 集群）的 CloudWatch 下。如需在浏览器中查看，请导航到账户中的 CloudWatch 主页，然后搜索集群名称。例如，如果集群名为 `my-hyperpod-rig`，则日志组将具有以下前缀：

- 日志组：`/aws/sagemaker/Clusters/my-hyperpod-rig/{UUID}`
- 进入日志组后，您可以使用节点实例 ID（例如 `hyperpod-i-00b3d8a1bf25714e4`）查找特定日志。
  - 此处的 `i-00b3d8a1bf25714e4` 代表运行训练作业的 hyperpod 机器友好名称。回想一下，在之前的命令 `kubectl get pods -o wide -w -n kubeflow | (head -n1 ; grep my-cpt-run)` 输出中，我们捕获了一个名为 `NODE` 的列。
  - 在本例中，“主”节点在 `hyperpod-i-00b3d8a1bf25714e4` 上运行，因此我们将使用该字符串来选择要查看的日志组。选择名为 `SagemakerHyperPodTrainingJob/rig-group/[NODE]` 的日志组

日志应该如下所示：

## b) 使用 CloudWatch Insights

如果您手边有作业名称，并且不想完成上述所有步骤，则只需查询 `/aws/sagemaker/Clusters/my-hyperpod-rig/{UUID}` 下的所有日志，即可找到单个日志。

## CPT

```
fields @timestamp, @message, @logStream, @log
| filter @message like /(?!i)Starting CPT Job/
| sort @timestamp desc
| limit 100
```

要完成作业，请将“Starting SFT Job”替换为“SFT Job completed”

然后，您可以单击结果，并选择显示“Epoch 0”的结果，因为那将是主节点。

## C) 使用 Amazon CLI

您可以选择使用 ... 来跟踪日志。在此之前，请使用 `aws --version` 检查 AWS CLI 版本。建议使用提供的工具脚本，以便在终端中实时跟踪日志

V1：

```
aws logs get-log-events \
  --log-group-name /aws/sagemaker/YourLogGroupName \
  --log-stream-name YourLogStream \
```

```
--start-from-head | jq -r '.events[].message'
```

V2 :

```
aws logs tail /aws/sagemaker/YourLogGroupName \  
  --log-stream-name YourLogStream \  
  --since 10m \  
  --follow
```

#### d) 设置机器学习流程：

您可以通过 MLFlow 跟踪指标。

##### 创建 MLFlow 应用程序

使用 Studio UI：如果您通过 Studio UI 创建训练作业，系统会自动创建默认 MLflow 应用程序，并在“高级选项”下默认选择该应用程序。

使用 CLI：如果您使用 CLI，则必须创建一个 MLflow 应用程序，并将其作为输入传递给训练作业 API 请求。

```
mlflow_app_name="<enter your MLflow app name>"  
role_arn="<enter your role ARN>"  
bucket_name="<enter your bucket name>"  
region="<enter your region>"  
  
mlflow_app_arn=$(aws sagemaker create-mlflow-app \  
  --name $mlflow_app_name \  
  --artifact-store-uri "s3://$bucket_name" \  
  --role-arn $role_arn \  
  --region $region)
```

##### 访问 MLflow 应用程序

使用 CLI：创建预签名 URL 以访问 MLFlow 应用程序 UI：

```
aws sagemaker create-presigned-mlflow-app-url \  
  --arn $mlflow_app_arn \  
  --region $region
```

```
--region $region \  
--output text
```

机器学习流程设置完成后，您可以在配方中传递 URI，或者在开始作业时使用覆盖。有关如何执行此操作的示例，请参阅 README。

## SFT 后如何评估模型？

### 先决条件

- 来自训练作业的 manifest.json 文件的检查点 S3 URI (适用于经过训练的模型)
- 以正确格式上传到 S3 的评估数据集
- 用于评估结果的输出 S3 路径

开箱即用的基准测试：使用开箱即用的基准测试来验证通用任务的性能。有关更多详细信息，请单击此处。

### 自带数据：

您也可以通过将数据格式化为以下格式，然后使用下述容器来提供自定义数据，如果需要，还可以获取推理结果以及用于校准的对数概率。

为每个任务创建具有以下结构的 jsonl：

```
{  
  "metadata": "{key:4, category:'apple'}",  
  "system": "arithmetic-patterns, please answer the following with no other words: ",  
  "query": "What is the next number in this series? 1, 2, 4, 8, 16, ?",  
  "response": "32"  
}
```

评估作业在推理阶段生成的输出将具有以下结构：

```
{  
  "prompt": "[{'role': 'system', 'content': 'arithmetic-patterns, please answer the following with no other words: '}, {'role': 'user', 'content': 'What is the next number in this series? 1, 2, 4, 8, 16, ?'}]",
```

```
"inference": "['32']",
"gold": "32",
"metadata": "{key:4, category:'apple'}"
}
```

### 字段描述：

- prompt：发送给模型的格式化输入内容
- inference：模型生成的响应
- gold：输入数据集中的预期正确答案，即输入中的响应字段
- metadata：从输入中传递的可选元数据

## 准备评估配置

启动评估作业的命令。使用“--override-parameters”修改配方中的任何条目。

```
hyperpod start-job -n kubeflow \
  --recipe evaluation/nova/nova_micro_p5_48x1_bring_your_own_dataset_eval \
  --override-parameters '{
    "instance_type": "p5.48xlarge",
    "container": "708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-evaluation-repo:SM-HP-Eval-latest",
    "recipes.run.name": "<your-eval-job-name>",
    "recipes.run.model_name_or_path": "<checkpoint-s3-uri>",
    "recipes.run.output_s3_path": "s3://<your-bucket>/eval-results/",
    "recipes.run.data_s3_path": "s3://<your-bucket>/eval-data.jsonl"
  }'
```

## 启动评估作业

不同配方及其对应映像的作业启动命令。

### 评估作业启动命令

| 模型       | 方法 | 子类别  | 映像 URI                   | 命令                               |
|----------|----|------|--------------------------|----------------------------------|
| Nova 2.0 | 评估 | Eval | 708977205387.dkr.ecr.us- | hyperpod start-job -n kubeflow \ |

| 模型 | 方法 | 子类别 | 映像 URI                                                      | 命令                                                                                                                                                                                                                                                              |
|----|----|-----|-------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    |    |     | east-1.amazonaws.com/nova-evaluation-repo:SM-HP-Eval-latest | <pre>--recipe evaluation/nova/nova/nova_2_0/nova_lite/nova_lite_2_0_p5_48_xl_gpu_ft_eval \ --override-parameters {'instance_type': "ml.p5.48xlarge", "container": "708977205387.dkr.ecr.us-east-1.amazonaws.com/nova-evaluation-repo:SM-HP-Eval-latest" }</pre> |

## 经验教训和提示

- SFT 数据集的质量至关重要。您应尽一切努力过滤掉低质量数据。如果您有一小部分在复杂性和准确性方面都极其高质量的数据，可以考虑将其放在训练结束时，以帮助模型更好地收敛。
- 我们利用文本和多模态 ( MM ) 数据集进行数据混合。我们对文本数据集的实验表明，添加 Nova 专有的“reasoning-instruction-following”类别可以显著提高通用基准测试的性能。如果您关心使用数据集进行 SFT 后通用基准测试性能下降的问题，我们建议将此类别纳入数据混合策略。
- 对于 MM 数据集，我们的实验表明，将超过 20% 的视频类别纳入混合有利于保持通用基准测试的性能。
- 此外，启用数据混合的 SFT 对学习率非常敏感，因此，基于调查发现，建议使用默认学习率进行微调，即 LoRA 为  $1e-5$ ，FR 为  $5e-6$ 。
- 最后，如果混合 Nova 专有数据集，则需要在延迟与性能之间进行权衡，因此，基于调查发现，建议最多混合 50% 以达到良好平衡。

# 强化学习

## Note

订阅后将提供详细文档

Nova Forge 提供高级强化学习能力，支持在自有环境中使用远程奖励函数。客户可选择集成自定义端点执行验证，以获取真实场景下的即时反馈，甚至可使用自定义编排工具在环境中协调代理的多轮评测。

## 使用自带编排工具进行代理多轮评测

对于需要多轮对话或奖励函数运行时长超过 15 分钟超时限制的 Forge 用户，Nova Forge 提供自带编排 (BYOO) 功能。。您可借此在自有环境中协调代理多轮评测（例如：使用化学工具对分子设计打分，或通过机器人仿真对高效完成任务给予奖励、对碰撞行为施加惩罚）。

主题

- [架构概述](#)
- [设置与执行](#)

## 架构概述

BYOO 架构依托客户自主管理的基础设施，实现对推演与生成流程的完全控制。

训练 VPC：

- 推演：将推演生成任务委托给客户基础设施，以此协调训练过程
- 训练器：根据接收到的推演结果执行模型权重更新

客户 VPC ( 如 EC2 上的 ECS )：

- 代理 Lambda：接收推演请求，并与客户基础设施进行协调
- 推演响应 SQS：用于将完成的推演结果返回到训练基础设施的队列
- 生成请求 SQS：用于模型生成请求的队列
- 生成响应 SQS：用于模型生成响应的队列

- 客户容器：实现自定义编排逻辑（可使用官方提供的初始工具包）
- DynamoDB：在整个编排过程中存储和检索状态

工作流：

1. “推演”将推演生成任务委托给“代理 Lambda”
2. “代理 Lambda”将推演 API 请求推送到“生成请求 SQS”
3. “客户容器”处理请求、管理多轮交互并调用奖励函数
4. 容器根据需要在 DynamoDB 中存储和读取状态
5. 容器将推演响应推送到“推演响应 SQS”
6. “推演”将完成的推演结果发送给“训练器”进行权重更新

## 设置与执行

有关详细设置说明、配方配置、请求和响应格式以及环境示例，请参阅提供给 Nova Forge 订阅用户的保密文档。如需获取 Nova Forge 文档，请按照以下步骤操作：

```
aws s3 cp s3://nova-forge-c7363-206080352451-us-east-1/v1/ ./ --recursive
```

下载资源后，即可在 docs 文件夹下找到所有文档。

## 负责任的人工智能工具包和内容审核

### 负责任的人工智能工具包

Nova Forge 提供了负责任的人工智能工具包，其中包含训练和评估数据，可在训练过程中使模型符合 Amazon Nova 负责任的人工智能指南，并在推理过程中实施运行时控制以调节模型响应。

**训练数据：**强调负责任的人工智能原则、安全考虑和负责任技术部署的案例和场景可用于数据混合，以便在持续预训练期间负责任地使模型对齐。

**评估：**用于测试模型检测和拒绝不当、有害或不正确内容能力的评估，作为一项基准测试任务提供，以确定基础模型性能与自定义模型性能之间的差异。

**运行时控制：**默认情况下，Amazon Nova 的运行时控制会在推理期间对模型响应进行审核。要修改这些运行时控制，请联系 AWS 客户经理，申请 Amazon Nova 的可自定义内容审核设置。

安全是 AWS 与用户的共同责任。更改基础模型或使用持续预训练来提高特定使用案例的性能，可能会影响新模型的安全性、公平性及其他特性。一种稳健的适应方法，可以最大限度地减少对基础模型中内置的安全性、公平性及其他保护措施的改变，同时最大限度地减少对模型未经自定义的任务的性能影响。需要对在代表使用案例的数据集上的应用程序进行端到端测试，以确定测试结果是否符合对安全性、公平性及其他属性的特定期望，以及整体有效性。有关更多信息，请参阅《Amazon Web Services 负责任的人工智能使用指南》、《Amazon Web Services 负责任的人工智能政策》、《Amazon Web Services 可接受使用政策》以及《Amazon Web Services 服务条款》。

## 负责任地使用

使用人工智能模型建立安全和信任措施是 AWS 与客户共同的责任。我们的目标是确保模型遵守 [AWS 可接受的使用策略](#)，在提供令人愉悦的客户体验的同时，减少不良后果。我们负责任的人工智能 (RAI) 做法围绕[负责任的人工智能的核心维度](#)进行构建。以下列表列出了这些维度。对于每个维度，我们都制定了指导方针，用于指导在整个模型开发生命周期中的决策。该生命周期涵盖了每个阶段：从初始数据收集和预训练，到部署后运行时缓解措施的实施。

- 公平：考虑对不同利益相关者群体的影响
- 可解释性：理解并评估系统输出
- 隐私和安全：合规获取、合理使用并妥善保护数据与模型
- 安全：防止有害输出和误用
- 可控性：拥有监控和引导人工智能系统行为的机制
- 准确性和可靠性：即使意外输入或对抗输入，也能实现正确的系统输出
- 治理：在人工智能供应链中采用最佳实践，包括提供商和部署者
- 透明度：让利益相关者能够就其与人工智能系统的互动做出明智的选择

## 指南要求

我们用来指导模型开发的指南包括但不限于对美化、促进或宣传以下内容的内容进行审核：

- 参与危险活动、自残或使用危险物质。
- 使用、滥用或交易受管制物质、烟草或酒精。
- 身体暴力或血腥内容。
- 虐待儿童或儿童性虐待材料。
- 虐待动物或贩运动物。
- 错误信息，例如将个人或团体视为存在蓄意欺骗行为、破坏具有公众普遍信誉的机构或危害人类健康或生计。
- 恶意软件、恶意内容或任何助长网络犯罪的内容。
- 对个人或群体的不尊重、歧视或刻板印象。
- 侮辱性言辞、亵渎性语言、淫秽手势、露骨的色情语言、淫秽作品、仇恨符号或仇恨团体相关内容。
- 除科学、教育或参考情况之外的全裸内容。
- 基于人口统计学特征对群体的偏见。

## 建议

**适用性：**人工智能模型的输出是概率性的，因此 Amazon Nova 可能会生成不准确或不适用的内容。客户应评估输出是否符合其应用场景的准确性和适当性，特别是这些输出将直接提供给最终用户时。此外，如果在需要做出重大决策的客户工作流程中使用 Amazon Nova，则客户必须评估其使用案例的潜在风险，并实施适当的人工监督、测试和其他针对特定使用案例的保护措施来降低此类风险。

**提示优化：**如果遇到 Amazon Nova 审核，请考虑根据上述指南检查所使用的提示。要使用 Amazon Nova 模型生成预期输出，建议策略是优化提示，降低产生不符合预期的结果的可能性。注意用户控制输入的情况，包括可能影响模型性能的像素内容。有关更多详细信息，请参阅本用户指南中的提示指导章节。

**隐私：**Amazon Nova 已可在 Amazon Bedrock 中使用。Amazon Bedrock 是一项托管服务，不会存储或查看客户提示或客户提示补全情况，也绝不会在客户之间共享或与 Amazon Bedrock 合作伙伴共享提示和补全信息。AWS 不会使用通过 Amazon Bedrock 服务生成的输入或输出内容来训练 Amazon Bedrock 模型，包括 Amazon Nova。有关更多信息，请参阅《AWS Service Terms》第 [50.3](#) 节和 [AWS 数据隐私常见问题](#)。有关特定服务的隐私信息，请参阅 [Amazon Bedrock 常见问题](#) 文档的“隐私和安全”部分。Amazon Nova 会采取措施来避免补全可能被解释为请求隐私信息的提示。如果用户担心自己的隐私信息已包含在 Amazon Nova 补全中，则应通过 [此处](#) 联系我们。

**安全：**包括 Amazon Nova 在内的所有 Amazon Bedrock 模型都包含企业安全功能，客户能够构建支持通用数据安全和合规标准（包括 GDPR 和 HIPAA）的生成式人工智能应用程序。客户可以使用 AWS PrivateLink 在自定义 Amazon Nova 和本地网络之间建立私有连接，而不会将客户流量暴露到互联网中。客户数据在传输过程中和静态时始终受到加密，客户可以使用自己的密钥对数据进行加密，例如使用 AWS Key Management Service。客户可以使用 AWS Identity and Access Management 来安全地控制对 Amazon Bedrock 资源的访问权限，包括自定义 Amazon Nova。此外，Amazon Bedrock 还提供全面的监控和日志记录功能，可以支持客户治理并满足审计要求。例如，Amazon CloudWatch 可以帮助跟踪审计所需的使用情况指标，并且在 Amazon Nova 与其他 AWS 系统集成时，AWS CloudTrail 可以帮助监控 API 活动并排查问题。客户也可以选择将元数据、提示和补全存储在自己的已加密 Amazon Simple Storage Service（Amazon S3）存储桶中。

**知识产权：**AWS 为已公开发布的 Amazon Nova 模型的输出提供无上限的知识产权（IP）赔偿保障（请参阅《[Service Terms](#)》第 50.10 节）。这意味着客户可以免受第三方指控这些 Amazon Nova 模型生成的输出侵犯知识产权或盗用（包括版权索赔）的索赔主张。此外，我们对使用服务的标准知识产权赔偿保障可保护客户免受第三方指控服务（包括 Amazon Nova 模型）和模型培训数据侵犯知识产权的索赔主张（包括版权索赔）。

# Amazon Nova 的配额

您的 AWS 账户对于 Amazon Bedrock 拥有默认配额（以前称为“限额”）。要查看 Amazon Bedrock 的服务配额，请执行以下操作之一：

- 按照[查看服务配额](#)中的步骤操作，然后选择 Amazon Bedrock 作为服务。
- 请参阅 AWS 一般参考中的 [Amazon Bedrock 端点和配额](#)。

## Note

若某项配额在服务配额中标注为不可调整，您可通过[提高限制表格](#)提交申请，由系统评估是否予以提高配额。

为了保持服务的性能并确保适当使用 Amazon Bedrock，分配给账户的默认配额可能会根据区域因素、付款历史记录、欺诈性使用和/或配额增加请求的批准情况进行更新。

## 请求增加 Amazon Nova 配额

您可以按照以下步骤请求为账户增加配额：

- 如果配额在 AWS 一般参考中 [Amazon Bedrock 端点和配额](#) 中的可调整列中被标记为是，则您可以按照[服务配额用户指南](#)服务配额用户指南中[请求增加配额](#)中的步骤进行调整。
- AWS 一般参考中 [Amazon Bedrock 端点和配额](#) 中的按需模型调用配额无法通过服务配额进行调整。请联系您的 AWS 账户经理，请求增加配额。

## Note

由于需求过大，将优先考虑那些产生的流量消耗了现有配额分配的客户。如果您不符合此条件，将可能拒绝您的请求。

- 即使在 AWS 一般参考中，[Amazon Bedrock 端点和配额](#) 中的可调整列将配额标记为否，您也可以通[过提高限额表单](#)提交增加配额的申请。

## 监控 Amazon Nova

在保持 Amazon Nova 和其他 AWS 解决方案的可靠性、可用性和性能方面，监控是很重要的部分。AWS 提供以下监控工具来监控 Amazon Nova，在出现问题时进行报告，并在适当的时候采取自动措施：

Amazon Nova 可通过 Amazon Bedrock 使用，因此请遵循 [Monitor the health and performance of Amazon Bedrock](#) 所述的所有监控指南。

# Amazon Nova 中的安全性

AWS 的云安全性的优先级最高。为了满足对安全性最敏感的组织的需求，我们打造了具有超高安全性的数据中心和网络架构。作为 AWS 的客户，您也可以从这些数据中心和网络架构受益。

安全性是 AWS 和您的共同责任。[责任共担模式](#)将其描述为云的安全性和云中的安全性：

- 云的安全性:AWS 负责保护在 AWS Cloud 中运行 AWS 服务的基础结构。AWS 还向您提供可安全使用的服务。第三方审核员定期测试和验证我们的安全性的有效性，作为 [AWS 合规性计划](#) 的一部分。要了解适用于 Amazon Nova 的合规性计划，请参阅 [AWS 按合规性计划提供的范围内服务](#)。
- 云中的安全性：您的责任由您使用的AWS服务决定。您还需对其他因素负责，包括数据敏感度、企业内部要求及适用法律法规。

Amazon Nova 可通过 Amazon Bedrock 使用，因此请遵循 [Security in Amazon Bedrock](#) 所述的所有安全指南。

# 使用 AWS CloudFormation 创建 Amazon Nova 资源

使用 CloudFormation 通过 Amazon Bedrock 部署 Amazon Nova 模型时，请遵循[使用 CloudFormation 创建 Amazon Bedrock 资源](#)中的所有 CloudFormation 指导。

有关更多信息，包括 YOUR-RESOURCE-PLURAL 的 JSON 和 YAML 模板示例，请参阅《AWS CloudFormation CloudFormation 用户指南》中的[YOUR-SERVICE-RESOURCE-TOPNODE-IN-CLOUDFORMATION \(在 local-phrases.ent 文件中修改此链接\)](#)。

## 了解有关 CloudFormation 的更多信息

要了解有关 CloudFormation 的更多信息，请参阅以下资源：

- [AWS CloudFormation](#)
- [《AWS CloudFormation 用户指南》](#)
- [CloudFormation API 参考](#)
- [《AWS CloudFormation 命令行界面用户指南》](#)

# 《Amazon Nova 开发人员指南》文档历史记录

下表介绍了 Amazon Nova 2.0 的文档版本。

| 变更                                   | 说明                                                                                                      | 日期              |
|--------------------------------------|---------------------------------------------------------------------------------------------------------|-----------------|
| <a href="#">Amazon Nova 2.0 初始版本</a> | 《Amazon Nova 2.0 开发人员指南》初始版本发布，新增以下模型：Nova 2 Lite、Amazon Nova Sonic 等。本指南介绍了增强的多模态功能、推理能力以及跨所有模型层的性能改进。 | 2025 年 12 月 2 日 |