PRJ-GAI-028: Fine-Tuning a Large Language Model (LLM)

Certification: AWS Certified Generative AI Developer – Professional
Domain: Model Fine-Tuning and Evaluation

---

1. Project Overview

This project demonstrates how to **fine-tune** a foundation model using **Amazon Bedrock**. While foundation models have incredible general-purpose capabilities, fine-tuning allows you to adapt them to a specific task, domain, or style. This process involves training a base model on a smaller, curated dataset of examples, which updates the model's weights to improve its performance on that specific task without altering its core capabilities.

Unlike Retrieval-Augmented Generation (RAG), which provides knowledge at inference time, fine-tuning actually changes the model itself. It is particularly effective for teaching a model a new skill, a specific response format, or a particular brand voice. We will prepare a dataset, launch a fine-tuning job in Amazon Bedrock, and then invoke our new, custom model to see the improvement in its responses.

Key Objectives

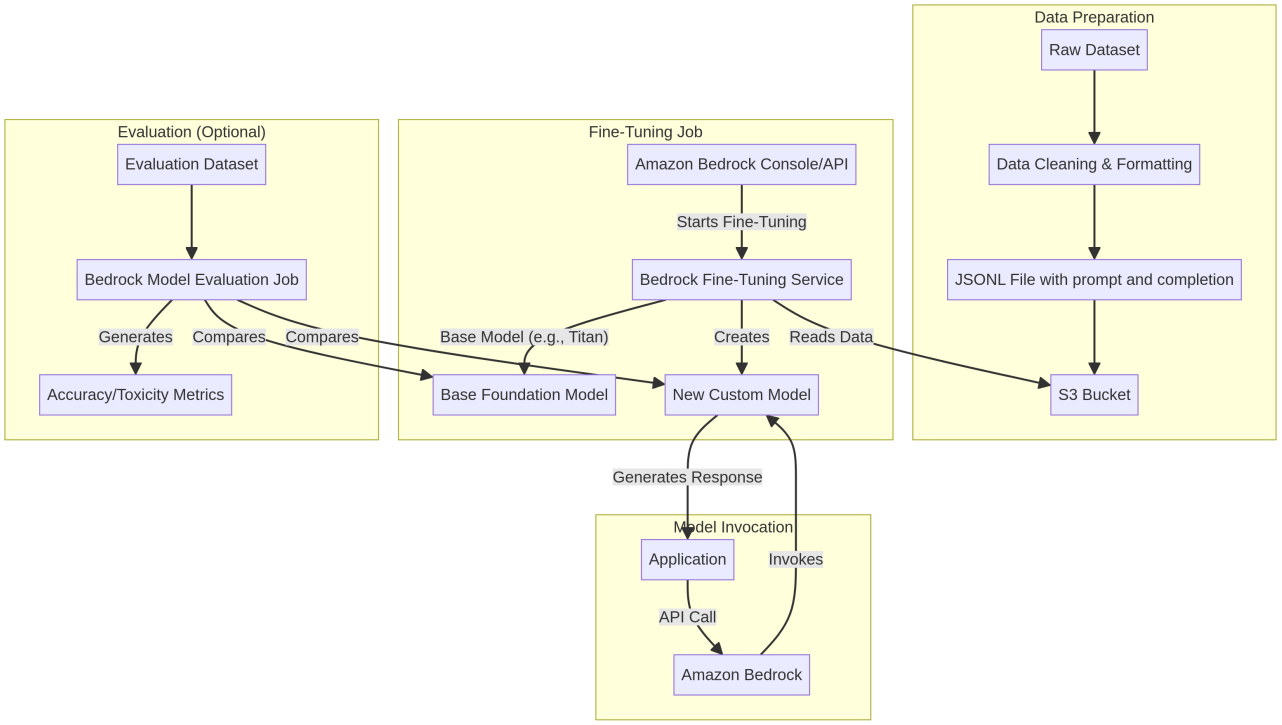
- Understand the difference between fine-tuning and RAG.
- Prepare a high-quality dataset for fine-tuning in the required JSON Lines (JSONL) format.
- Start, monitor, and manage a fine-tuning job using the Amazon Bedrock console and APIs.
- Invoke the resulting custom model with its unique ARN.
- Optionally, use Bedrock's model evaluation features to quantitatively measure the performance improvement of the fine-tuned model compared to the base

model.

---

2. Architecture

The architecture is a straightforward workflow involving data preparation, a managed fine-tuning job, and invocation of the resulting custom model.



Fine-Tuning Workflow:

1. **Data Preparation:**
    - A high-quality dataset of prompt-completion pairs is assembled. This is the most critical step for successful fine-tuning.
    - The data is cleaned, formatted, and saved as a **JSON Lines (JSONL)** file, where each line is a JSON object containing a `"prompt"` and a `"completion"` field.
    - This training file is uploaded to an S3 bucket.
2. **Fine-Tuning Job:**
    - From the Amazon Bedrock console or using the API, we start a new fine-tuning job.

- We configure the job by selecting:
  - The **base model** we want to fine-tune (e.g., Amazon Titan, Cohere Command, Meta Llama 2).
  - The S3 location of our training data.
  - Hyperparameters for the training job (e.g., number of epochs, learning rate).
- Amazon Bedrock's fully managed service then handles the entire training process. It provisions the necessary infrastructure, runs the training job, and creates a **new, private, custom model**.

3. **Model Invocation:**
   - Once the fine-tuning job is complete, the new custom model is available in our account.
   - It has a unique Amazon Resource Name (ARN).
   - Our application can now invoke this model via the Bedrock API by specifying its ARN instead of the base model's ID. The custom model will generate responses that are specialized to the task it was trained on.
4. **Model Evaluation (Optional):**
   - To get objective metrics on the model's improvement, we can run a **model evaluation job** in Bedrock.
   - This involves providing a separate evaluation dataset and choosing metrics (e.g., accuracy, toxicity, G-Eval).
   - Bedrock runs the evaluation dataset against both the base model and our new custom model, generating a side-by-side comparison report.

---

3. Prerequisites

- An AWS account with administrative permissions.
- Access to foundation models enabled in **Amazon Bedrock**.
- A prepared dataset of at least a few dozen high-quality examples.

---

4. Step-by-Step Implementation Guide

Step 4.1: Prepare the Fine-Tuning Dataset

This is the most important part of the process. The quality of your data will directly determine the quality of your fine-tuned model. Let's imagine we want to fine-tune a model to be a helpful, concise assistant for summarizing legal documents.

1. **Create the Data:** Create a file named `training-data.jsonl`. Each line must be a valid JSON object.

```
{"prompt": "Summarize the following legal clause: ...", "completion":
"This clause outlines the terms of confidentiality..."}
{"prompt": "Explain the liability section of this contract: ...",
"completion": "The liability section limits the supplier's
responsibility to the value of the contract..."}
{"prompt": "What are the key obligations in this non-disclosure
agreement: ...", "completion": "The key obligations are to maintain
secrecy, not to copy confidential materials, and to return all
documents upon termination..."}
```

   (Note: For a real job, you would need hundreds or thousands of high-quality examples.)

2. **Upload to S3:**
   - Create a new S3 bucket (e.g., `bedrock-finetuning-data-<your-account-id>`).
   - Upload the `training-data.jsonl` file to this bucket.

Step 4.2: Run the Fine-Tuning Job

1. Go to the **Amazon Bedrock console**.
2. In the left navigation pane, under **Foundation models**, click **Custom models**.
3. Click **Customize model -> Create fine-tuning job**.
4. **Select model:**

    - Choose a base model that supports fine-tuning (e.g., **Amazon Titan Text G1 - Express**).
5. **Job configuration:**
    - **Job name:** `legal-summary-assistant-v1`
    - **Custom model name:** `legal-summary-assistant-v1`
6. **Input data:**
    - **Training data S3 location:** Browse S3 and select the `training-data.jsonl` file you uploaded.
7. **Hyperparameters:**
    - For this first job, you can leave the default hyperparameters (e.g., Epochs: 10, Learning rate: 0.00001).
8. **Output data:**
    - Specify an S3 location where Bedrock can save the output logs and artifacts from the training job.
9. **Service access:**
    - Create a new IAM role that grants Bedrock permission to access your S3 buckets and other necessary services.
10. **Create fine-tuning job:** Click the button to start the job.

Step 4.3: Monitor the Job

- The job will now appear in the **Custom models** list with a status of **In progress**. Fine-tuning can take a significant amount of time (from 30 minutes to many hours), depending on the model, dataset size, and hyperparameters.
- You can click on the job to view its details and monitor its progress.
- Once complete, the status will change to **Completed**.

Step 4.4: Invoke the Custom Model

1. Go to **Playgrounds -> Text** in the Bedrock console.
2. **Select model:**
    - Under the **Category** dropdown, you will now see a **Custom** category.

- Select your `legal-summary-assistant-v1` model.

3. **Test the model:** Enter a prompt similar to the ones in your training data. The model's response should be in the concise, summary-focused style it was trained on, demonstrating an improvement over the base model for this specific task.

4. **API Invocation:** To use the model in your application, you need its ARN.
   - Go to the **Custom models** page and select your model.
   - Copy the **Model ARN**.
   - When using the Bedrock API (e.g., with Boto3), you will use this ARN in the `modelId` parameter:

```
import boto3


bedrock = boto3.client(service_name="bedrock-runtime")


response = bedrock.invoke_model(
    modelId="arn:aws:bedrock:us-east-1:123456789012:custom-
model/amazon.titan-text-express-v1:0:2abcdefg/xxxxxxxxxxxx", #
Your custom model ARN
    contentType="application/json",
    accept="*/*",
    body=json.dumps({
        "inputText": "Summarize this clause about force majeure:
...",
        "textGenerationConfig": {"maxTokenCount": 100}
    })
)
```

---

5. Best Practices

- **Data Quality is Key:** The most important factor for successful fine-tuning is a diverse, high-quality, and clean dataset. Garbage in, garbage out.

- **Start Small:** Begin with a smaller dataset and fewer epochs to iterate quickly and ensure the process is working before committing to a long, expensive training job.
- **RAG vs. Fine-Tuning:** These are not mutually exclusive. A powerful pattern is to fine-tune a model to be good at a specific task (like summarizing) and then use RAG to provide it with the specific data to summarize.
- **Evaluation:** Always have a separate test/evaluation dataset that the model has not seen during training. Use this to objectively measure the model's performance.

---

6. Cleanup

1. **Delete the custom model:** In the **Custom models** page, you can delete the fine-tuned model to avoid ongoing storage costs (if any).
2. **Delete the S3 data:** Remove the training data and the output artifacts from your S3 buckets.
3. **Delete the IAM role** created for the fine-tuning job.

Business Context

The Problem

Organizations want to leverage generative AI for content creation, customer service, and automation but face challenges with data privacy, prompt injection attacks, and hallucinations. Lack of guardrails leads to inappropriate content generation and compliance risks.

The Solution

Secure generative AI application with content filtering, prompt engineering, and responsible AI controls. Implements Amazon Bedrock with guardrails, RAG (Retrieval Augmented Generation) for accuracy, and comprehensive logging for auditability. Protects sensitive data while enabling AI innovation.

Business Value

- **Productivity Gains:** Automates content creation, reducing manual effort by 70%
- **Customer Experience:** $^{24}/_{7}$ AI-powered support improves satisfaction scores
- **Data Privacy:** Keeps sensitive data within AWS, no third-party AI exposure
- **Responsible AI:** Content filters prevent inappropriate or harmful outputs

Risk Mitigation

Prevents data leakage to third-party AI providers, blocks prompt injection attacks, filters inappropriate content, and ensures compliance with AI regulations.

GRC Mapping

Compliance Frameworks

- **NIST AI RMF:** Govern, Map, Measure, Manage
- **ISO/IEC 42001:** AI Management System
- **OWASP Top 10 for LLM:** Prompt injection, data leakage, model theft prevention
- **Responsible AI Framework:** Transparency, fairness, accountability

Security Controls Implemented

- Content filtering and moderation
- Prompt injection detection and prevention
- Data anonymization before AI processing
- Model access controls and API rate limiting
- Comprehensive logging of AI interactions

Audit Evidence

- AI interaction logs with prompts and responses
- Content filter activation records

- Data access and processing logs
- Model usage and cost tracking

Regulatory Alignment

- **AI Act (EU):** High-risk AI system requirements
- **GDPR:** Article 22 (Automated decision-making), Article 35 (DPIA)
- **CCPA:** Consumer data privacy in AI systems
- **SOC 2:** CC6.1 (Data access), CC7.2 (Monitoring)