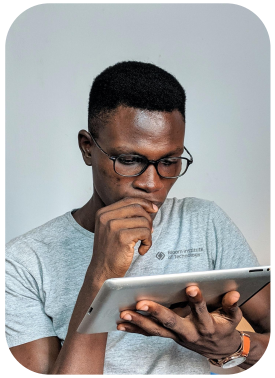
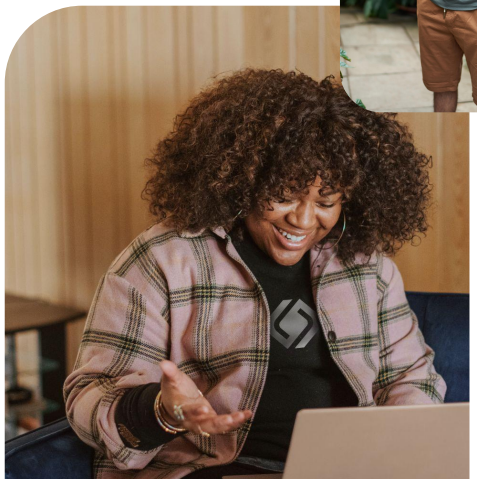
Bloom Institute of Technology

Fine-tuning

Adjust the weights of a LLM for a specific use case. This process embeds examples into the model's weights, improving output relevance.





Core Competencies

The student must demonstrate...

1. Understanding of fine-tuning and when we should use it (15 min)
2. Model selection criteria (5 min)
3. Gathering and preparing data (5 min)
4. Train the model (5 min)
5. Evaluate the model (5 min)
6. Bonus: How to use the HyDE prompt engineering pattern (10 min)

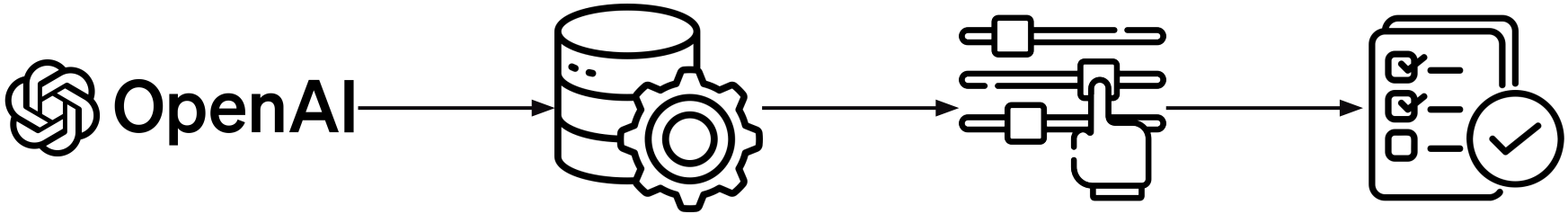
Fine-tuning Overview

1. **Select a Pre-Trained Model**: Choose a model that has been trained on large, diverse datasets.

2. **Prepare and Input New Data**: Gather and prepare a dataset specific to your task or domain. Format the data and train the model with this new data to begin the fine-tuning process.

3. **Adjust Model Weights**: The weights of the model are adjusted based on the new data, refining the model's understanding and improving its performance on specific tasks. *Weights are numerical values representing relationships between pieces of information.*

4. **Evaluate and Validate Output:** Assess the fine-tuned model to ensure it meets specific requirements and performs optimally, involving human validation to verify the accuracy and relevance of the results.



When to Consider Fine-Tuning

Fine-tuning adjusts a model's weights to better fit a specific use case, embedding examples directly into the model and reducing the need for prompts or system messages.

Use Case	Details	Considerations
Teaches Intuition Where Words Fall Short	Helps models develop an understanding that goes beyond explicit prompting.	Risk of overfitting to specific examples; may lose ability to handle diverse queries.
Trains Smaller Models to Perform Specific Tasks Better	Improves the efficiency and accuracy of smaller models for targeted tasks.	Fine-tuning may be time-consuming and computationally expensive.
Token Savings Due to Shorter Prompts	Reduces the number of tokens needed, leading to more concise cost savings.	May require significant effort to initially fine-tune; not always applicable to all tasks.
Narrows the Range of Possibilities	Focuses the model's output to be more relevant and aligned with specific use cases.	Might reduce the flexibility of the model; may need frequent updates for new data.

Check For Understanding: When should you use RAG versus fine-tuning a model?

Model Selection

Selecting an LLM to fine-tune involves matching the model to specific needs. LLM benchmarks provide a standardized framework to compare models' capabilities in specific tasks. Evaluate LLMs on Chatbot Area.

Task	Benchmark	Evaluation Criteria
Chatbot Assistance	ChatBot Arena, MT Bench	Conversational fluency (smooth and natural dialogue), task success rate (ability to achieve specific goals)
Language Understanding	MMLU, SuperGLUE	Task diversity (variety of tasks and domains), reasoning ability (depth of comprehension and logical thinking)
Reasoning	ARC, HellaSwag	Logical reasoning (ability to follow logical steps), commonsense performance (understanding everyday scenarios)
Coding	HumanEval, MBPP, SWE-bench	Code functionality (correctness and efficiency of generated code), problem-solving accuracy (ability to solve given problems)

Metrics Used for Comparing Performance Across Tasks

1. **Accuracy**: Percentage of correct answers.
2. **BLEU Score**: Alignment of generated text with human references.
3. **Perplexity**: Model's surprise/confusion; lower is better.
4. **Human Evaluation**: Expert judgment on quality, relevance, or coherence.

Fine-Tune GPT-3.5 to Generate Better Unit Tests

Objective: Showcase the fine-tuning process from start to finish, focusing on training `GPT-3.5` to generate unit tests at a cheaper price. Follow the complete code using this repo.

Overview of Tasks

1. **Synthesize Data:** Generate fake data that accurately mimics pirate-themed dialogue. Ensure the data is diverse and covers various aspects of unit testing in Python.

2. **Format Data:** Organize and structure the synthesized data. Format the data according to OpenAI's fine-tuning specifications.

3. **Fine-Tune the Model:** Use the OpenAI to fine-tune `GPT-3.5` with the prepared data.

4. **Evaluate the Outputs:** Assess the fine-tuned model's performance and ensure it meets the desired stylistic criteria and accuracy.

Synthesize and Format Data

1. **Setup LangSmith Connection:** Configure LangSmith connection to save the generated dataset.

2. **Define Generate Function**: `generate_unit_tests` creates unit tests with a `SystemMessage`. This workaround speeds up data generation for class.

3. **Create Partial Functions**: Use partial to define functions to generate unit tests for multiple functions concurrently and the save the data directly on LangSmith.

4. **Invoke Chain for Reviews**: Combine the prompt and LLM into a chain and invoke it for each review to generate responses efficiently.

5. **Format for OpenAI**: Ensure the output matches the format required to fine-tune OpenAI models.

Sample Format Required to Fine-tune OpenAI:

```
{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role":"user", "content": "What's the capital of France?"}, {"role": "assistant", "content": "Paris, as if everyone doesn't know that already."}]}
```

Fine-tune GPT-3.5

Use a training file to fine-tune OpenAI's `GPT-3.5` using the API directly. Format the data, create JSONL files, and start a tuning job. Access the complete step-by -step procedure here.

1. **Setup and Initialization**:
    a. Import necessary modules.
    b. Initialize the OpenAI client using the API key from environment variables.
2. **Prepare Data**:
    a. Define the names of the training and validation data files.
    b. Write the provided training and validation data into JSONL files.
3. **Upload Data Files**:
    a. Upload the training and validation data files to OpenAI.
    b. Retrieve and print the file IDs for reference.
4. **Create Fine-Tuning Job**:
    a. Use the uploaded files to create a fine-tuning job with specific hyperparameters.
    b. Retrieve and print the job ID and initial status.
5. **Handle Interruptions and Stream Events**:
    a. Set up a signal handler to manage interruptions.
    b. Stream and print events related to the fine-tuning job, handling potential disconnections.
6. **Check Other Jobs and Retrieve Model ID**:
    a. Retrieve and print the ID of the fine-tuned model.

Hands-on Homework.

Fine-tune Company Standards using OpenAI

1. **Compile Standards**: Gather and document company coding standards and guidelines.
2. **Generate Synthetic Data**: Create synthetic data sets that demonstrate code adhering to these guidelines.
3. **Fine-Tune and Evaluate**: Fine-tune GPT-3.5 with this synthetic data and evaluate its performance to assess improvements.
4. **Increase Data for Reliability**: Expand the synthetic data set to further enhance the model's reliability in following the guidelines.