Fine-tuning

Overview

Finetuning a model means updating the model itself over a set of data to improve the model in a variety of ways. This can include improving the quality of outputs, reducing hallucinations, memorizing more data holistically, and reducing latency/cost.

The core of our toolkit revolves around in-context learning / retrieval augmentation, which involves using the models in inference mode and not training the models themselves.

While finetuning can be also used to "augment" a model with external data, finetuning can complement retrieval augmentation in a variety of ways:

Embedding Finetuning Benefits

- Finetuning the embedding model can allow for more meaningful embedding representations over a training distribution of data --> leads to better retrieval performance.

LLM Finetuning Benefits

- Allow it to learn a style over a given dataset
- Allow it to learn a DSL that might be less represented in the training data (e.g. SQL)
- Allow it to correct hallucinations/errors that might be hard to fix through prompt engineering
- Allow it to distill a better model (e.g. GPT-4) into a simpler/cheaper model (e.g. gpt-3.5, Llama 2)

Integrations with LlamaIndex

This is an evolving guide, and there are currently three key integrations with LlamaIndex. Please check out the sections below for more details!

- Finetuning embeddings for better retrieval performance
- Finetuning Llama 2 for better text-to-SQL
- Finetuning gpt-3.5-turbo to distill gpt-4

Ask AI

Finetuning Embeddings

We've created comprehensive guides showing you how to finetune embeddings in different ways, whether that's the model itself (in this case, `bge`) over an unstructured text corpus, or an adapter over any black-box embedding. It consists of the following steps:

1. Generating a synthetic question/answer dataset using LlamaIndex over any unstructured context.
2. Finetuning the model
3. Evaluating the model.

Finetuning gives you a 5-10% increase in retrieval evaluation metrics. You can then plug this fine-tuned model into your RAG application with LlamaIndex.

- Fine-tuning an Adapter
- Embedding Fine-tuning Guide
- Router Fine-tuning

Old

- Embedding Fine-tuning Repo
- Embedding Fine-tuning Blog

Fine-tuning LLMs

Fine-tuning GPT-3.5 to distill GPT-4

We have multiple guides showing how to use OpenAI's finetuning endpoints to fine-tune gpt-3.5-turbo to output GPT-4 responses for RAG/agents.

We use GPT-4 to automatically generate questions from any unstructured context, and use a GPT-4 query engine flow to generate "ground-truth" answers. Our `OpenAIFineTuningHandler` callback automatically logs questions/answers to a dataset.

We then launch a finetuning job, and get back a distilled model. We can evaluate this model with Ragas to benchmark against a naive GPT-3.5 flow.

Ask AI

- GPT-3.5 Fine-tuning Notebook (Colab)
- GPT-3.5 Fine-tuning Notebook (Notebook link)
- React Agent Finetuning

- [WIP] Function Calling Fine-tuning

Old

- GPT-3.5 Fine-tuning Notebook (Colab)
- GPT-3.5 Fine-tuning Notebook (in Repo)

Fine-tuning for Better Structured Outputs

Another use case for fine-tuning is to make the model better at outputting structured data. We can do this for both OpenAI and Llama2.

- OpenAI Function Calling Fine-tuning
- Llama2 Structured Output Fine-tuning

Fine-tuning Llama 2 for Better Text-to-SQL

In this tutorial, we show you how you can finetune Llama 2 on a text-to-SQL dataset, and then use it for structured analytics against any SQL database using LlamaIndex abstractions.

The stack includes `sql-create-context` as the training dataset, OpenLLaMa as the base model, PEFT for finetuning, Modal for cloud compute, LlamaIndex for inference abstractions.

- Llama 2 Text-to-SQL Fine-tuning (w/ Gradient.AI)
- Llama 2 Text-to-SQL Fine-tuning (w/ Modal, Repo)
- Llama 2 Text-to-SQL Fine-tuning (w/ Modal, Notebook)

Fine-tuning An Evaluator

In these tutorials, we aim to distill a GPT-4 judge (or evaluator) onto a GPT-3.5 judge. It has been recently observed that GPT-4 judges can reach high levels of agreement with human evaluators (e.g., see https://arxiv.org/pdf/2306.05685.pdf).

Thus, by fine-tuning a GPT-3.5 judge, we may be able to reach GPT-4 levels (and by proxy, agreement with humans) at a lower cost.

- Fine-tune LLM Correctness Judge
- Fine-tune LLM Judge

Ask AI

Fine-tuning Cross-Encoders for Re-Ranking

By finetuning a cross encoder, we can attempt to improve re-ranking performance on our own private data.

Re-ranking is key step in advanced retrieval, where retrieved nodes from many sources are re-ranked using a separate model, so that the most relevant nodes are first.

In this example, we use the `sentence-transformers` package to help finetune a crossencoder model, using a dataset that is generated based on the `QASPER` dataset.

- Cross-Encoder Finetuning
- Finetuning Llama 2 for Text-to-SQL
- Finetuning GPT-3.5 to Distill GPT-4

Cohere Custom Reranker

By training a custom reranker with CohereAI, we can attempt to improve re-ranking performance on our own private data.

Re-ranking is a crucial step in advanced retrieval processes. This step involves using a separate model to re-organize nodes retrieved from initial retrieval phase. The goal is to ensure that the most relevant nodes are prioritized and appear first.

In this example, we use the `cohere` custom reranker training module to create a reranker on your domain or specific dataset to improve retrieval performance.

- Cohere Custom Reranker

Ask AI