

Augmenting LLMs: Fine-Tuning or RAG?

 **Damien from the AiEdge** <aiedge@substack.com>

Mon, Sep 18, 2023 at 8:30 PM

Reply-To:  Damien from the AiEdge


<reply+29gl5a&1h4rbj&&7cbef4d620dcf1106746c3bb03ed93168bc9ab9f851ad89f97ba50699978432f@mg1.substack.com>

To: gurusriniv81@gmail.com

Forwarded this email? [Subscribe here](#) for more



The AiEdge NEWSLETTER

 *Hey Damien here! This is a sneak peek of today's paid newsletter for our premium subscribers. Get access to this issue and all future issues by subscribing here:*

Upgrade to paid

Here are the latest articles you may have missed:

- Introduction to LangChain: Augmenting LLMs with Tools
- Introduction to LangChain: Retrieval Augmented Generation
- Deep Dive: How to Build an Image Retrieval System like Google Images
- Introduction to LangChain: Vector Database Basics

Augmenting LLMs: Fine-Tuning or RAG?

DAMIEN BENVENISTE

SEP 18 · PREVIEW



READ IN APP ↗

- *Fine-Tuning VS RAG*
- *The cost of fine-tuning*
- *The problems with RAG*

Fine-Tuning VS RAG

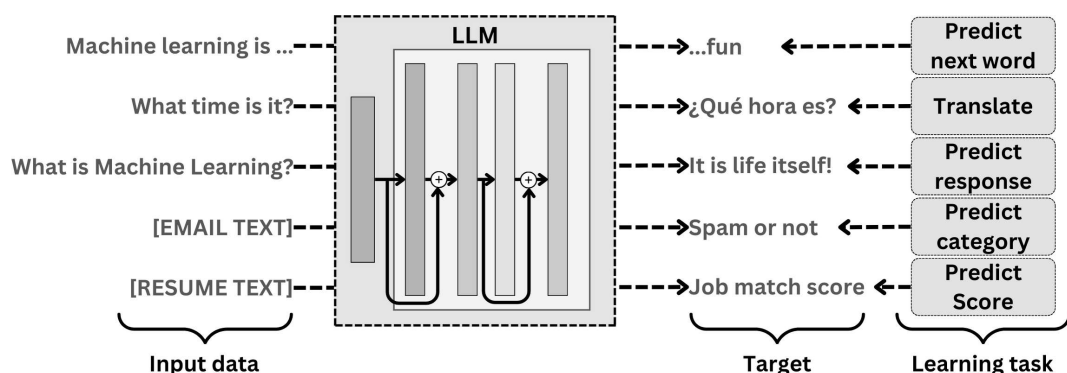
The unnecessary debate

Let's say you have a business use case for using an LLM, but you need it to be a bit more specialized to your specific data. I often see the problem being simplified to Fine-Tuning vs. Retrieval Augmented Generation (RAG), but this is somewhat of a false dichotomy. Those are two ways to augment LLMs, but each approach has different applications, and they can even be used in a complementary manner.

What is fine-tuning?

Fine-tuning assumes you will continue training an LLM on a specific learning task. For example, you may want to fine-tune an LLM on the following tasks:

- English-Spanish translation
- Custom support message routing
- Specialized question-answers
- Text sentiment analysis
- Named entity recognition
- ...

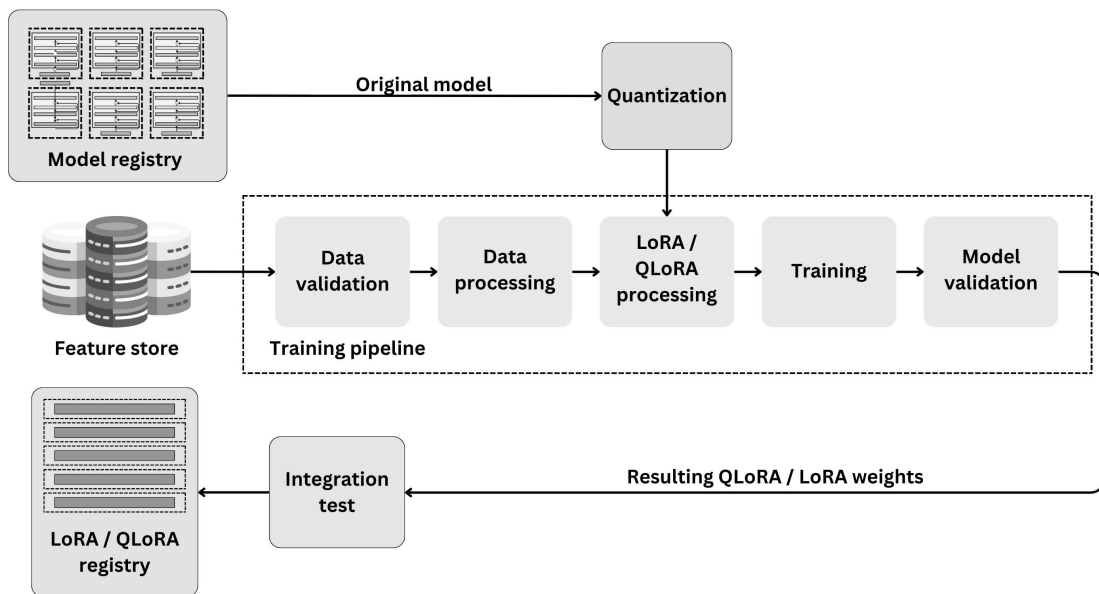


Fine-tuning assumes you have training data to specialize an LLM on a specific learning task. That means you need to be able to identify the correct input data, the proper learning objective, and the right training process.

System design for fine-tuning

Regardless of the specific application, fine-tuning will require a training pipeline, a serving pipeline, and potentially a continuous training pipeline.

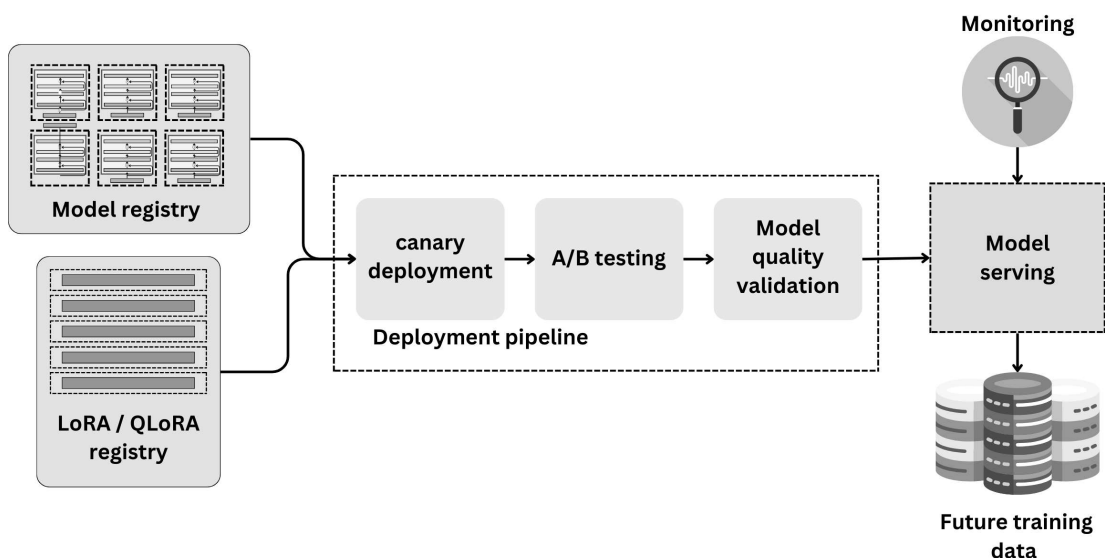
The training pipeline will need the following components:



- **The model registry:** The model registry will contain the original models or different versions of fine-tuned models. A model registry requires capturing the model version and additional metadata describing the model.
- **The quantization module:** It is typical to quantize models these days to save on memory costs, but this is not a requirement. If the model is quantized, the resulting model will need to be stored in the model registry. Quantization means converting the model weights from floats to integers. This operation can divide the model size by 4.
- **The feature store:** As for any ML model training, the data needs to be carefully prepared and stored in a feature store. In the context of LLMs, the requirements around the feature store may be relaxed if the data at serving time is only generated by a user.
- **The data validation and pre-processing modules:** When the data is injected into the training pipeline, it needs to be validated and most likely pre-processed for training.

- **LoRA / QLoRA modifications:** It is now typical to fine-tune LLMs using Low-Rank Adapters (LoRA) or its Quantized version (QLoRA). The idea is to replace, within the model, some of the large matrices with smaller ones for the gradient computation. When we fine-tune, we only update the weights of those newly inserted matrices. The gradient matrices are much smaller and, therefore, require much less GPU memory space. Because the pre-trained weights are frozen, we don't need to compute the gradients for a vast majority of the parameters.
- **The LoRA / QLoRA registry:** With LoRA / QLoRA, we need a registry to keep track of the fine-tuned weights and their versions.
- **Model validation module:** Like any model training, the resulting model needs to be validated on validation data. This assumes we have the right data for the task. This can be tricky because we may want to fine-tune a model on a specific task, but we may also want to retain its original capabilities. For the specific task, you may have the right validation data, but you may be missing the data you need for the original capabilities, leaving you unable to assess if the model is not forgetting its previous programming.

The serving pipeline will need the following components:

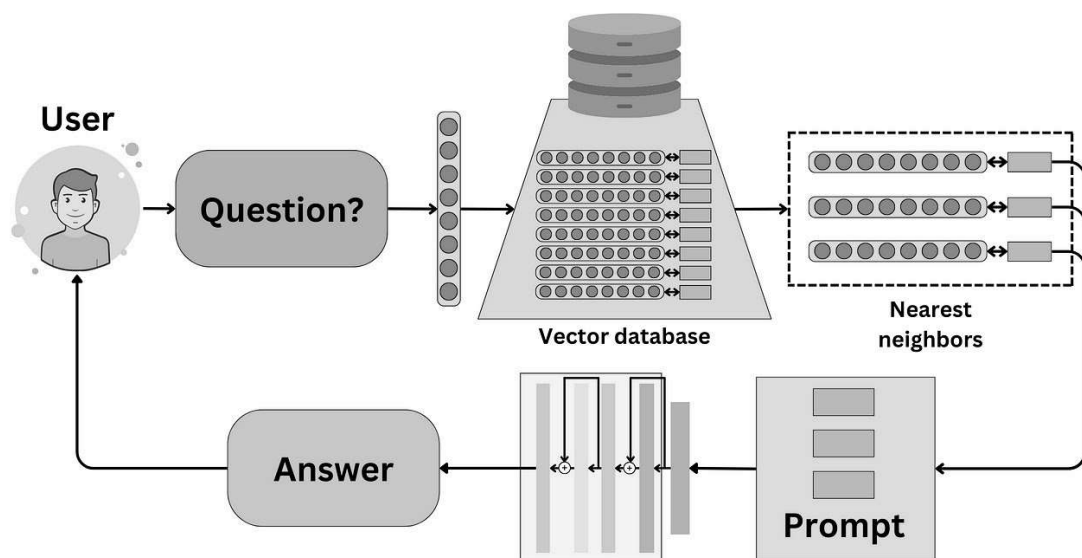


- Once ready for deployment, the model and its related LoRA / QLoRA weights are pulled from the model registry and passed through the deployment pipeline. We may have a series of tests, such as canary deployment, to make sure the model fits in a serving pipeline and an A/B test experiment to test it against the production model. After satisfactory results, we can propose the new model as a replacement for the production one.

- The model needs to be continuously monitored using typical ML monitoring systems
- As soon as the users interact with the served model, this may be an opportunity to start aggregating data for the next training update.

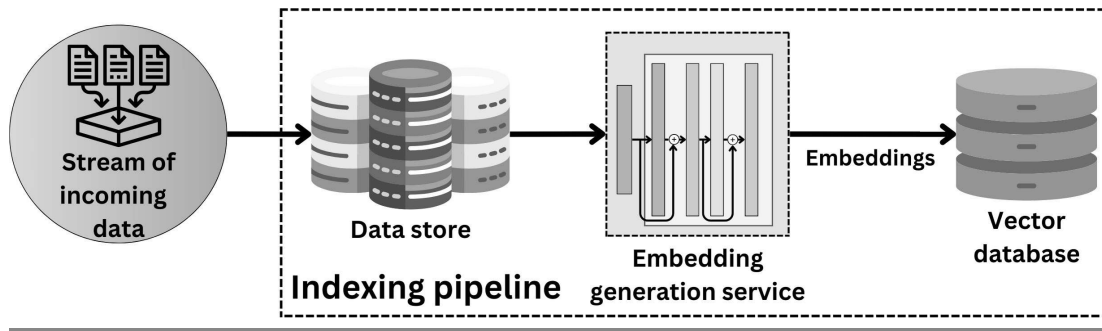
What is RAG?

Retrieval Augmented Generation or RAG means that you expose an LLM to new data stored in a database. We don't modify the LLM; rather, we provide additional data context in the prompt for the LLM to answer questions with information on the subject. The idea with RAG is to encode the data you want to expose to your LLM into embeddings and index that data into a vector database. When a user asks a question, it is converted to an embedding, and we can use it to search for similar embeddings in the database. Once we found similar embeddings, we construct a prompt with the related data to provide context for an LLM to answer the question. Similarity here is usually measured using the cosine similarity metric.



System design for RAG

For RAG, we don't have a training pipeline. We just need an indexing pipeline and a serving pipeline. The indexing pipeline is used to convert the data into vector representation and index it in a Vector database:



Keep reading with a 7-day free trial

Subscribe to

The AiEdge Newsletter

to keep reading this post and get 7 days of free access to the full post archives.

Start trial

A subscription gets you:

- ✓ Subscriber-only weekly posts and full archive
- ✓ Post comments and join the community
- ✓ Access to all the AiEdge's Deep Dives



LIKE



COMMENT



RESTACK

© 2023 AiEdge

548 Market Street PMB 72296, San Francisco, CA 94104

[Unsubscribe](#)

Get the app



Start writing