Featured LLM

Optimizing RAG systems with fine-tuning techniques

January 19, 2024 8 min

Your Email

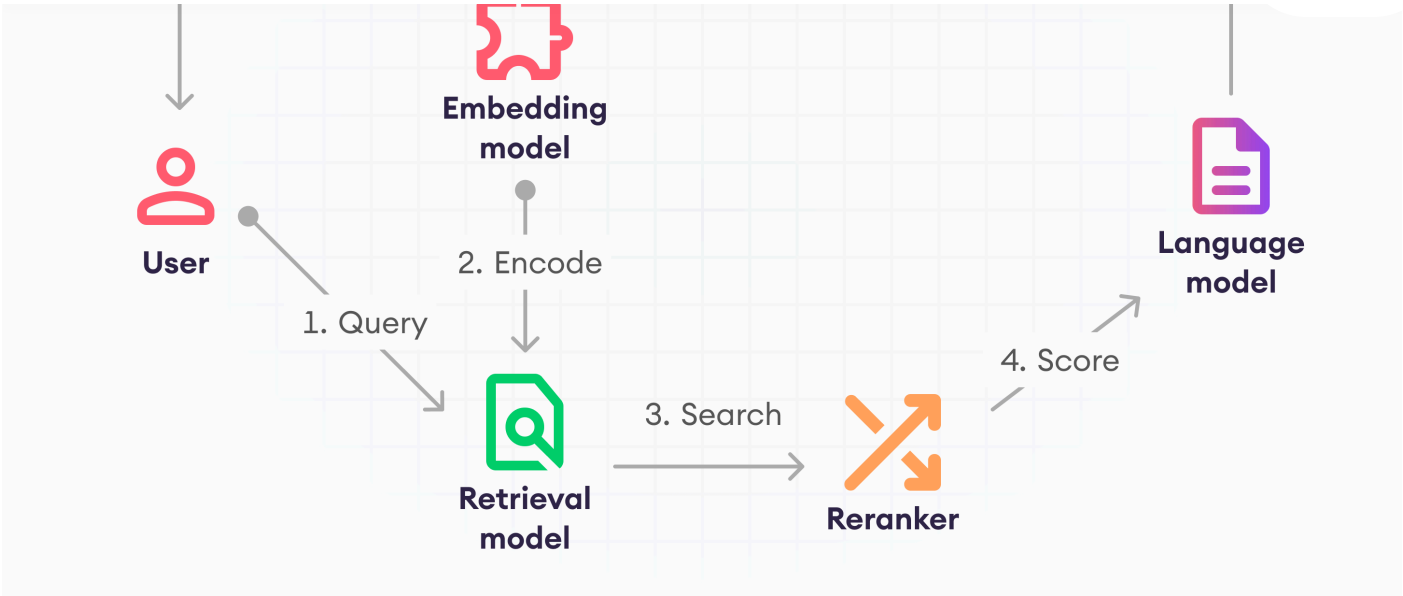
Fine-tuning a retrieval augmented generation (RAG) system has become increasingly relevant with the rapid evolution of large language models (LLMs). Implementing RAG systems using tools like HuggingFace, open-source vector databases, and LLM-APIs presents unique challenges and opportunities. While these systems show promising results in question-answering tasks on benchmarks, they sometimes fail to provide accurate results on custom datasets.

Sometimes, the system struggles to answer questions even when the necessary information is in the dataset, raising concerns about how well its components work. The key to enhancing RAG systems lies in understanding why these inconsistencies occur, identifying the underlying issues, and applying targeted solutions.

Turns out that fine-tuning the troublesome component of the RAG system can be an effective method to improve the system's performance. This article aims to shed light on evaluating RAG system components, finding the defect, and using fine-tuning to "cure" that component.

The components of a retrieval augmented generation system

- **An embedding model** that encodes document passages as vectors.
- **A retriever** that runs a question through the embeddings model and returns any encoded documents near the embedded question.
- **A reranker** (optional) that returns a relevance score when given a question and record.
- **A language model** that receives the records from the retriever or reranker together with the question and returns an answer.



When building a RAG system, it's easy to assume that taking the highest-performing models on their respective benchmarks on HuggingFace would yield great results; after all, they performed great on RAG, right? It turns out that there are more accurate techniques than evaluating models solely based on the model's performance.

The issue with benchmarks for large language models

A high benchmark score only really says anything if the type of questions and documents you use are represented in the dataset used for the benchmark. When building RAG systems for specialized external knowledge or internal company data, your existing data will likely differ from the one used to test the models, which might cause several issues.

- Documents or questions might be very similar to anyone but a particular domain expert. A more generic embedding model may be unable to differentiate between these.
- Similarly, a reranker might need to correctly understand domain-specific data nuances, like the jargon in the field, marking relevant information as irrelevant or vice versa.

Evaluation criteria – finding the defective component

Now that we know what can go wrong, it's time to move on to improve the model. To improve our model, we need to establish a way of evaluating the performance of the entire RAG system. Understanding the technicalities of the performance is critical to identifying the source of the low performance and comparing versions with different components. However, understanding how to conduct the evaluation can be tricky. We need to create a set of criteria to identify which system component we should focus our improvement efforts on to measure when the system improves. An example of criteria that would be able to locate the issue are:

- **Document relevance:** Do the retrieved documents contain relevant data for the query?
- **Reranking relevance:** Are the reranked results more relevant than before?
- **Correctness:** Does the model answer the question correctly based on the supplied documents?
- **Hallucination:** Did the model add data that weren't mentioned in the documents?

It's essential to develop a grading rubric to ensure consistency, especially when more than one person is involved in the rating. SuperAnnotate worked with Databricks on a project where it helped evaluate their Documentation Bot, and you can find an example grading rubric in this article. One of their ML engineers wrote about the project in this article where you can also find an example grading rubric.

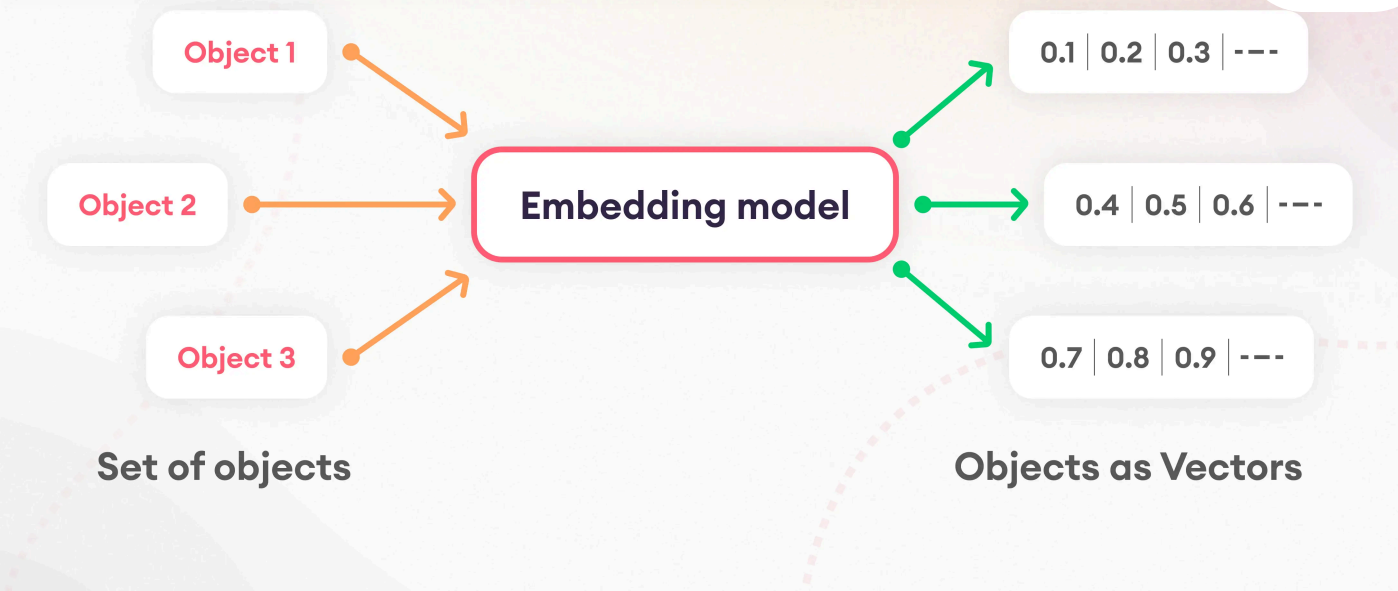
Solutions - fine-tune the defective component

the rubric. You'll likely find that at least one of the evaluation scores is relatively low, and that'll determine where we start our improvement work. The key to many of the solutions below is to fine-tune the underperforming component, thus ensuring that your system benefits from both fine-tuning and RAG. The exact approach will differ depending on your language model, but let's outline the general methods for each component below.

Embeddings

If the document relevance score is low, your model isn't returning the documents that contain the answer to the question, or it produces a lot of non-relevant data. An embedding model transforms a piece of text into a vector. In the rag system, we generally chunk our dataset into smaller segments, encode them all to vectors using our model, and then store them in a vector database. We then encode our question and hope that the resulting vector is close to the vector of the document with the answer.

We must create a dataset of question and document pairs to fine-tune an embedding model. These can be positive pairs, meaning that the document can answer the question, and negative pairs, where the record can't answer the question. A common choice of embedding model is from the SentenceTransformers library, and you can find details on how to fine-tune those models here. If you use models from the BGE family, the following resource can be a good place to learn more about fine-tuning those.



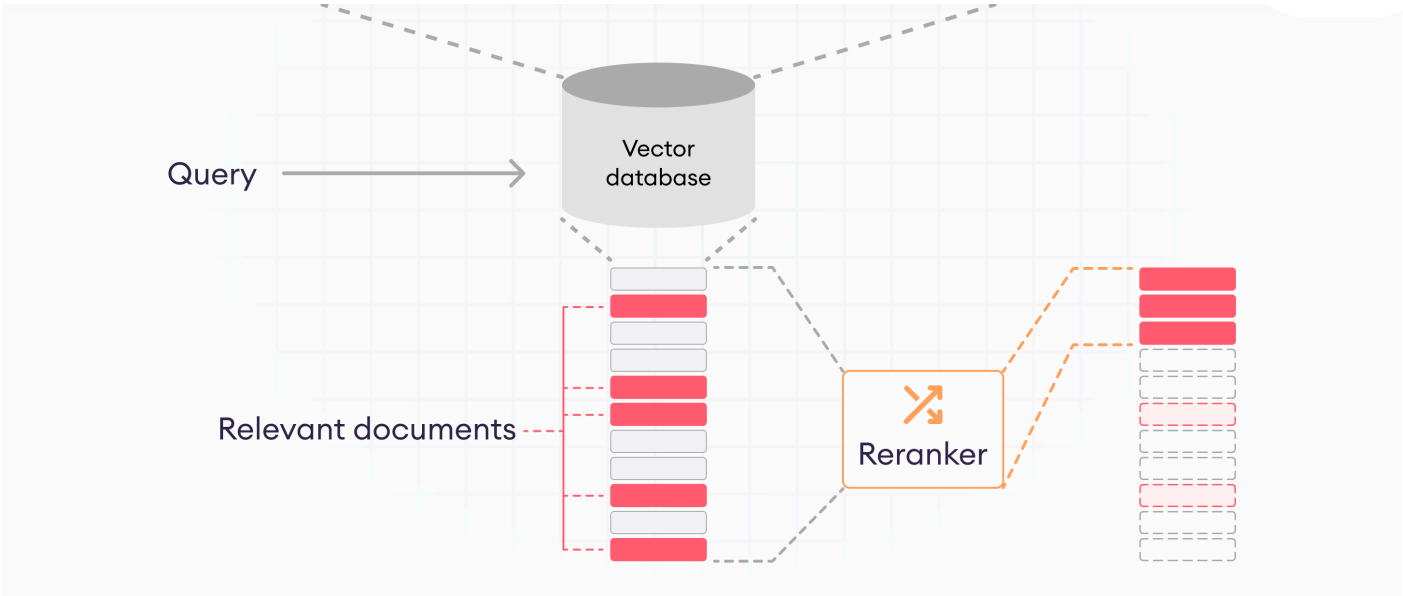
Reranker

An initial list of potential matches is reordered in the reranking part of the pipeline. One may ask if that differs from what the embeddings do. Why have a reranker?

One answer here is that embeddings create a vector that is essentially a compressed version of the text. When comparing the similarity between the compressed question and the document, some information will unfortunately be lost. The reranker, on the other hand, computes a similarity score based on the uncompressed versions of the question and answer. By this, it achieves a higher quality similarity calculation but at higher computational costs.

Another possibility is that we combine several types of search systems. Embeddings search may be used with a classic word-matching search or other methods. Here, the reranker takes the list of all matches and orders them based on relevance regardless of source.

If the reranker is underperforming, fine-tuning can be used with a task-specific dataset of question-and-answer pairs like the embedding model.



Large language model

It’s possible that the LLM used isn’t performing well when answering the question about the task-specific data. This may be due to the language model used, and a good first step is to try out a few different LLMs to see which one is best.

Starting with a large model like GPT-4 can be a good start, but there could be constraints in production due to cost, data security concerns, or a combination of both. While open-source models are an alternative, their immediate performance often doesn't measure up to the standards set by OpenAI's offerings.

Most LLMs are first pre-trained on a large corpus of training data to learn to predict the next token given some text. Supervised fine-tuning (training data of question/answer pairs) and reinforcement learning are used to shape these language models to perform specific tasks. If you can access the data used to train a model, you might discover the prompting style used for its RAG capabilities. Using a similar style of prompting could then lead to the best performance. A model properly trained to work on RAG will only use the information available in the provided documents and decline to answer if the provided documents don’t contain the answer. Otherwise, experimenting with different versions of a prompt might improve results.

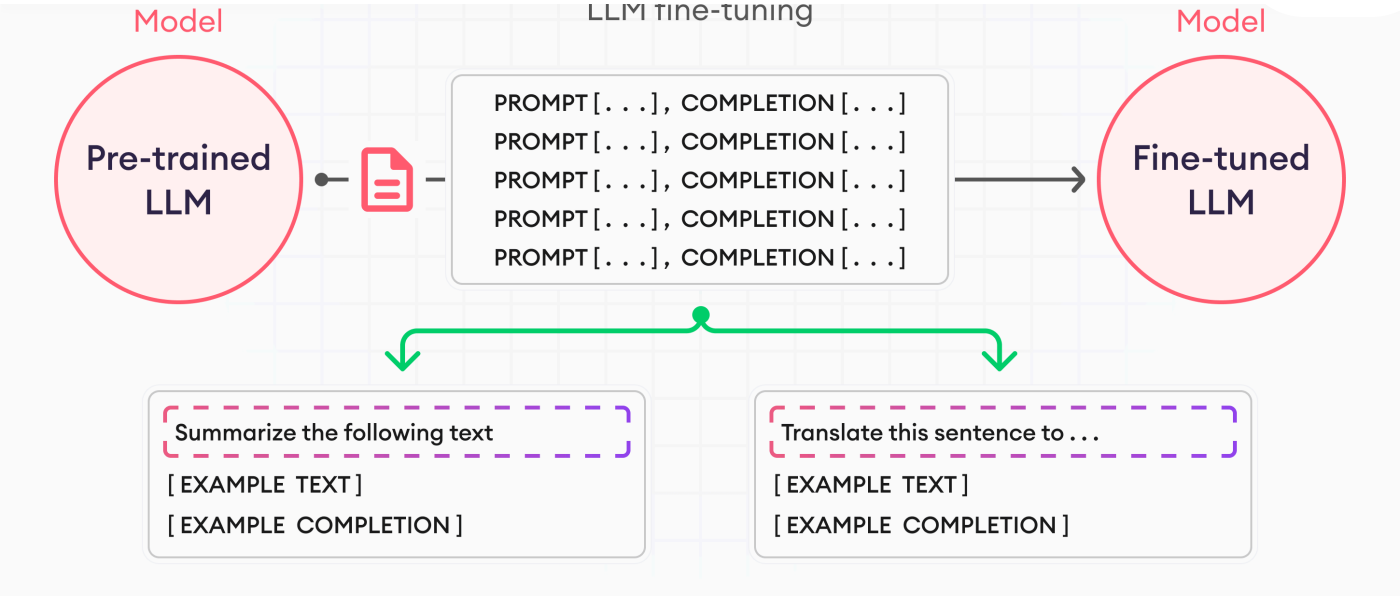
either the prompt or the model will be necessary. If you have access to the data used to train the LLM, you can figure out exactly what prompts were used to imbue it with RAG capabilities, and following that prompting style could lead to performance increases. In most cases, the training data isn't available, and then you can experiment with prompt engineering and try different language models.

If nothing else works, fine-tune

If other methods like prompt engineering don't work, fine-tuning could be the key. Since GPT-4 performs effectively, it's a good strategy to use it to generate answers and then use those answers to fine-tune a smaller model. This can cut down on the amount of data collection needed. If this isn't possible, the only other option might be to use data written by humans for fine-tuning, which can be costly, but sometimes it's the only way to go.

Training data is the backbone of a successful LLM model. The right data can transform a good model into a great one, capable of handling complex tasks with remarkable accuracy. The challenge is that creating this perfect training data isn't easy.

That's where companies like SuperAnnotate step in. Superannotate helps companies fine-tune their LLMs with the highest-quality training data, laying a solid foundation for their AI model to thrive and excel. Feel free to try the LLM & Gen AI playground to explore the ready-made templates for your LLM use cases or build your own use case.



Emerging technique: RAFT

A new technique recently gained attention that combines the strengths of RAG and fine-tuning–retrieval augmented fine-tuning (RAFT).

RAG lets the model search in relevant documents to answer questions; that's why it's called 'retrieval.' Fine-tuning, on the other hand, is about 'feeding' the pre-trained model a smaller, specific dataset to adapt it for a particular task. People often use the phrase 'RAG or fine-tuning' or 'RAG vs. fine-tuning' until researchers discovered that RAG combined with fine-tuning is a much more versatile approach.

RAFT is about training LLMs and making them better at specific topics while improving in-domain RAG performance. This means that RAFT not only ensures the models are fluent in domain-specific knowledge through fine-tuning but also ensures the question-document-answer fit is proper through RAG.

Final remarks

Setting up a RAG system might seem easy, but the real test comes when you need it to excel in a specific task. This can be tricky and often requires a significant investment. The crucial step is pinpointing the exact part of the system that's not