Fine-Tuning a Linear Adapter for Any Embedding Model

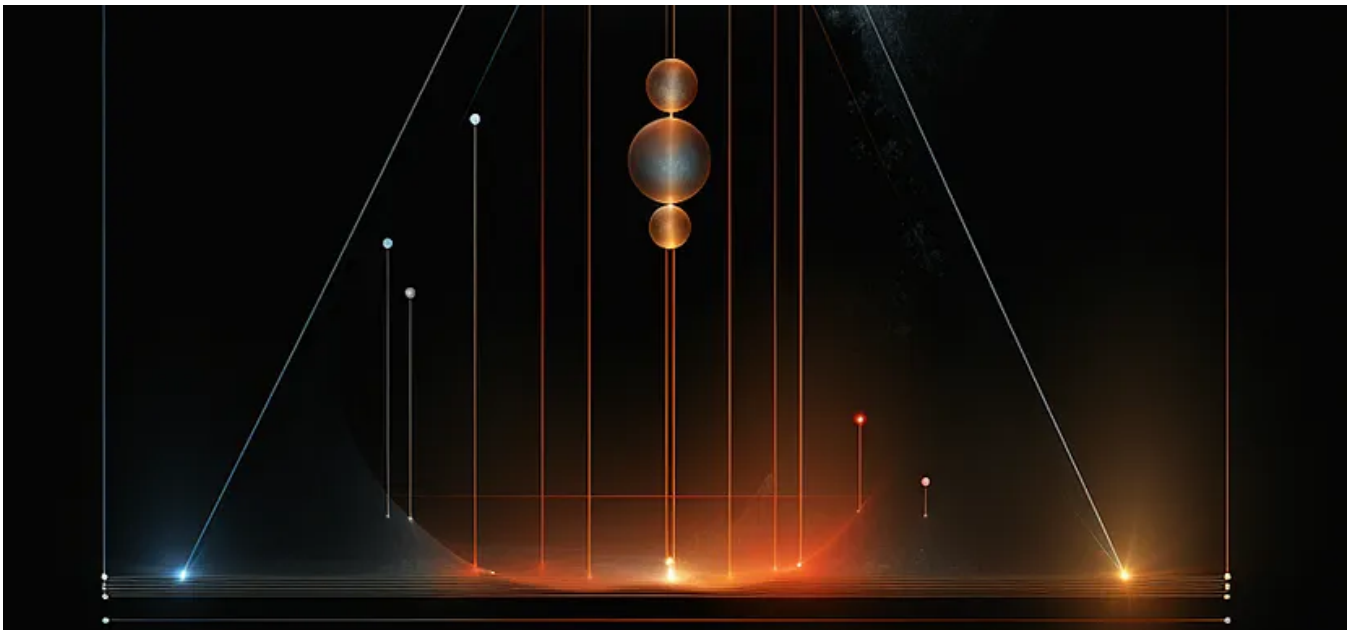
Jerry Liu · Follow
Published in LlamaIndex Blog
6 min read · Sep 6, 2023

We've added capabilities in LlamaIndex allowing you to fine-tune a linear adapter on top of embeddings produced from *any* model (`sentence_transformers`, OpenAI, and more).

This allows you to transform your embedding representations into a new latent space that's optimized for retrieval over your specific data and queries. This can lead to small increases in retrieval performance that in turn translate to better performing RAG systems.

A nice bonus: you do *not* need to re-embed your documents by using this adapter! Simply transform the query instead.

We have a full end-to-end guide showing how you can generate a synthetic dataset, fine-tune the linear adapter, and evaluate its performance.



thanks Midjourney

Context

The concept of fine-tuning your embedding model is powerful. In fact, we were inspired to both add a full example repository / blog post as well as native abstractions in LlamaIndex showing how you can fine-tune a sentence_transformers model over any unstructured text corpus (with our `SentenceTransformersFinetuneEngine`).

However, this approach has some limitations:

- The `SentenceTransformersFinetuneEngine` is limited to fine-tuning `sentence_transformers` models.
- After finetuning the embedding model, you will need to re-embed your document corpus.

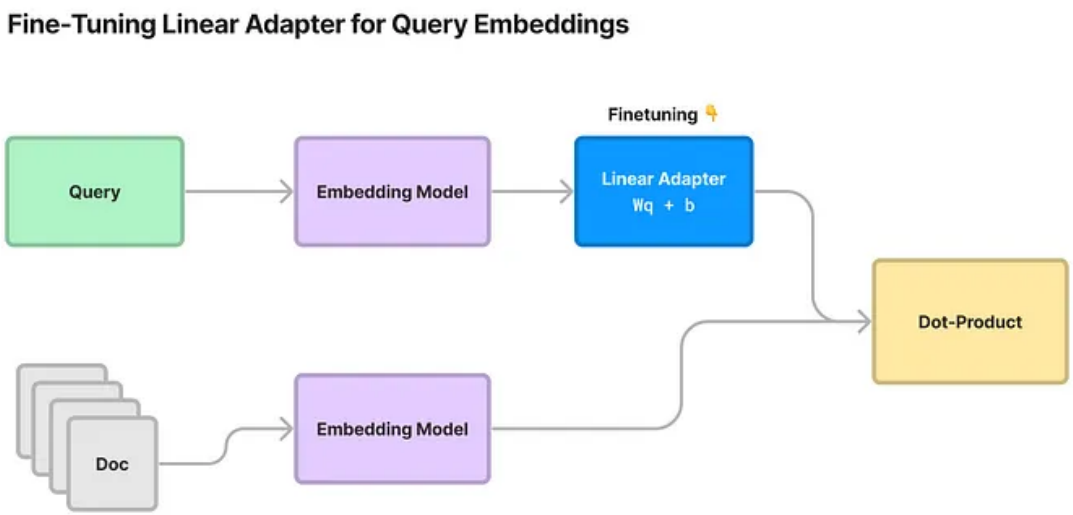
During our Finetuning + RAG webinar last Friday, Jo (Vespa) mentioned the exact same problem: fine-tuning the embeddings model requires you to reindex your documents. However, his work with Vespa explored the concept of "freezing" document embeddings using a foundation model, and instead training a transformation on the query embedding.

This inspired us to explore a similar embedding fine-tuning approach that was simultaneously more general but also allowed us to freeze existing document embeddings.

Approach

Our brand-new `EmbeddingAdapterFinetuneEngine` fine-tunes a **linear adapter** on top of query embeddings produced by any model. The **linear adapter** is simply a linear transformation that specifically transforms the query embedding *while keeping document embeddings fixed*.

The linear adapter can be used on top of any existing embeddings model: SBERT embeddings, OpenAI embeddings, Cohere embeddings, and more. As a result you can just plug this in on top of any embedding model that you're already using!



Since document embeddings are unchanged, this means that you can always fine-tune this linear adapter *after* you've generated embeddings for your documents. You can choose to arbitrarily re-train this adapter on top of changing data distributions, without needing to re-embed all your documents.

Technical Details

As mentioned above, the linear adapter simply performs a linear transformation on top of the query embedding while keeping the Document embeddings fixed (with a weight matrix W + bias term b):

$$\vec{q_t} = W\vec{q} + b$$

And that's it! If document embeddings can be represented as a (n x d) matrix D, where n is number of documents and d is the embedding dimension, then

embedding similarity is just measured by

$$D\vec{q}_t$$

The linear adapter is trained using a similar loss term as the `MultipleNegativesRankingLoss` function in `sentence_transformers` — given a batch of positive (question, context) examples, the function uses cross-entropy loss under the hood to penalize the ground-truth (question, context) pairs for being far apart and swapped pairs for being too close.

Additional Notes: We ended up writing the bulk of this fine-tuning logic in plain PyTorch, but taking heavy inspiration from the `sentence_transformers` source code. We couldn't use sentence_transformers directly since we take in embeddings as inputs rather than raw text. You can take a look at some of our training code here.

Notebook Walkthrough

In this notebook walkthrough, we follow a similar set of steps as our previous blog post on embedding fine-tuning:

1. Generate a synthetic question-context dataset for both training and evaluation.
2. Fine-tuning our linear adapter on top of an existing model (e.g. SBERT)
3. Getting the embedding model, and evaluating it.

As with the previous post, we use the UBER and LYFT 10K as example data. We use Lyft to generate our training dataset and Uber to generate our evaluation dataset.

The full guide is here: https://gpt-index.readthedocs.io/en/latest/examples/finetuning/embeddings/finetune_embedding_adapter.html

Generate a Synthetic Dataset for Trraining and Evaluation

We use our helper abstractions, `generate_qa_embedding_pairs` , to generate our training and evaluation dataset. This function takes in any set of text nodes (chunks) and generates a structured dataset containing (question, context) pairs.

```
from llama_index.finetuning import (
    generate_qa_embedding_pairs,
    EmbeddingQAFinetuneDataset,
)

# generate
train_dataset = generate_qa_embedding_pairs(train_nodes)
val_dataset = generate_qa_embedding_pairs(val_nodes)

# save
train_dataset.save_json("train_dataset.json")
val_dataset.save_json("val_dataset.json")

# load
train_dataset = EmbeddingQAFinetuneDataset.from_json("train_dataset.json")
val_dataset = EmbeddingQAFinetuneDataset.from_json("val_dataset.json")
```

Fine-tuning our Linear Adapter

We then fine-tune our linear adapter on top of an existing embedding model. We import our new `EmbeddingAdapterFinetuneEngine` abstraction, which takes in an existing embedding model and a set of training parameters.

In this example we use the `bge-small-en` sentence-transformers model, but we can also use any embedding model in LlamaIndex/LangChain.

```
from llama_index.finetuning import EmbeddingAdapterFinetuneEngine
from llama_index.embeddings import resolve_embed_model
import torch

base_embed_model = resolve_embed_model("local:BAAI/bge-small-en")
# alternative: use OpenAI
# from llama_index.embeddings import OpenAIEmbedding
# openai = OpenAIEmbedding()

finetune_engine = EmbeddingAdapterFinetuneEngine(
    train_dataset,
    base_embed_model,
    model_output_path="<model_output_path>",
    epochs=4,
    verbose=True,
    # can optionally pass along any parameters that go into `train_model`
    # optimizer_class=torch.optim.SGD,
    # optimizer_params={"lr": 0.01}
)
```

We can then call fine-tune to kick off the fine-tuning job. Training a linear model is quite straightforward and doesn't require heavy machinery — this can easily run on a Macbook.

```
finetune_engine.finetune()
```

Getting the Embedding Model, and Evaluating it

Once the fine-tuning job is then, we can then fetch our embedding model.

We can either directly fetch it from our `finetune_engine` , or import our new `LinearAdapterEmbeddingModel` and construct it in a more manual fashion.

Option 1:

```
embed_model = finetune_engine.get_finetuned_model()
```

Option 2:

```
from llama_index.embeddings import LinearAdapterEmbeddingModel

embed_model = LinearAdapterEmbeddingModel(base_embed_model, "<model_output_path
```

The next step is to evaluate it. We compare the fine-tuned model against the base model, as well as against `text-embedding-ada-002` .

We evaluate with two ranking metrics:

- **Hit-rate metric:** For each (query, context) pair, we retrieve the top-k documents with the query. It's a *hit* if the results contain the ground-truth context.
- **Mean Reciprocal Rank:** A slightly more granular ranking metric that looks at the "reciprocal rank" of the ground-truth context in the top-k retrieved set. The

reciprocal rank is defined as 1/rank. Of course, if the results don't contain the context, then the reciprocal rank is 0.

Some additional comments:

- We ran with 4 epochs over the Lyft documents
- We used Adam as an optimizer with the default learning rate (we tried SGD and it didn't work as well)

Results

	retrievers	hit_rate	mrr
0	ada	0.870886	0.730105
1	bge	0.787342	0.643038
2	ft	0.798734	0.662152

Quantiative metrics (hit-rate and MRR) for ada, bge, and our fine-tuned model

In terms of hit-rate, the base model gets 78.7% hit-rate on the validation dataset, and the fine-tuned model gets 79.8%. In the meantime `text-embedding-ada-002` gets 87.0%.

In terms of MRR, the base model gets 64.3%, and the fine-tuned model gets 66%. `text-embedding-ada-002` gets 68.4%.

There is some performance bump from the fine-tuned model, though admittedly it is small — it is smaller than the performance bump gained through fine-tuning sentence_transformers directly on the latest dataset.

That said, a performance bump is still a performance bump, and it's very cheap for you to spin up and try yourself! So you can decide whether or not this would make sense for you.

Conclusion

We created a brand-new module in LlamaIndex that allows you fine-tune a linear adapter on top of any embedding model.

It can help you eke out some marginal improvement in retrieval metrics; importantly, it allows you to keep document embeddings fixed and only transform

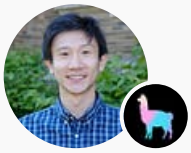
the query.

Resources

Guide: https://gpt-index.readthedocs.io/en/latest/examples/finetuning/embeddings/finetune_embedding_adapter.html

Training code (if you want to take a look for yourself): https://github.com/jerryjliu/llama_index/blob/main/llama_index/finetuning/embeddings/adapter_utils.py

Fine Tuning | Embedding | Llamaindex | NLP | AI



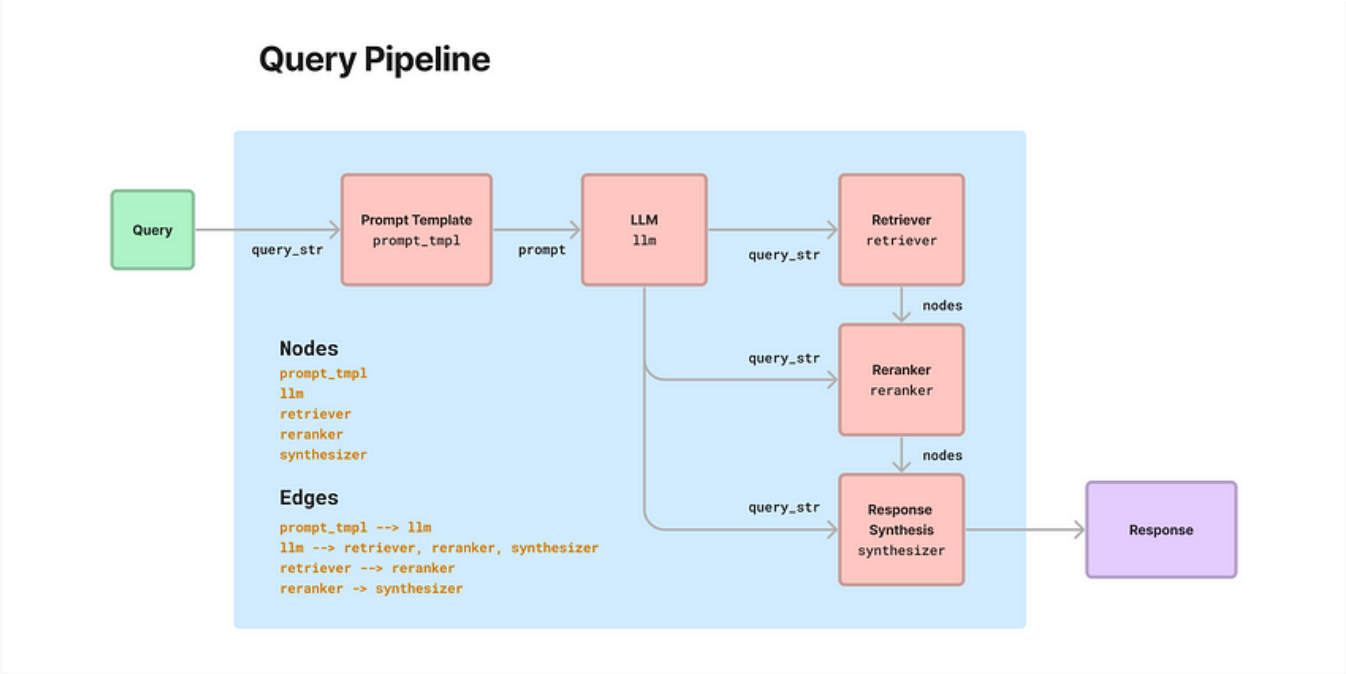
Follow

Written by Jerry Liu

5.7K Followers · Editor for LlamaIndex Blog

Creator of LlamaIndex

More from Jerry Liu and LlamaIndex Blog



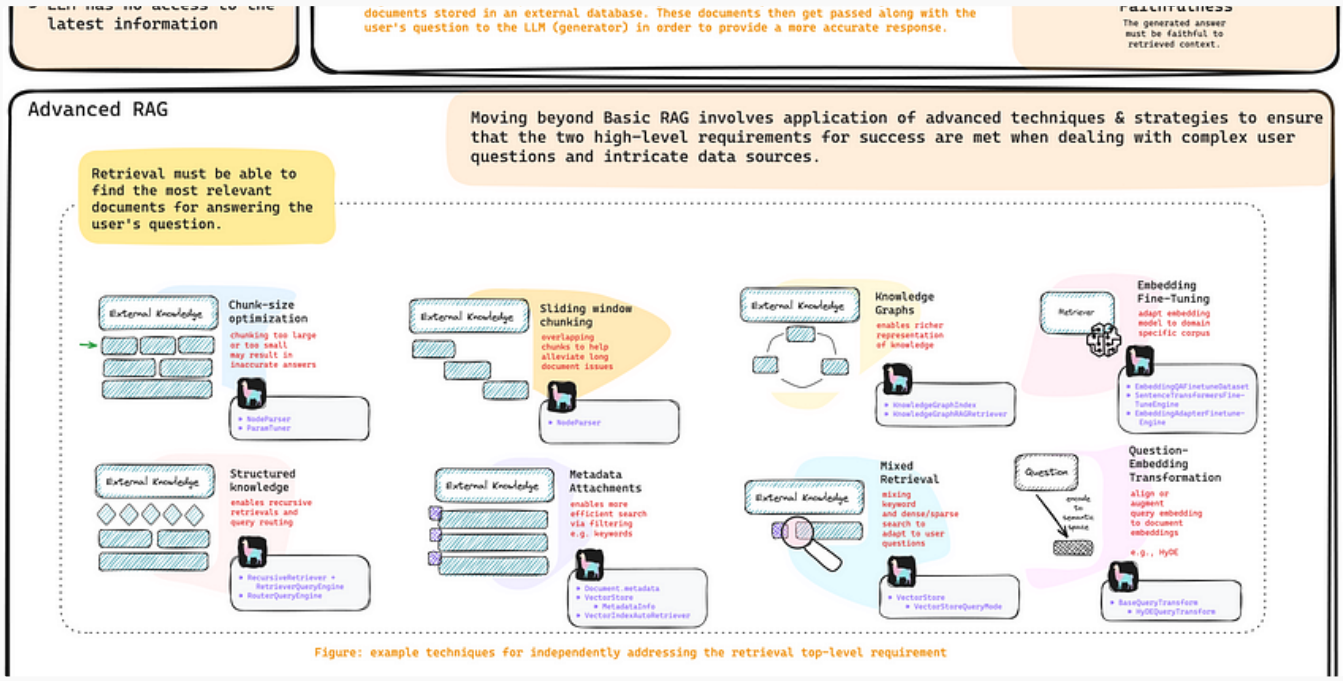
Jerry Liu in LlamaIndex Blog

Introducing Query Pipelines

Today we introduce Query Pipelines, a new declarative API within LlamaIndex that allows you to concisely orchestrate simple-to-advanced...

6 min read · Jan 8, 2024

339 4



Andrei in LlamaIndex Blog

A Cheat Sheet and Some Recipes For Building Advanced RAG

It's the start of a new year and perhaps you're looking to break into the RAG scene by building your very first RAG system. Or, maybe...

7 min read · Jan 5, 2024

1.5K 5

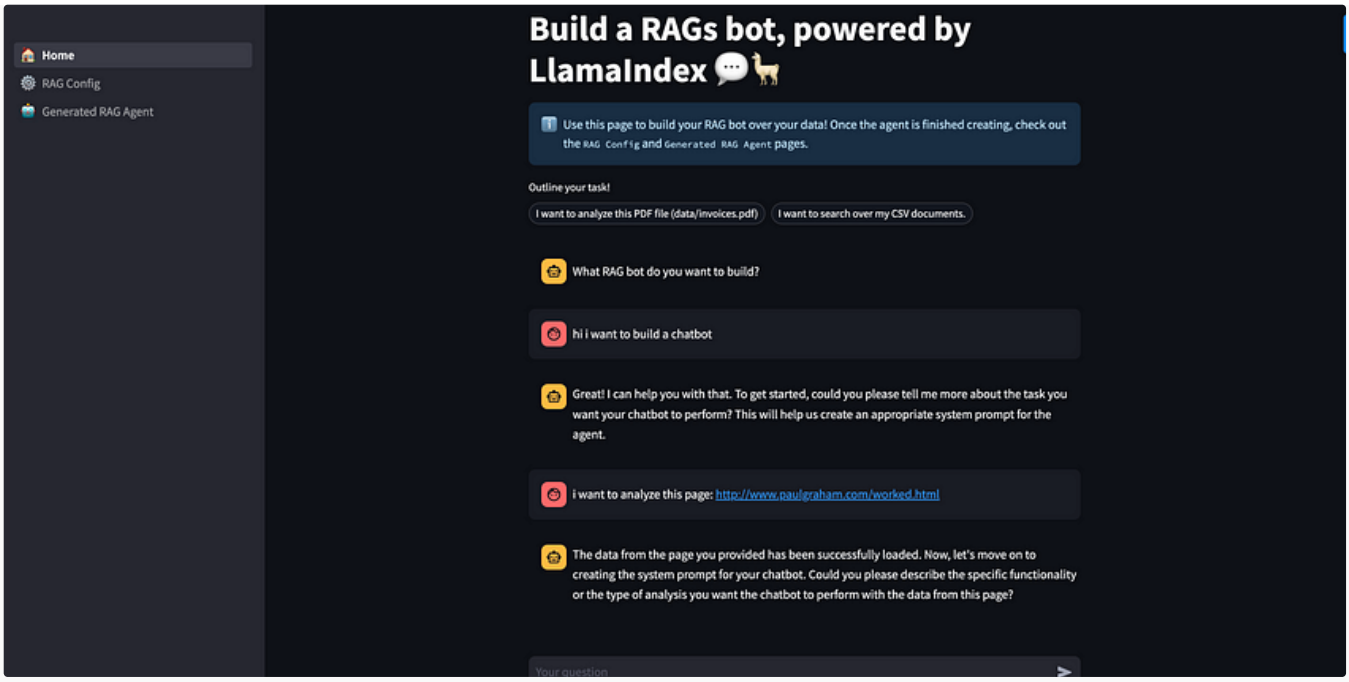
LlamaIndex in LlamaIndex Blog

Running Mixtral 8×7 locally with LlamaIndex and Ollama

You may have heard the fuss about the latest release from European AI powerhouse Mistral AI: it's called Mixtral 8×7b, a "mixture of...

6 min read · Dec 21, 2023

757 8



Jerry Liu in LlamaIndex Blog

Introducing RAGs: Your Personalized ChatGPT Experience Over Your Data

Today we introduce RAGs, a Streamlit app that allows you to create and customize your own RAG pipeline and then use it over your own data —...

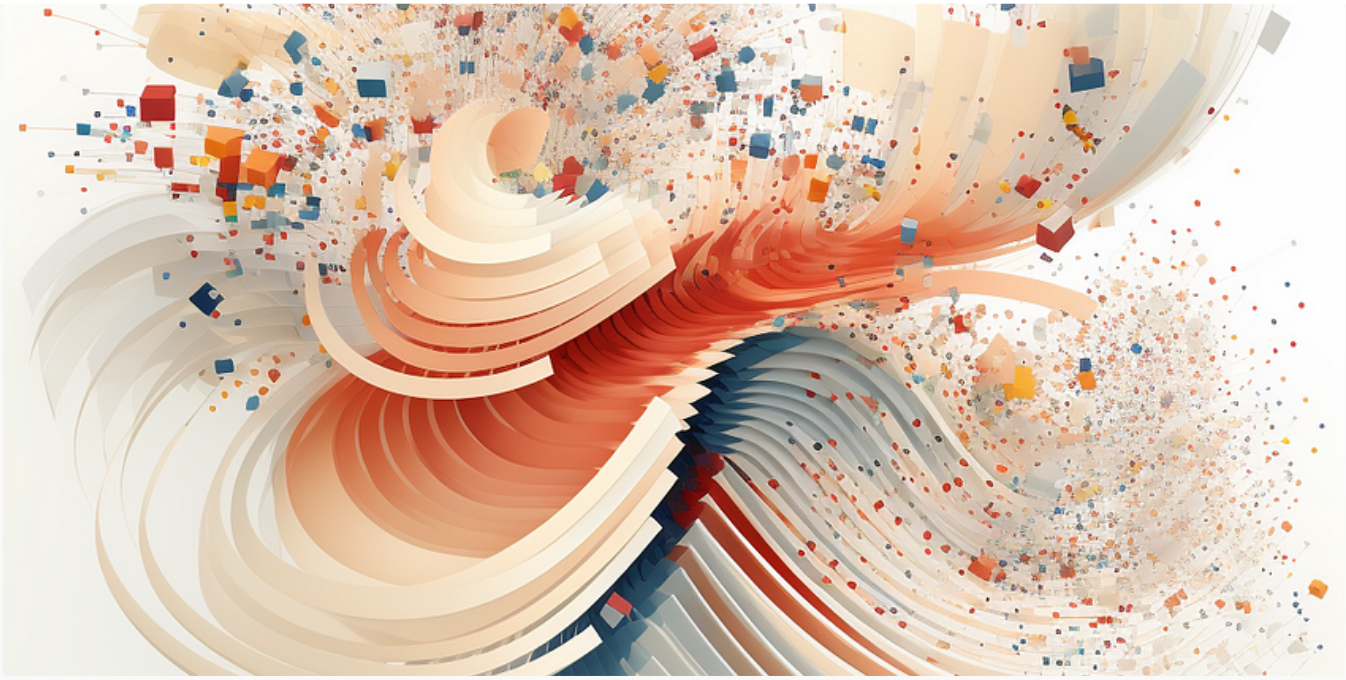
4 min read · Nov 21, 2023

454 4

See all from Jerry Liu

See all from LlamaIndex Blog

Recommended from Medium



Jerry Liu in LlamaIndex Blog

Fine-Tuning Embeddings for RAG with Synthetic Data

UPDATE 9/10/2023: We've included embedding finetuning abstractions into the LlamaIndex repo, so this repo is technically outdated! Please...

6 min read · Aug 25, 2023

630 3

Wenqi Glantz in Better Programming

Fine-Tuning GPT-3.5 RAG Pipeline with GPT-4 Training Data

NVIDIA SEC 10-K filing analysis before and after fine-tuning

· 11 min read · Sep 4, 2023

256 2

Lists

The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 304 saves

Natural Language Processing

1187 stories · 665 saves

Generative AI Recommended Reading

52 stories · 716 saves

What is ChatGPT?

9 stories · 290 saves

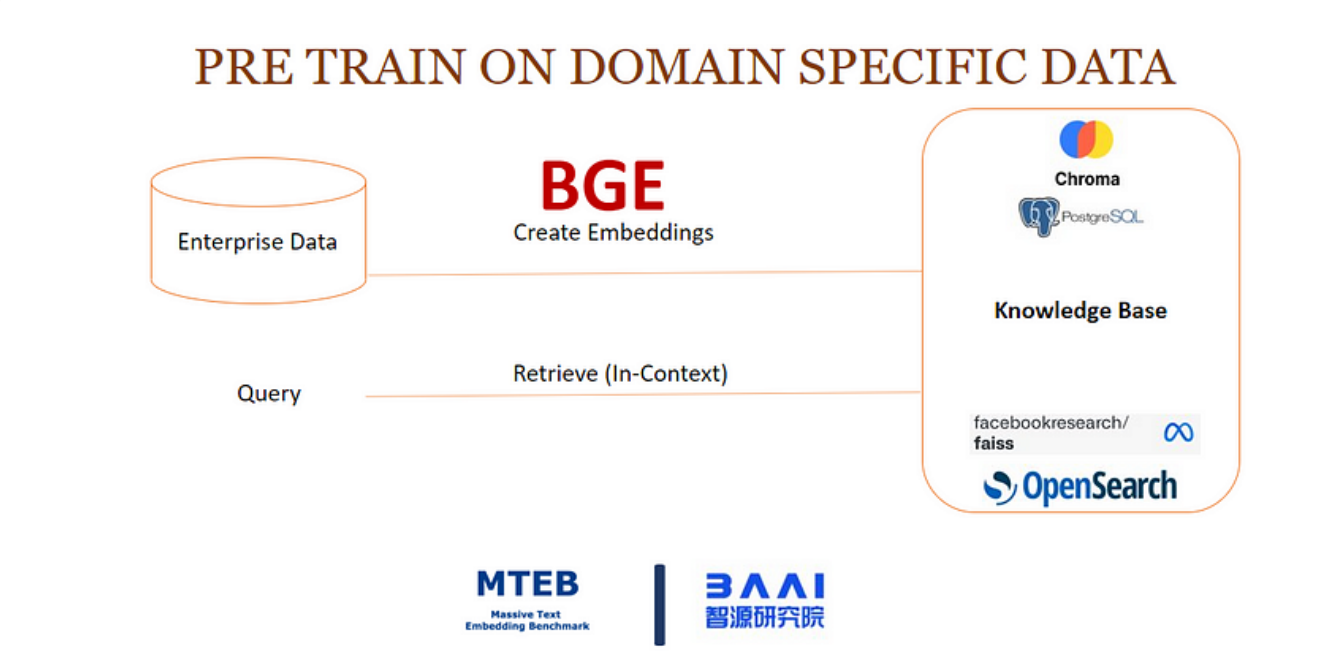
Datadrifters in GoPenAI

BG Embeddings (BGE), Llama v2, LangChain, and Chroma for Retrieval QA

Embeddings play a pivotal role in natural language modeling, particularly in the context of semantic search and retrieval augmented...

· 8 min read · Aug 20, 2023

181 3



Yogendra Sisodia

State-of-the-Art BGE embeddings for retrieval augmented generation

Introduction

2 min read · Aug 20, 2023

9 1

Venkat Ram Rao

Fine Tuning a Sentence Transformer Model

A quick introduction to fine tuning Sentence Transformers for Semantic Search.

· 8 min read · Dec 22, 2023

55 1

Ozgur Guler

How to improve RAG peformance ?—Advanced RAG Patterns—Part2

In the realm of experimental Large Language Models (LLMs), creating a captivating LLM Minimum Viable Product (MVP) is relatively...

12 min read · Oct 18, 2023