Understanding LlamaIndex

- Understanding LlamaIndex

Ram N Sangwan

LLM Fine Tuning?

LLMs are not trained on **your** data, which may be private or specific to the problem you're trying to solve.

It's behind APIs, in SQL databases, or trapped in PDFs and slide decks.

You may choose to **fine-tune** a LLM with your data, but:

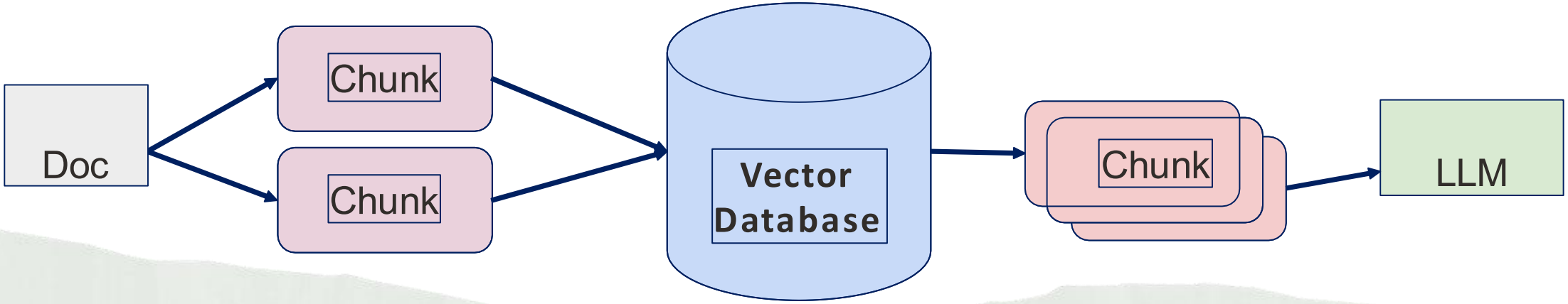
- Training a LLM is **expensive**.
- Due to the cost to train, it's **hard to update** a LLM with latest information.
- **Observability** is lacking. When you ask a LLM a question, it's not obvious how the LLM arrived at its answer.

Challenges with Naive RAG (Response Quality)

- **Bad Retrieval**
  - **Low Precision:** Not all chunks in retrieved set are relevant
    - Hallucination + Lost in the Middle Problems
  - **Low Recall:** Now all relevant chunks are retrieved.
    - Lacks enough context for LLM to synthesize an answer
  - **Outdated information:** The data is redundant or out of date.
- **Bad Response Generation**
  - **Hallucination:** Model makes up an answer that isn't in the context.
  - **Irrelevance:** Model makes up an answer that doesn't answer the question.
  - **Toxicity/Bias:** Model makes up an answer that's harmful/offensive.

How can LlamaIndex help?

- **Data**: Store additional information beyond raw text chunks.
- **Embeddings**: Optimize embedding representations.
- **Retrieval**: Do better than top-k embedding lookup.
- **Synthesis**: Use LLMs for more than generation.



How can LlamaIndex help?

LlamaIndex provides the following tools:

- **Data connectors** ingest existing data from their native source and format. Could be APIs, PDFs, SQL, and (much) more.
- **Data indexes** structure your data in intermediate representations that are easy and performant for LLMs to consume.
- **Engines** provide natural language access to your data. For example:
  - Query engines are powerful retrieval interfaces for knowledge-augmented output.
  - Chat engines are conversational interfaces for multi-message, “back and forth” interactions with your data.
- **Data agents** are LLM-powered knowledge workers augmented by tools.
- **Application integrations** Tie LlamaIndex into your ecosystem. This could be LangChain, Flask, Docker, ChatGPT, or… anything else!

LlamaIndex with RAG?

When to use LlamaIndex with RAG.

Large Graphs

When working with large graphs representing extensive knowledge bases, LlamaIndex can help in efficiently selecting a subset of relevant nodes for retrieval.
This can improve the scalability and speed of the RAG model.

Diverse Retrieval

LlamaIndex is particularly useful when you want diverse retrieval results.

By selecting nodes with diverse content representations, LlamaIndex can help ensure that the retrieved information covers a wide range of topics or perspectives.

Resource Efficiency

LlamaIndex allows for more resource-efficient retrieval by focusing on a smaller subset of nodes in the graph.
This can be beneficial in scenarios where computational resources are limited or where quick response times are required.

LlamaIndex with RAG?

LlamaIndex with RAG might not be appropriate for:

Specific Retrieval Needs

If your task requires highly specific information that may not be well-represented in the subset selected by LlamaIndex, using it might not yield optimal results.
In such cases, other retrieval methods or direct graph traversal techniques may be more suitable.

Small Graphs

For small graphs where the overhead of applying LlamaIndex is unnecessary or where diversity in retrieval results is not a priority, using LlamaIndex might be overkill.

Simple retrieval methods or even brute-force approaches could suffice in such scenarios.

Performance Trade-offs

While LlamaIndex can improve efficiency and diversity, it also introduces computational overhead for index construction and maintenance.

If the overhead outweighs the benefits, or if real-time performance is critical, alternative approaches may be preferred.

Embeddings in LlamaIndex

Embeddings in LlamaIndex

Embeddings are used in LlamaIndex to represent your documents

- Embedding models take text as input, and return a long list of numbers used to capture the semantics of the text.
- These embedding models have been trained to represent text this way, and help enable many applications, including search!
- At a high level, if a user asks a question about dogs, then the embedding for that question will be highly similar to text that talks about dogs.
- When calculating the similarity between embeddings, there are many methods to use (dot product, cosine similarity, etc.).

Cohere Chat with LlamaIndex

To use Cohere's chat functionality with LlamaIndex create a Cohere model object and call the chat function.

```
from llama_index.llms.cohere import Cohere
from llama_index.core.llms.types import ChatMessage


cohere_model = Cohere(api_key="{API_KEY}")
message = ChatMessage(role="user",content= "Who founded Cohere?")
resp = cohere_model.chat([message])
print(resp)
```

Cohere Embeddings with LlamaIndex

- To use Cohere's embeddings with LlamaIndex create a Cohere Embeddings object with an embedding model from this list and call get_text_embedding.

```
from llama_index.embeddings.cohereai import CohereEmbedding

embed_model = CohereEmbedding(
    cohere_api_key="{API_KEY}",
    model_name="embed-english-v3.0", # Supports all Cohere embed models
    input_type="search_query", # Required for v3 models
)

# Generate Embeddings
embeddings = embed_model.get_text_embedding("Welcome to Cohere!")

# Print embeddings
print(len(embeddings))
print(embeddings[:5])
```

Thank You