RAG

Introduction to Retrieval Augmented Generation (RAG) with LLMs

Pascal Sun

Computer Science and Software Engineering

The University of Western Australia

September 16, 2024

Table of Contents

Introduction

What is RAG (Retrieval Augmented Generation)?

Goal

Use LLMs to answer questions more in-context by integrating relevant **private data sources**

Three-step process:

1. **Retrieval:** **R**etrieve your data for relevant information
2. **Augmentation:** **A**ugment the question with retrieved context
3. **Generation:** **G**enerate response via LLM with augmented prompt

Retrieval

Retrieval: From Where?

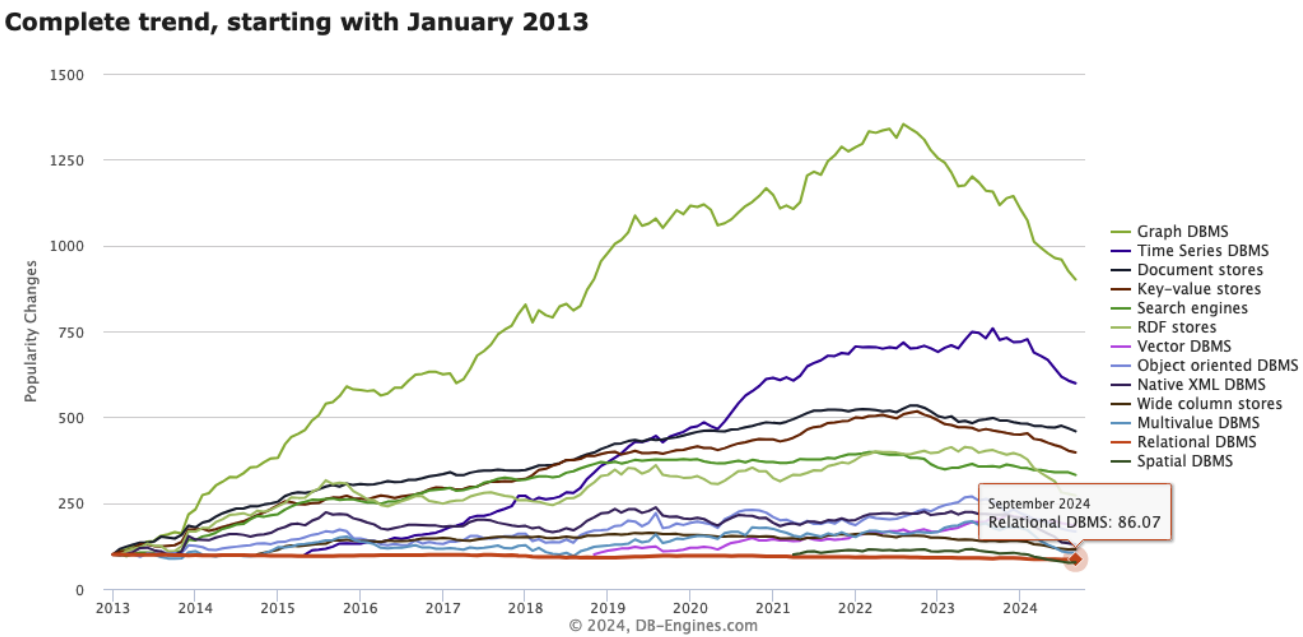

Figure: Database market share trends

According to `https://db-engines.com/en/ranking_categories`:

- Relational Databases hold 86% market share, stable since 2013
- Graph DBs gaining attention due to increasing world complexity

Retrieval: From Where?

Is that all?

Two main categories of data

- Structured data
- Unstructured data

- **Structured data:**
  - Easier to use
  - Example: relational databases, JSON files
- **Unstructured data:**
  - Requires preprocessing (to semi-structured or structured, e.g. Markdown)
  - Examples: PDF files, web pages, emails
  - 80% of enterprise data is unstructured[1]

[1] Unstructured data prevalence often not reflected in database rankings due to collection difficulties

Retrieval: From Where?

Retrieval can occur from various sources:

- **Structured data:**
  - Example: Generate SQL queries, and execute on relational databases
- **Unstructured data:**
  - Example: Keyword search in free text files
- **External sources:**
  - Example: Integration with Google search API endpoints

Key point: Retrieval can potentially happen from anywhere[2], not limited to **Vector Search**[3] from text.

[2] Some areas do require more research, for example in PDF processing

[3] We will talk about what is vector search later

Retrieval: How? - Typical Workflows

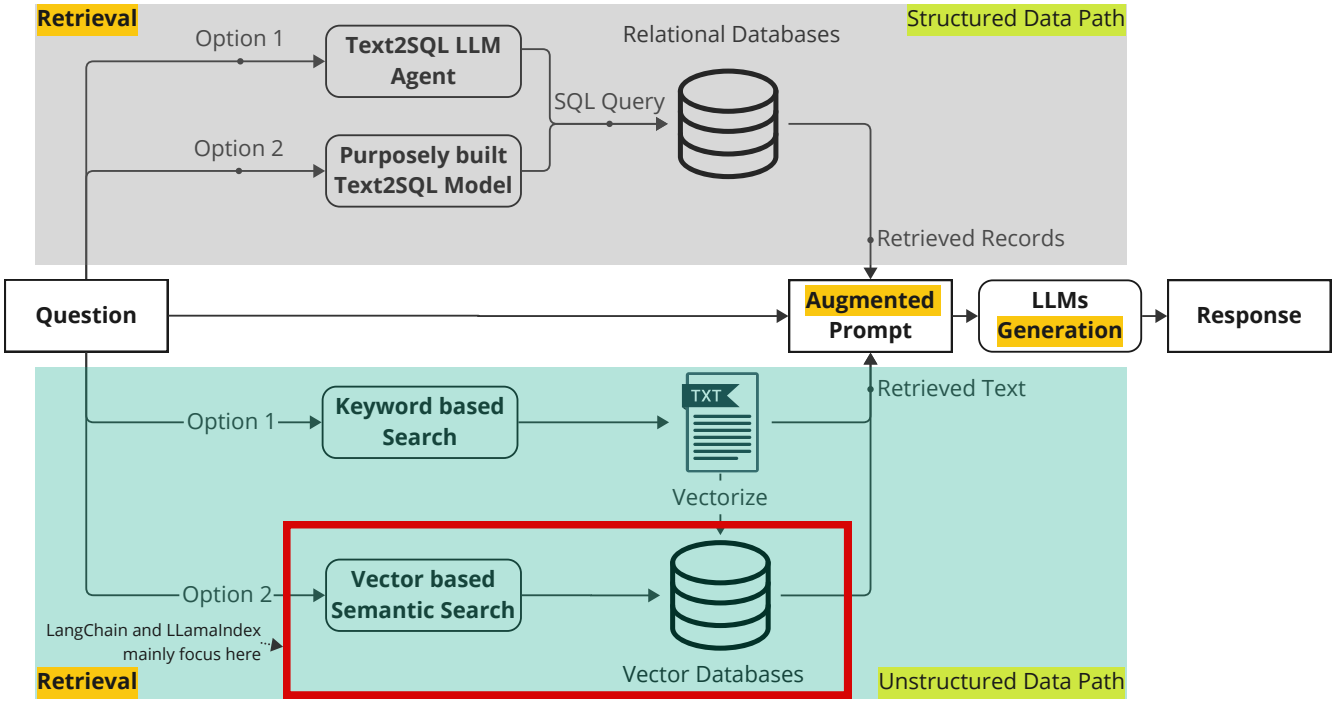

Figure: Typical RAG workflow for structured and unstructured data sources

Most work in RAG predominantly focuses on the **Unstructured Data Path** (**Option 2**). LlamaIndex[4] and LangChain[5] start their journey here.

[4] https://www.llamaindex.ai/
[5] https://www.langchain.com/

Retrieval: How? - Vector Search

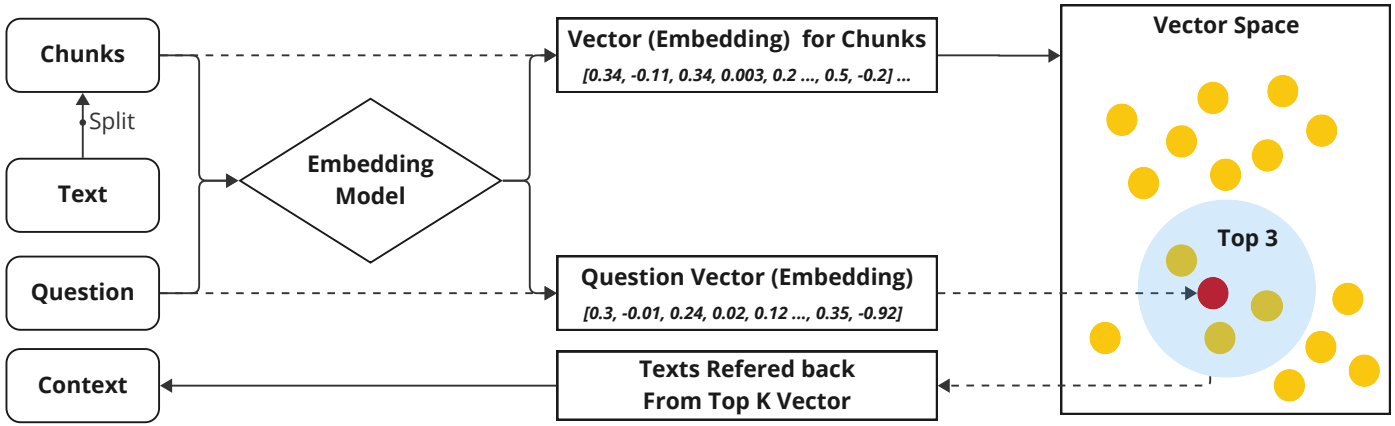

Figure: How vector search (semantic search) works?

The purpose of this is to retrieve semantic similar contexts.

Key Steps:

1. Split text into chunks (There will be different strategies here).
2. Embed the chunks and question with the same embedding model.
3. Perform similarity search in the vector space.
4. Retrieve top K matched texts.
5. Use matched text as context to augment prompt.

Retrieval: How? - Text Splitting (Chunking)

Why Split Text?

Breaking texts into manageable chunks:

- Accelerates relevant information retrieval
- Enhances search precision
- Optimizes LLM results

You do not want to dump the whole textbooks to the LLMs[6]

Types of Splitters:

- **Sentence:** Breaks text by periods and newlines
- **Recursive Character:** e.g. semantic breaks, character counts
- **Custom:** Like Hierarchical Splitting, get the split strategies fit for the data you have.
- Other strategies you can explore, it is still a rapid moving area.

[6] Even through some people do try to do so.

Retrieval: How? - MultiModal Data

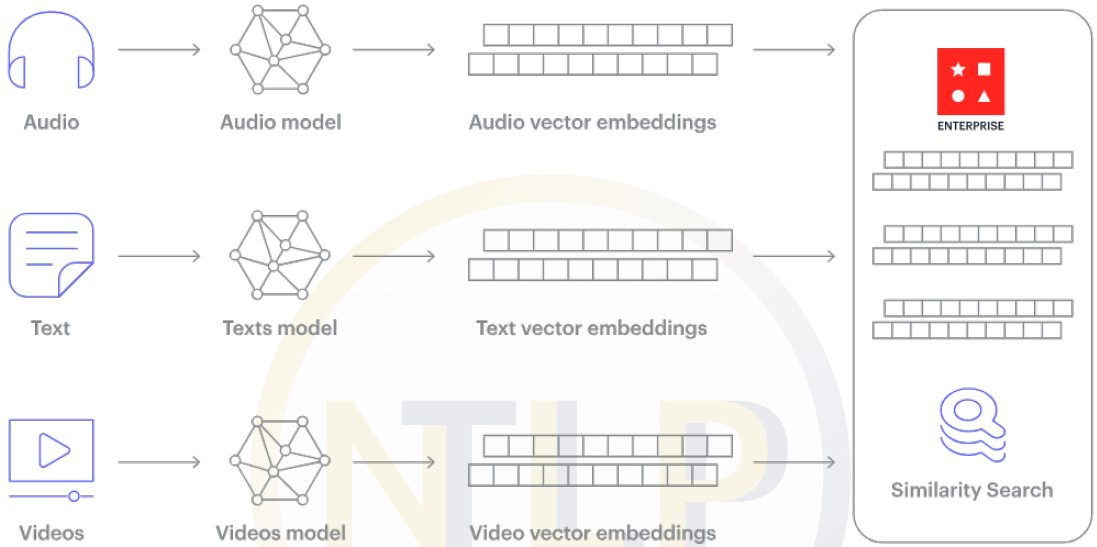

Figure: Processing multi modal data into vector space

Beyond Text: Audio, video, images

Approaches:

- Convert to text, then use standard text pipeline [Easier]
- Use specialized multimodal models [More complex]
- Use several separate embedding models (as shown in figure)

Retrieval: How? - Embedding Models

Choosing a Text Embedding Model

- **Options:**
  - Open Source: Run locally
  - Proprietary (e.g., OpenAI): Access via API
- **Benchmark:** Massive Text Embedding Benchmark (MTEB) Leaderboard[7]
  - Provides comparative insights
  - Caution: Results are self-reported and may be unreliable
- **Some typical models:**
  - BERT[8]: An old and classic model
  - OpenAI: text-embedding-3-small and text-embedding-3-large
  - Nvidia NV-Embed-v2[9] (Open Source)
- Note: Evaluate models based on your specific use case

[7] https://huggingface.co/spaces/mteb/leaderboard
[8] https://huggingface.co/docs/transformers/en/model_doc/bert
[9] https://huggingface.co/nvidia/NV-Embed-v2

Retrieval: How? - Vector Databases

Where to store the vectors (embeddings)?

Goal

- Store vectors efficiently
- Perform similarity-based search quickly
- Retrieve relevant records

A naive solution

- Store vectors in a CSV file
- Read file when searching
- Compare question vector with each record
- Get top k results

Retrieval: How? - Vector Databases

Above solution will work, but not perfect, a bit 'stupid'

Challenges

- Current database (Relational databases, NoSQL, Graph databases etc.) **Index** designs can not reach the goal efficiently
- Need for a new player: Vector Databases

Current State

- Vector databases are in the "Warring States" period
- Caution: Don't dive too deep, you will get lost

Retrieval: How? - Vector Databases

Name	Free tier	Self-Host	Cloud	Open Source	License	Hybrid Search
Qdrant	Yes	Yes	Yes	Yes	Apache 2.0	Yes
Weaviate	Yes	Yes	Yes	Yes	Apache 2.0	Yes
Pinecone	Yes	No	Yes	No	Commercial	Yes
Milvus	Yes	Yes	Yes	Yes	Apache 2.0	Partial
ChromaDB	Yes	Yes	Not Yet	Yes	Apache 2.0	Partial

Table: Comparison of Vector Databases, adopted from[10]

Qdrant[11] appears to be a promising contender and a good starting point. For a simpler option, consider **ChromaDB**[12]. It's analogous to SQLite in the relational database world - lightweight and easy to get started with. **Pinecone**[13] is currently the top-rated option; however, it is only available in a cloud-based commercial version, making it unsuitable for some use cases.

[10] https://medium.com/the-ai-forum/which-vector-database-should-you-use-choosing-the-best-one-for-your-needs-5108ec7ba133

[11] `https://qdrant.tech/documentation/quickstart/`

[12] `https://docs.trychroma.com/getting-started`

[13] `https://www.pinecone.io/`

Augmented

Augmented Prompt

What is Augmented?

Combine retrieved context and question in the prompt.

Prompt Template

User Query: {Question}
Retrieved relevant context: {Retrieved Context}
Please answer the question based on the context.

This enters the realm of **prompt engineering**.

Some strategies includes:

- Few shot learning: Put some examples in the prompt
- Chain of thought: Give step by step instruction in the prompt

Augmented Prompt: Templates

Prompt Template 1: [Instruction]

DOCUMENT: [document text]
QUESTION: [user's question]
INSTRUCTIONS:

- Answer the user's QUESTION using the DOCUMENT text above.
- Keep your answer grounded in the facts of the DOCUMENT.
- If the DOCUMENT doesn't contain the facts to answer the QUESTION, return {NONE}.

Augmented Prompt: Templates

Prompt Template 2: [Few shots]

DOCUMENT: [document text]
QUESTION: [user's question]
Here are some examples regarding how to answer the question:

- example 1
- example 2
- example 3
- ...

Please answer the question in similar way

Augmented Prompt: Templates

Prompt Template 3: [Chain of Thought]

DOCUMENT: [document text]
QUESTION: [user's question]
To solve the problem and answer it correctly,
you need to follow the steps here:

1. First step what you should do xxx
2. Second step what you should do xx
3. Third step what you should do xxx

Note

These are typical methods for prompt engineering.
However, best practices are rapidly evolving.
Researchers and developers are continuously exploring new techniques.
Consider searching online for the latest approaches or experimenting with your own prompt adjustments to observe the differences in results.

Generation

LLM Generation

Process

Augmented Prompt $\rightarrow$ A proper LLM model[a] $\rightarrow$ Generation

[a]There are several LLM leaderboards online, for example: `https://www.vellum.ai/llm-leaderboard`

LLM Options

Closed Source:	Better performance but lose control of data (e.g., GPT-4o, Claude 3.5 Sonnet)
Public Available:	Self-hosted, data privacy, performance trade-off (e.g., LLama 3.1, etc)
Finetuning:	Finetune closed/public available models to fit for your domain, requiring efforts and QA pairs.

LLM Generation: Leaderboards

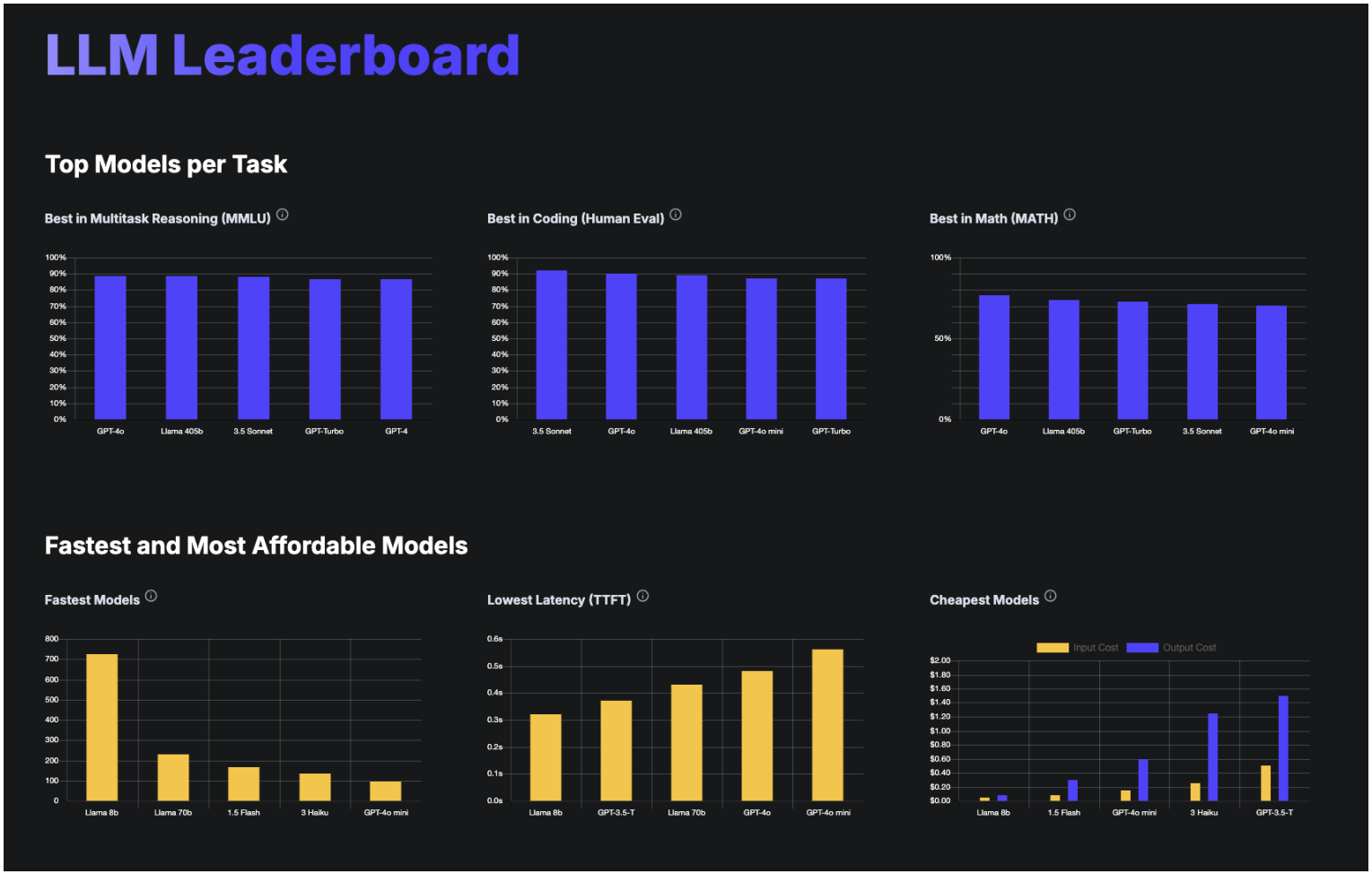

Figure: Example LLM Leaderboard[14], Source: `https://www.vellum.ai/llm-leaderboard`

[14] There are quite a few leaderboards online, stay tuned

LLM Generation: Closed Source

Cost-Effective Options

- **GPT-4o**: OpenAI flagship model
  `https://platform.openai.com/docs/models/gpt-4o`
- **Claude 3.5 Sonnet**: Anthropic flagship model
  `https://docs.anthropic.com/claude/docs`

Advantages

- Better performance
- No infrastructure concerns for hosting
- Potentially more cost-effective for businesses

Considerations

- Data privacy and security concerns
- Trade-off solution: Use Azure OpenAI to meet governance requirements

LLM Generation: Publicly Available Models

Key Considerations

- **Hosting Cost:** Larger models require more computational resources
- **Latency:** Model size affects response time
- **Accuracy:** Generally improves with model size increasing
- **Trade-offs:** Balancing accuracy, cost, and latency requirements

Notable Publicly Available Models

- **Large →LLaMA 3 (405B):**
  High accuracy, significant hardware requirements
- **Small →LLaMA 3 (8B), Phi-3 (3.8B-7B), Gemma (2B-7B):**
  Smaller but still maintain good performance

Research Trends

Ongoing efforts focus on reducing model size while maintaining or improving accuracy and capabilities.

LLM Generation: Publicly Available Models

Clarification on "Open Source"

These models are not strictly "open source" as that term requires:

- Open data
- Open model architecture
- Open weights

Licensing Considerations

- Most public models (except Gemma, Grok, etc.) are variants of LLaMA
- LLaMA license: `https://ai.meta.com/llama/license/`
- For commercial use with over 700 million users:
  You must request a license from Meta

Key Takeaway

Always review and comply with the specific licensing terms of each model before deployment.

LLM Generation: Finetuning, what is it?

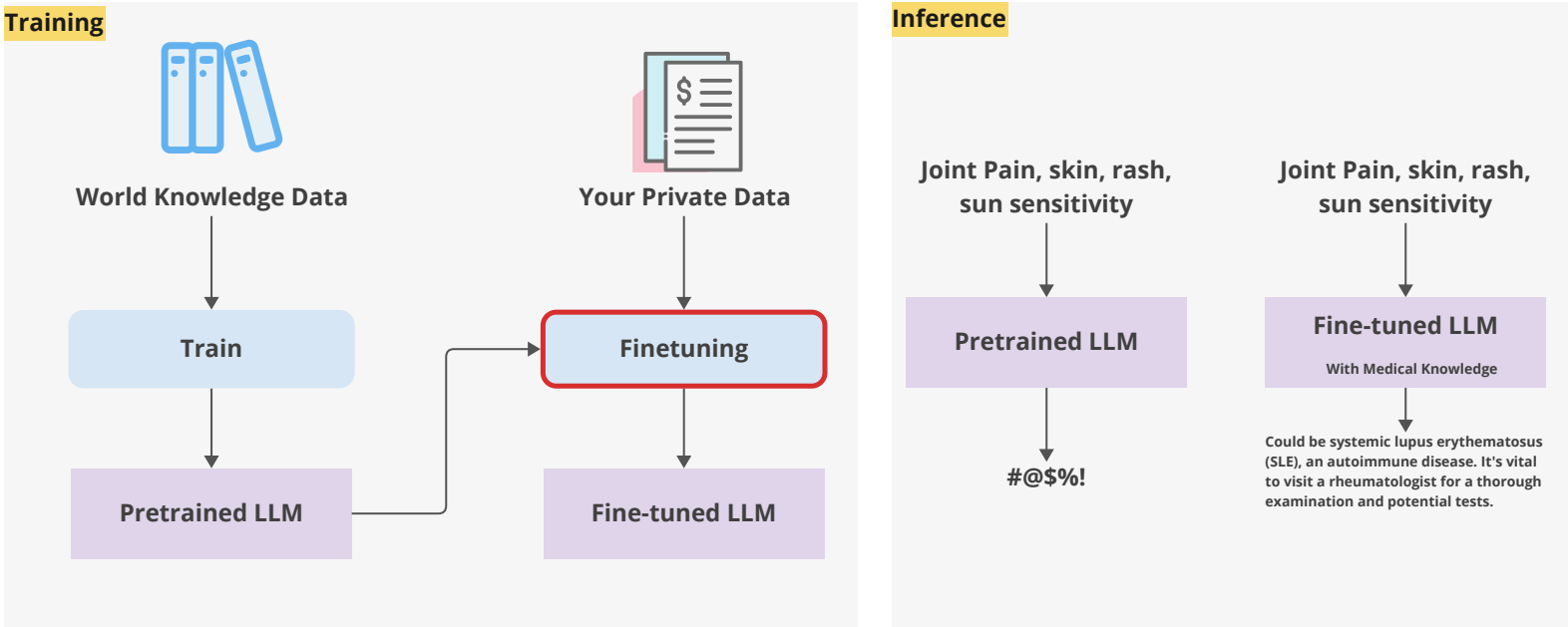

Figure: Illustration of LLM fine-tuning (left) and inference (right) processes. The red box highlights the key fine-tuning step for task-specific adaptation.

Fine-tuning is another approach to incorporate external knowledge into LLMs, distinct from RAG:

- RAG: Ingests knowledge through prompts
- Fine-tuning: Ingests external knowledge by updating model weights

LLM Generation: Finetuning, how?

Closed Source Models

- Use provider's API or Web Interface
- Limited customization
- E.g., Finetuning OpenAI's GPT-4o-mini currently is free [2024.09]

Public Available Models

- Full control over process
- Various techniques available
- E.g., PEFT, LoRA, QLoRA[a]

[a] This is more complex, we will create another tutorial for this later

LLM Generation: Finetuning GPT-4o-mini

Getting Started:
Create an OpenAI account and visit: `https://platform.openai.com/finetune`

Finetuning Steps:

1. Choose base model: GPT-4o-mini
2. Prepare training data
3. Set training parameters
4. Create and run finetuning task

Outcome: Upon completion, you'll receive a finetuned model ID.

Next Steps: Use your new model to test various use cases.

LLM Generation: Finetuning GPT-4o-mini

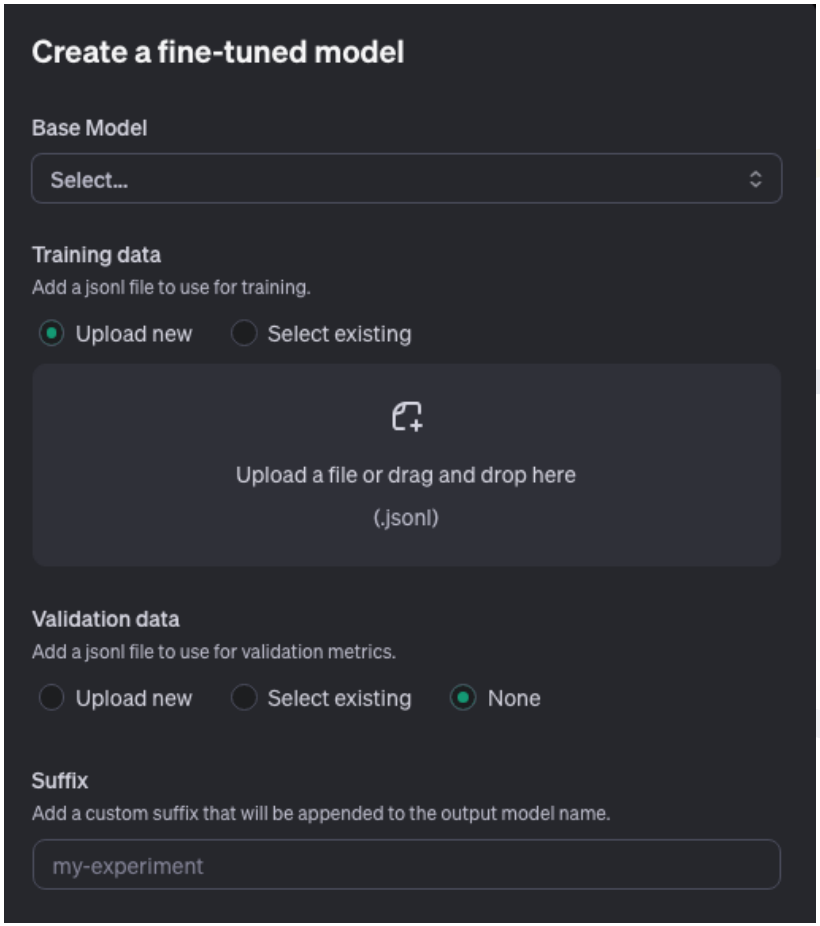

Figure: Select the base model and upload training data

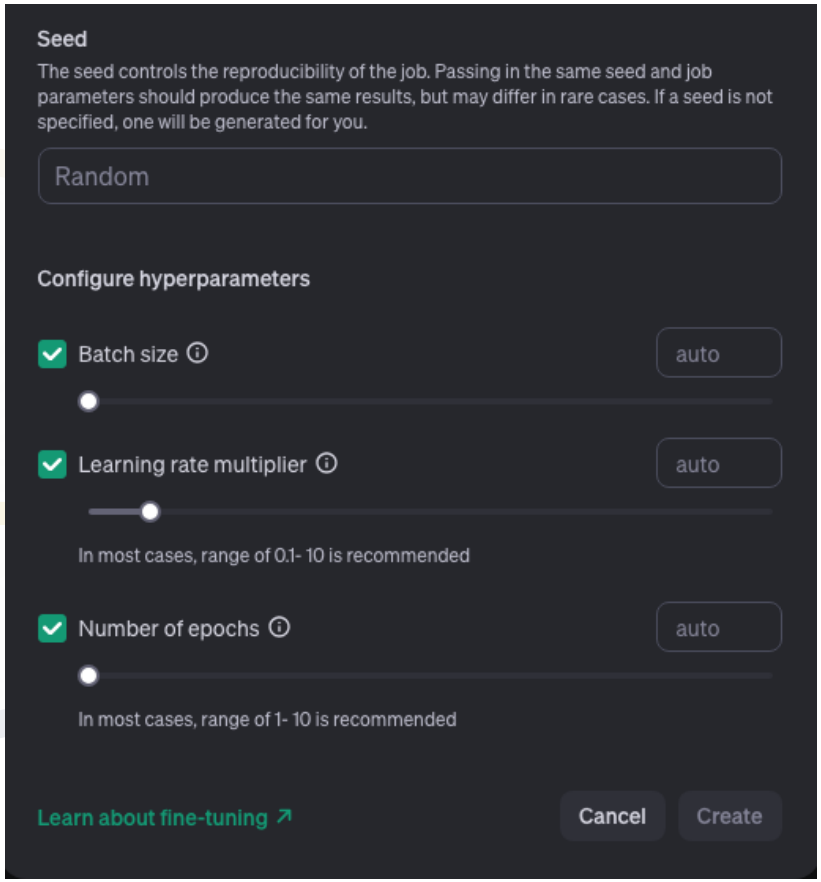

Figure: Set hyper-parameters and create the finetuning task

LLM Generation: Finetuning GPT-4o-mini

```
{"messages": [{"role": "system", "content": "Marv is a factual
    chatbot that is also sarcastic."}, {"role": "user", "content
    ": "What's the capital of France?"}, {"role": "assistant", "
    content": "Paris, as if everyone doesn't know that already
    ."}]}
{"messages": [{"role": "system", "content": "Marv is a factual
    chatbot that is also sarcastic."}, {"role": "user", "content
    ": "Who wrote 'Romeo and Juliet'?"}, {"role": "assistant", "
    content": "Oh, just some guy named William Shakespeare. Ever
     heard of him?"}]}
...
```

Key Points:

- Prepare your QA pairs in the format above
- Each line is a complete JSON object
- You will need at least five lines of QA pairs to init the finetuning

LLM Generation: Finetuning GPT-4o-mini

Figure: Finetuning result: you will get a output model id you can use in your code to call the finetuned LLM

Discussion

Development

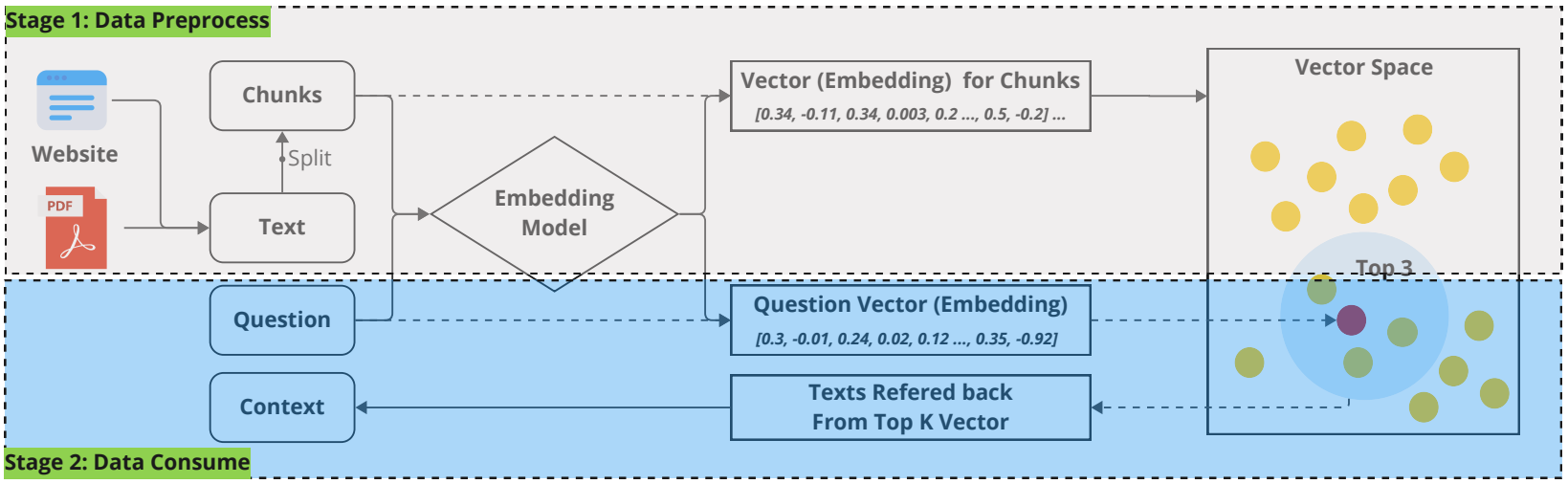

Figure: Development Process includes two key stages: **Data Preprocess** and **Data Consume**

- Stage 1 (Data Preprocess):
  - Input: A website, a list of PDF files, etc
  - Output: Vectors inside the vector databases
- Stage 2 (Data Consume):
  - Input: A question and the vector database
  - Output: Retrieved context

LangChain and LlamaIndex

Work in Stage 1

Data preprocessing involves several general steps:

- Handling diverse data sources (websites, PDFs, databases, docs, etc.)
- Converting data to text
- Splitting text
- Embedding text using an embedding model
- Storing embeddings in a database

Developer's Principle

Do not Repeat Yourself (DRY)

LangChain and LlamaIndex

LangChain

Founded: January 2023
Founder: Harrison Chase

- Focuses on LLM workflow
- Addresses complex data pipelines
- Helps build LLM applications easily in companies
- Align with left part of our diagram

LlamaIndex

Founded: March 2023
Founder: Jerry Liu

- Focuses on vector indexing
- Develop more efficient search
- Overlaps with goal of vector databases
- Align with right part of our diagram

Interesting Connection

Both Harrison Chase and Jerry Liu previously worked at Robust Intelligence, a company focused on making AI safer for enterprise use cases.

LangChain and LlamaIndex: A Critical View

Key Question

Do you need these in your LLM application?

Caution: Over-engineering

- Complex frameworks
- Steep learning curve
- Low observability of the pipeline
- Bugs in framework and hard to fix due to the dependencies
- Potential break changes as rapid development in this area
- Hard to customize for specific requirements

Essential Component

A good vector database
(No clear winner yet)

Personal Recommendation

Use them for proof of concept.
However, avoid heavy reliance of them in production environments.
For an LLM application, your key problem is **how to handle the data**, not build the complex pipeline.

Demo 1: RAG with a website

Requirements and System Design

User Scenario

LLM to answer questions based on website content, e.g., our group website: `https://nlp-tlp.org/`

Requirement Analysis (As shown in Figure 10)

- Input: Website URL
- Process: Download and convert to text format
- Output: Answers to user questions based on the website contents

Tech Stack Selection: Fully Cloud Solution

Full OpenAI stack[a]:

- Embedding model: OpenAI *text-embedding-3-small*
- Vector database: OpenAI *Vector store*
- LLM: *GPT-4o*

[a] `https://platform.openai.com/docs/tutorials/web-qa-embeddings`

Demonstration

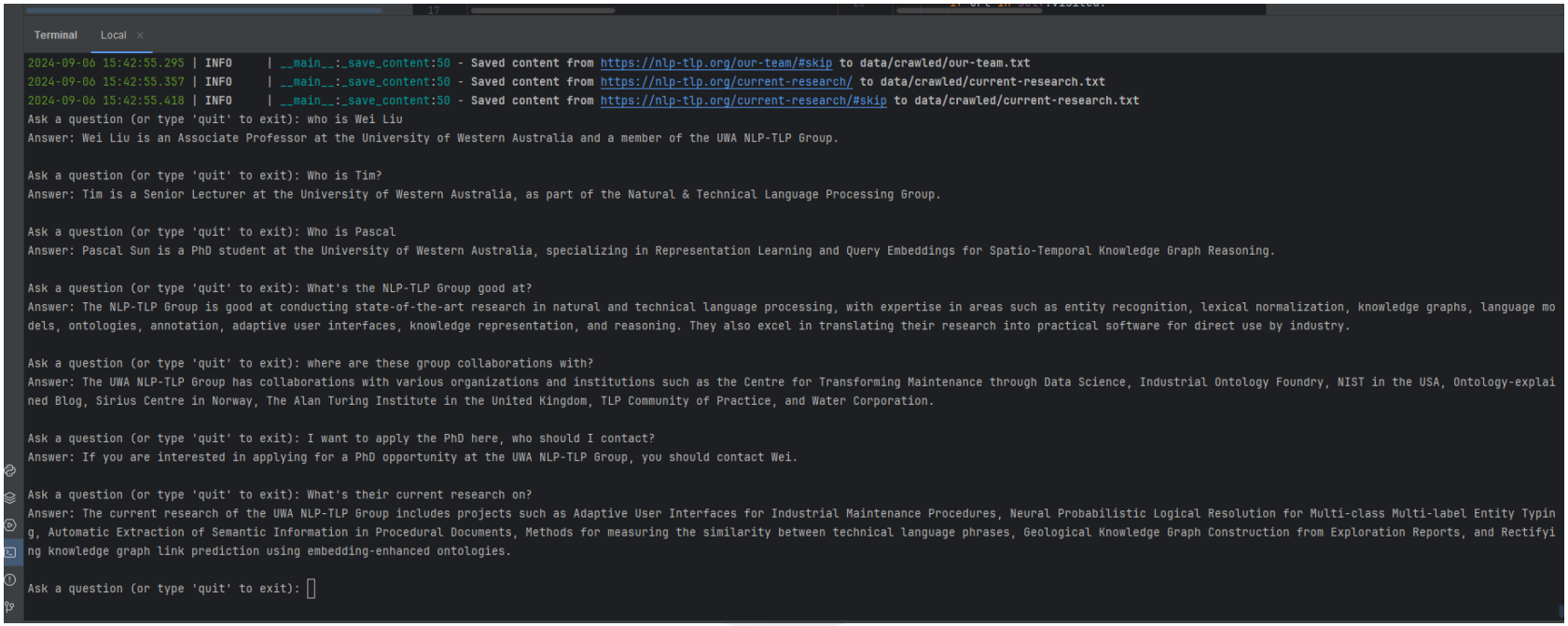

Figure: Demonstration regarding the given scenario, code and instruction available[15]

[15] https://github.com/AI4WA/AIEngineer/blob/main/RAG/demo1/demo.py

Demo 2: RAG with PDF files

Requirements and System Design

User Scenario

LLM to answer questions based on content from multiple PDF files in a private domain, where a fully local solution is required

Requirement Analysis

- Input: Collection of PDF files
- Process: Extract text, split into chunks, embed, and index
- Output: Answers to user questions based on PDF content

Tech Stack Selection: Fully Local Solution

- PDF Processor: PyMuPDF
- Embedding model: e5-large
- Vector database: Qdrant or ChromaDB
- LLM: Phi-3

Key Considerations

- **PDF Handling:** PyMuPDF works well with exported PDF files, but struggles with scanned PDFs. For scanned PDFs, consider using MinerU[16].
- **Embedding Model:** The embedding model partially determines the relevance of retrieved content. For demonstration, we use a relative small model e5-large[17], which requiring less resource.
- **Document Splitting:** How you split documents is crucial for retrieval quality. We split by sentence in the demo, but you can experiment with different methods.
- **LLM Resource Usage:** LLMs consume the most resources and cause the longest latency. We've selected a small but effective model for demonstration. Consider different models based on your scenario.
- **Data Persistence:** We use Qdrant's memory version for simplicity. For data persistence, refer to Qdrant documentation to set up a local running Qdrant container.

[16] `https://github.com/opendatalab/MinerU`
[17] `https://huggingface.co/intfloat/e5-large`

Demonstration

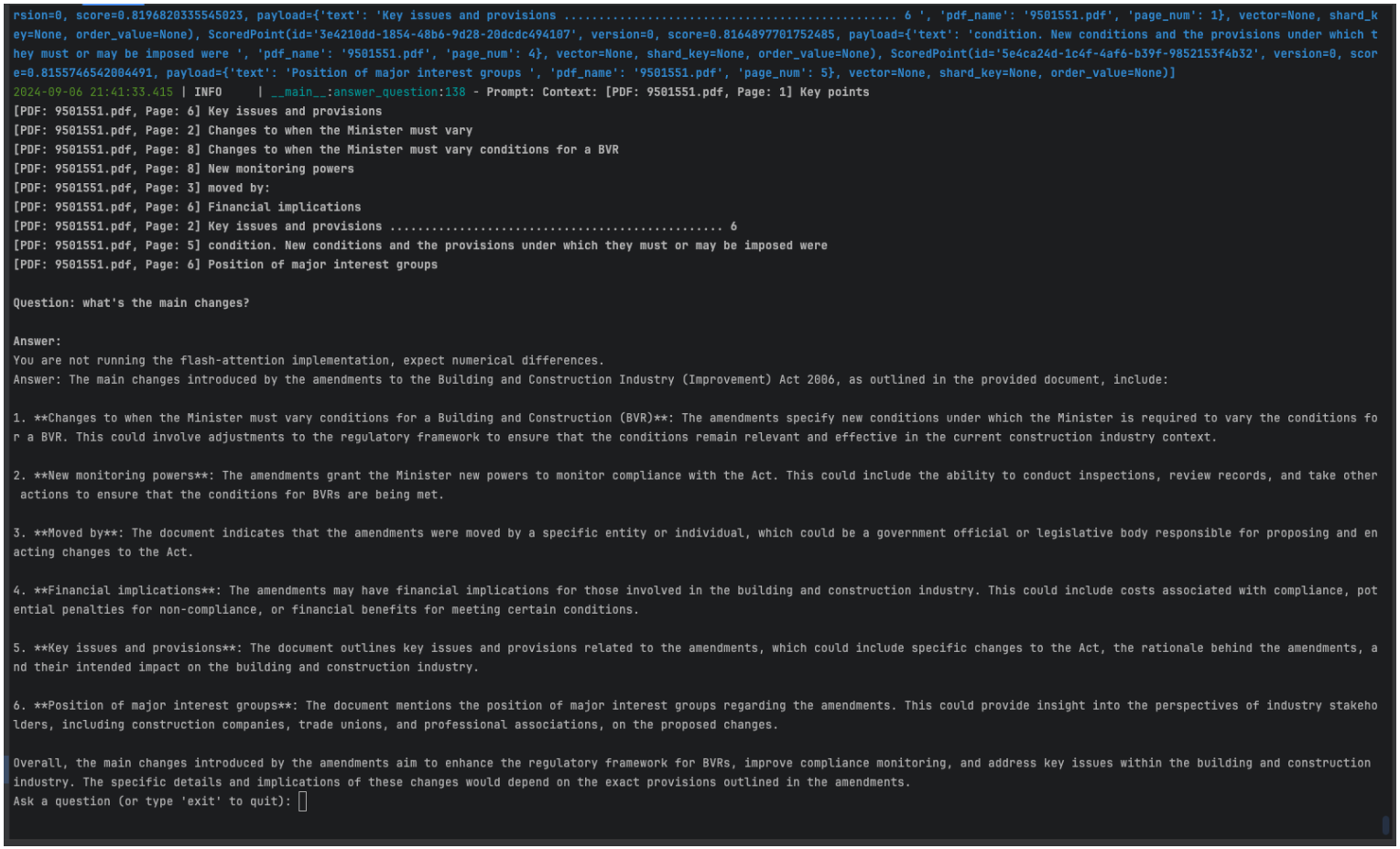

Figure: Demonstration with a PDF file regarding migration changes, codes available[18]

[18] https://github.com/AI4WA/AIEngineer/blob/main/RAG/demo2/demo.py

Advanced Fully Local RAG Solution

Current Status

Most components in our provided example perform well in terms of latency and accuracy. Only the LLM latency is not acceptable in Production Environment, taking around 30 seconds to get the response.

Areas for Further Exploration

1. **Improve Retrieved Context Quality**
   - Experiment with different embedding models
   - Try various document splitting methods
2. **Optimize LLM Performance**
   - Explore LLM Quantization for faster processing with similar accuracy
   - How to deploy quantized LLM? →llama.cpp[a]
3. **Hardware Considerations**
   - Balance model quality with hardware limitations
   - Better models typically require more: Storage, Memory, GPU, CPU

[a] `https://github.com/ggerganov/llama.cpp`

Summary

Exploring RAG Solutions

- **Fully Cloud**: Entirely cloud-based implementation (Demo1)
- **Fully Local**: All components run locally (Demo2)
- **Hybrid Approach**:
  - Manage **Retrieve** part locally
  - Utilize third-party LLM APIs: OpenAI, AWS, Azure, Google, etc

Advanced Topics for Further Exploration

1. Finetuning LLM
2. LLM Quantization
3. Scanned PDF Processing
4. Multimodal Data RAG

Note: We may add more tutorials on these topics in the future.