7-Day Challenge: Building LLM Applications

Candice You

Study Guide

Day 1 (Jan 6th): Introduction to Prompt Engineering and LLM Fundamentals (Self-Paced)

Course: ChatGPT Prompt Engineering for Developers.

Day 2 (Jan 7th): Building Systems with the ChatGPT API (Self-Paced)

Course: Building Systems with ChatGPT API.

Day 3 (Jan 8th): Introduction to LangChain for LLM Applications (Self-Paced)

Course: LangChain for LLM Application Development.

Day 4 (Jan 9th): Retrieval Augmented Generation (RAG) using LangChain (Self-Paced)

Course: LangChain Chat with Your Data.

Day 5 (Jan 10th): Functions, Tools, and Agents with LangChain (Self-Paced)

Course: Functions, Tools, and Agents with LangChain.

Day 6 (Jan 11th): Use Case Tutorial and Demonstration (Online Meeting)

Develop a chatbot for your data. Build a simple LLM agent by integrating tools.

Day 7 (Jan 12th): Showcase (Online Meeting)

Share your projects and learn from other participants.

Example Use Case

- **A risk manager or financial analyst may need to..**
  - Read regulation documents to search relevant parts of rule or detect recent rule changes (e.g., text, pdfs).
  - Sentiment analysis based on news articles (e.g., html, xml).
  - Extract market data for prices, volumes.. (e.g., csv, database).
  - Augment feature from unstructured data as input of other machine learning models.

- **Can we build AI powered financial assistant?**
  - LLM:
    - Extract structured information from unstructured files.
    - Translation, Summarization, Classification, Content Generation.

  - LLM + Tools:
    - Integrate LLM with other libraries or services.
    - Connect to external data and services.
    - Use custom statistical, machine learning and financial models.

What is LLM?

- **GPT**
  - Generative: Capable of predicting content
  - Pre-trained: Saved networks with weights pre-trained on large data
  - Transformer: Transform an input into another type of output

- **Embedding**
  - Embedding model (encoder) creates vector representation of a piece of text that captures its semantic meaning in its context.
  - Similar content is represented as similar vectors in the high dimensional vector space, measured by cosine similarity.

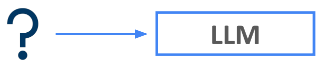
- **Self-attention**
  - Capture long-range dependencies and contextual information in natural language.

- **Transformer**
  - A deep learning framework composed of a stack of self-attention layers.
  - Traditional Transformer: voice-to-text, text-to-voice, text-to-image, etc
  - GPT: Generate text by sequentially predicting the next token in a sentence given the preceding tokens.

Methods to optimize LLM Usage

Prompt Engineering

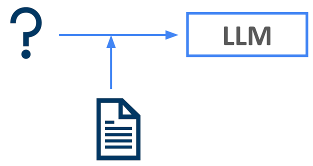
Provide context and guide LLM’s behavior



- Give instruction on the role and task of LLM through ‘system message’:
  - Summarization, feature extraction.
  - Translation (e.g., programming language)
  - Classification (e.g., sentiments)
  - Content generation (e.g., Personalized chatbot)
- Provide context information.
- Define output schema.
- **Few-shot:** Provide input-output examples.
- **Chain-of-Thought:** Breakdown the task into sub-tasks and ask for step-by-step reasoning.
- **ReAct:** Combines reasoning with acting and enables interaction with external tools.

RAG

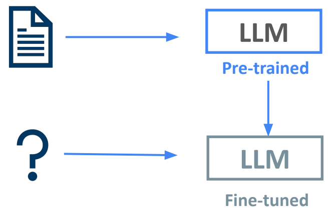
Provide LLM with domain-specific content.



- Retrieval:
  - Split document to smaller chunks.
  - Create embeddings for document chunks and save in vector database.
  - Given a query, search the most similar chunks.
  - Provide the query and the relevant chunks to LLM.
- Generation:
  - Question-Answering.
  - Enable memory and chat.
  - Router and pipeline.
  - Agentic RAG.

Fine-tuning

Continue training the LLM on a domain-specific dataset.



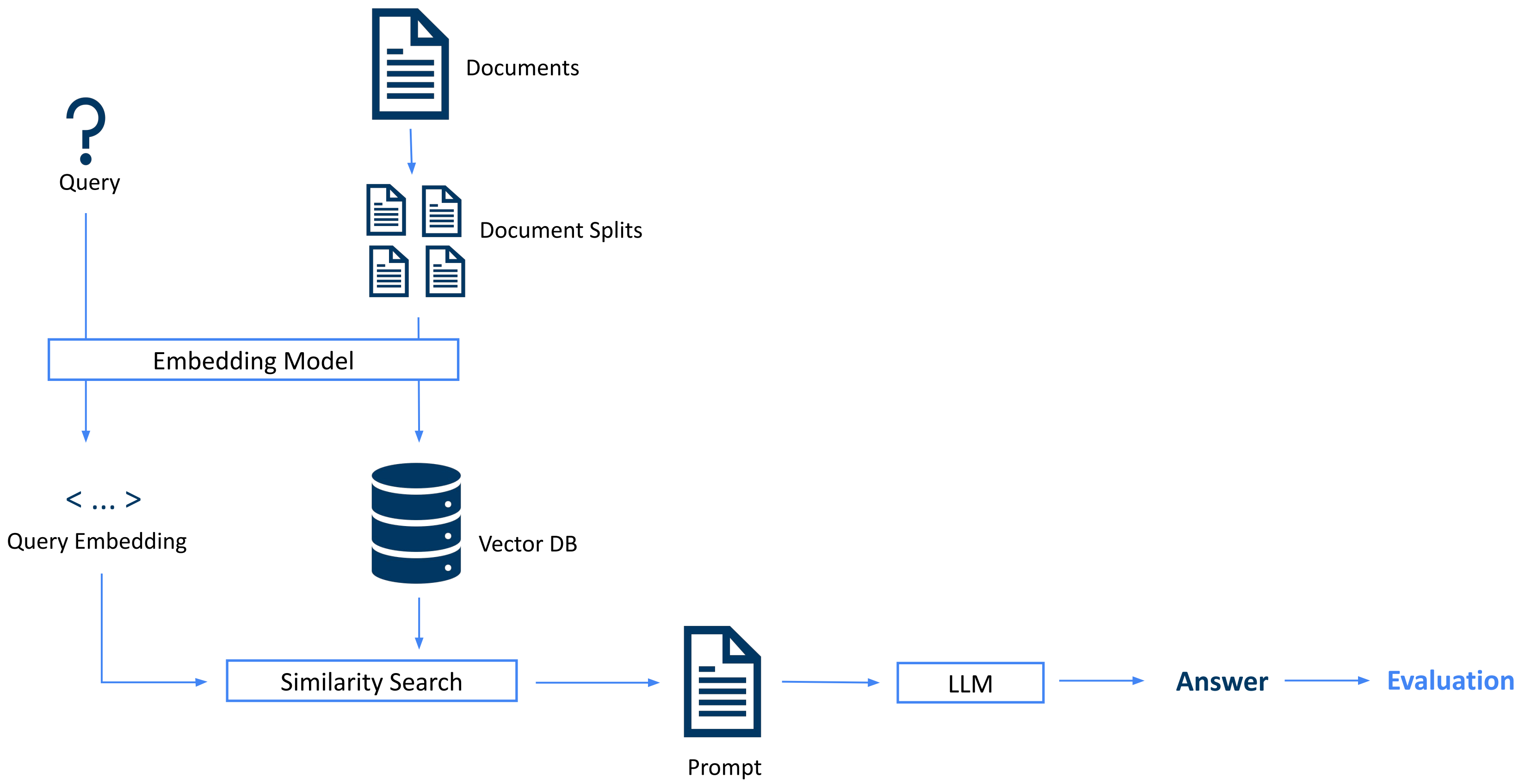
- Training Data preparation
- Select hyperparameters to tune
- Fine-tune the model
  - OpenAI fine-tuning API: https://platform.openai.com/docs/guides/fine-tuning/
  - Fine-tune open-source model
- Train-test split
- Evaluation

Model Selection

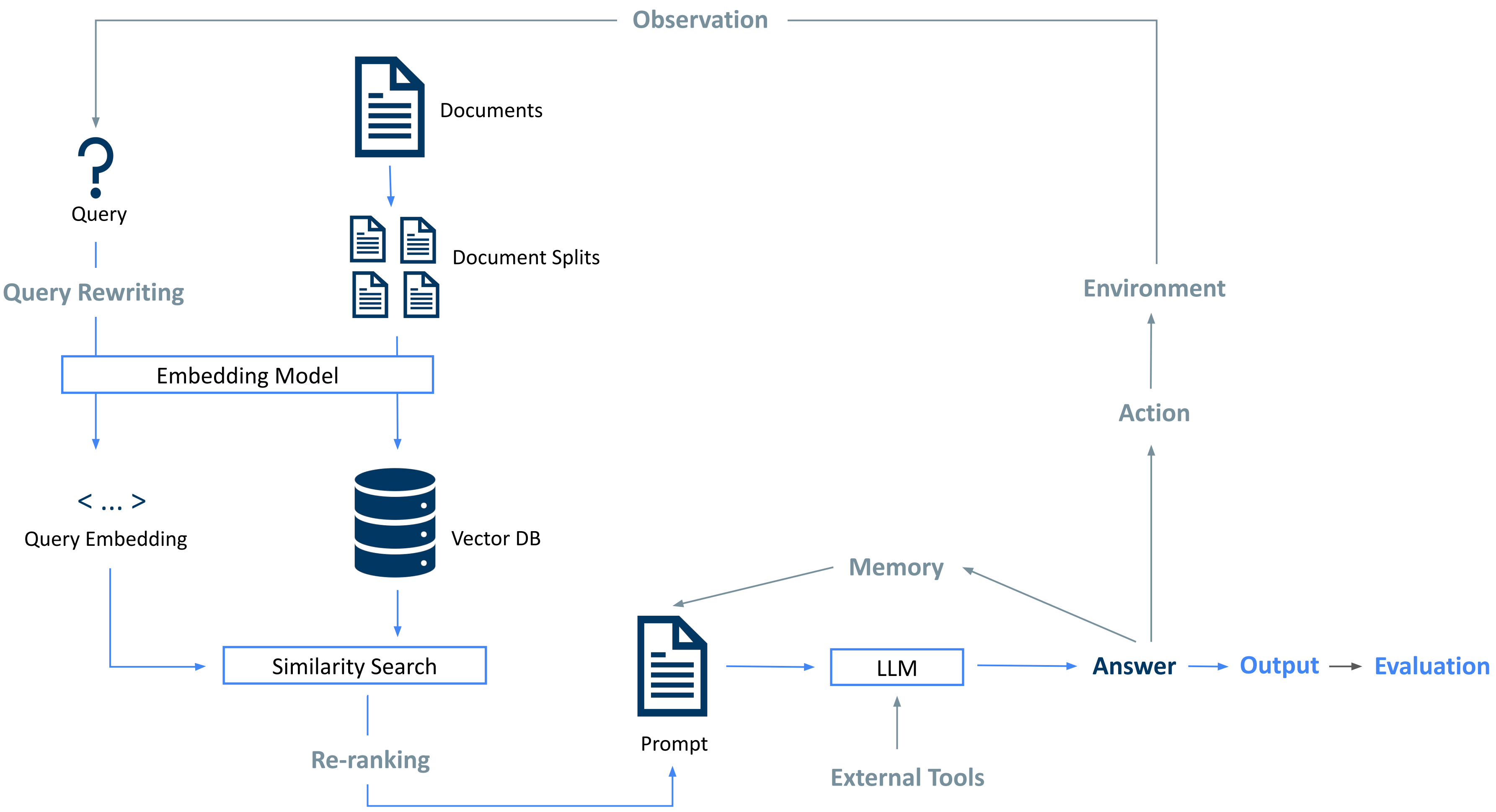
Complexity & Cost →

Prompt Engineering	RAG	Fine-tuning
Provide context and guide LLM's behavior Pro: ■ Provide quick iterations to validate the suitability of LLM on use case. ■ Establish a baseline to evaluate gaps for further optimization. Con: ■ Limited by context size of the LLM. ■ Inefficient token usage and latency. ■ Not scalable for systematic handing of complex problems.	**Provide LLM with domain-specific content.** Pro: ○ Introduce large amount of data to LLM as new knowledge. ○ Reduce hallucination by restricting the content. Con: ○ Cannot embed domain-specific terms (e.g., medicine names, financial products). ○ Cannot teach LLM new ways to 'think' (e.g., a new language). ○ Token usage.	**Continue training the LLM on a domain-specific dataset.** Pro: ○ Improve model performance and efficiency. ○ Reduce token usage. ○ Can teach the model a new language, writing style, or complex instructions. Con: ○ Not efficient for adding new knowledge to the model. ○ Involved training data preparation. ○ Slow feed-back loop, heavy compute.

RAG Overview



RAG Overview



RAG – Retrieval

Document Loading	**Convert data to text:** - (Semi-)Structured data: such as SQL database, csv/excel, JSON, blockchain smart contract etc. - Unstructured data: text, PDF, webpage, etc.
Document Splitting	LLM token limits: LLM can only take in a few thousand tokens at a time. For large documents, we need to split them into smaller **chunks**, and only pass the most relevant chunks to LLM.
Embedding and Vector Stores	**Indexing:** Create embedding (index) for the each of the document chunks. **Vector Store:** Embeddings are stored in a vector database, where you can easily look up similar vectors.
Retrieval	**Retriever** is an interface that retrieve relevant information based on user query. **Similarity search:** Compare query embedding with all vectors in the vector store. Search for top k similar chunks. Baseline method: retrieval with cosine similarity.

RAG – Generation

Question Answering	**‘Stuff’** method: Give all data to LLM at once and make a single call. This is limited to a small number of selected chunks. **‘Map Reduce’** method: Pass each chunk to LLM for a response and use another LLM call to summarize the individual responses. **‘Refine’** method: Build answer sequentially based on previous responses. This takes longer due to serial processing.
Chat	**Memory** is implemented by storing the conversation history as a memory variable. The full conversation is provided as context and consolidated with the follow-up question into a new standalone question. As the conversation gets longer, more tokens are sent to LLM, increasing the cost.
Chains / Pipeline	**Chains** execute **sequences of calls** to an LLM or other services. **Router** template tell LLM how to route between different query engines. ‘Chains’ in LangChain is similar to ‘QueryPipeline’ in LlamaIndex.
Evaluation	**Basic Evaluation:** Calculate the similarity of sample ground-truth answers and predicted answers, using LLM. **RAGAs score:** Faithfulness, Answer Relevancy, Context Precision, Context Recall. - Low Faithfulness: no fact behind the answer. - Low Answer Relevancy: factually correct, but not related to the question.

RAG Optimization

- **Model Selection:** Use embedding and LLM models fine-tuned on domain data.
- **Document Splitting Optimization:** Retain meaningful semantic relationships.
  - Experiment with the optimal chunk size and chunk overlap.
  - Define a hierarchy of separators (e.g. sections, paragraphs, sentences, characters).
  - Define additional metadata, for metadata filter (e.g. Chapter, Section).
- **Query Rewriting:**
  - Use LLM to re-write the query, to optimize the format and implement constraints. e.g., split the original question to a filter on metadata and a search.
  - Hypothetical document embeddings (HyDE): use LLM-generated hypothetical answer for similarity search.
- **Hybrid Search:** Use a combination of vector search (semantic similarity) and keyword search (exact match).
- **Re-ranking:** Use a two-stage pass for more accurate retrieval.
  - Stage 1: Use embedding-based retrieval to get a large set of candidate data chunks. This is fast but less accurate.
  - Stage 2: Re-calculate similarity for all candidates using original data (rule-based, LLM-based, etc.).
- **Router:** Classify the query and select the most relevant content, tools, or query engines based on the class.
- **Tools**: Integrate external functions or services, such as search engines, math tools, SQL database, or custom APIs.

RAG Optimization

- **Maximum Marginal Relevance:** Optimized retrieval to maximize diversity in retrieved data chunks.

- **Memory Optimization:** As the conversation gets longer, more tokens for memory are sent to LLM, increasing the cost.
  - Sliding-window memory: use most recent conversation history.
  - LLM summary memory: Use LLM to generate a summary of the conversation up to a token limit.
  - Vector data memory: Store text embeddings of conversation history in a vector database. Retrieve most relevant blocks.

- **Agentic RAG**
  - Integrate multiple RAG query engines, routers and external **tools**.
  - LLM is the reasoning engine of an agent and decides which **actions** to take.
  - Custom tools can be defined using python function and 'tool' decorator in LangChain.
  - Input schema for the tools can be defined using Pydantic.
  - Agent can write codes based on the text using PythonPEPL and execute the code to get the returned answer.

LangChain

- **Why LangChain?**
  - LangChain & LlamaIndex:
    - Open-source orchestration framework for developing LLM applications.
    - Generic interface for any LLM.
    - Modular design for flexible combination of different components.
  - LangChain vs LlamaIndex:
    - LangChain is versatile and focus on building a wide range of Gen AI applications.
    - LlamaIndex is best for indexing and retrieval, focus on building search applications.
- Example LangChain Classes:
  - Document Loading
    - (Semi-)Structured: SQLDatabaseLoader, CSVLoader, JSONLoader, BlockchainDocumentLoader, etc.
    - Unstructured: TextLoader, PyPDFLoader, WebBaseLoader, etc.
  - Document Splitting: RecursiveCharactorTextSplitter, CharactorTextSplitter (split at separator), TokenTextSplitter
  - Question Answering: RetrievalQA
  - Chat: ConversationalRetrievalChain
  - Query Pipeline: SimpleSequentialChain. LLMRouterChain
  - Memory:
    - Full memory: ConversationBufferMemory
    - ‘Sliding-window’ memory: ConversationBufferWindowMemory, CoversationTokenBufferMemory
    - LLM summary memory: ConversationSummaryBufferMemory