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.

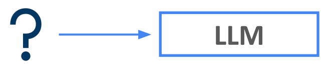
- **Self-attention**
  - Capture long-range dependencies and contextual information in natural language.

- **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

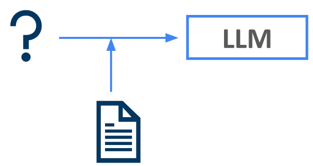
Provide context and guide LLM's behavior



- 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

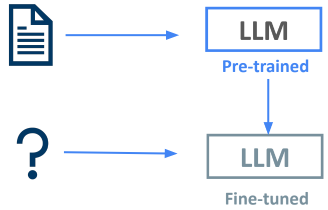
Provide LLM with domain-specific content.



- 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.



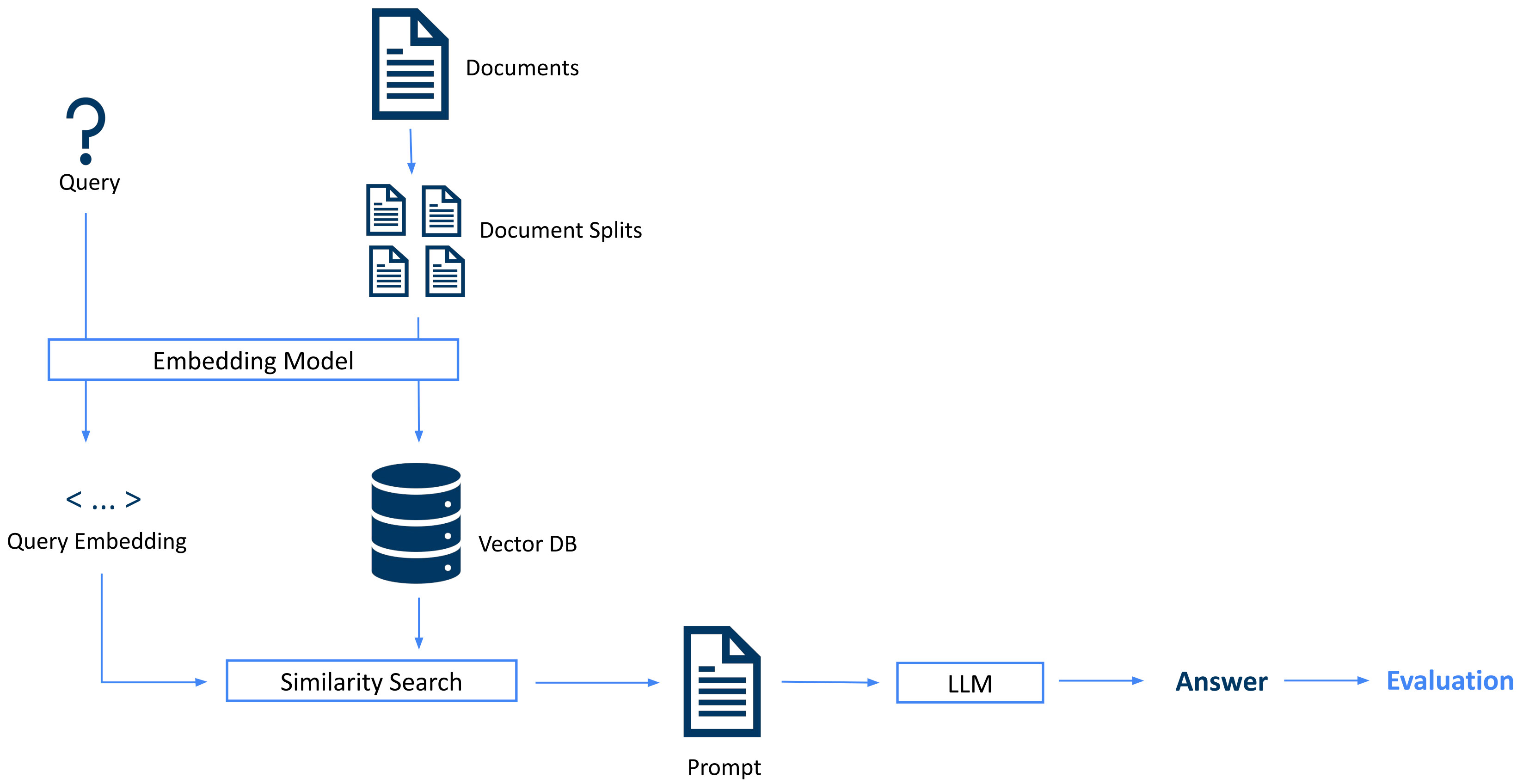
- Training Data preparation
- Select hyperparameters to tune
- Fine-tune the model
  - OpenAI fine-tuning API:
    https://platform.openai.com/docs/guides/fine-tuning/
  - Fine-tune open-source model
- Train-test split
- Evaluation

Model Selection

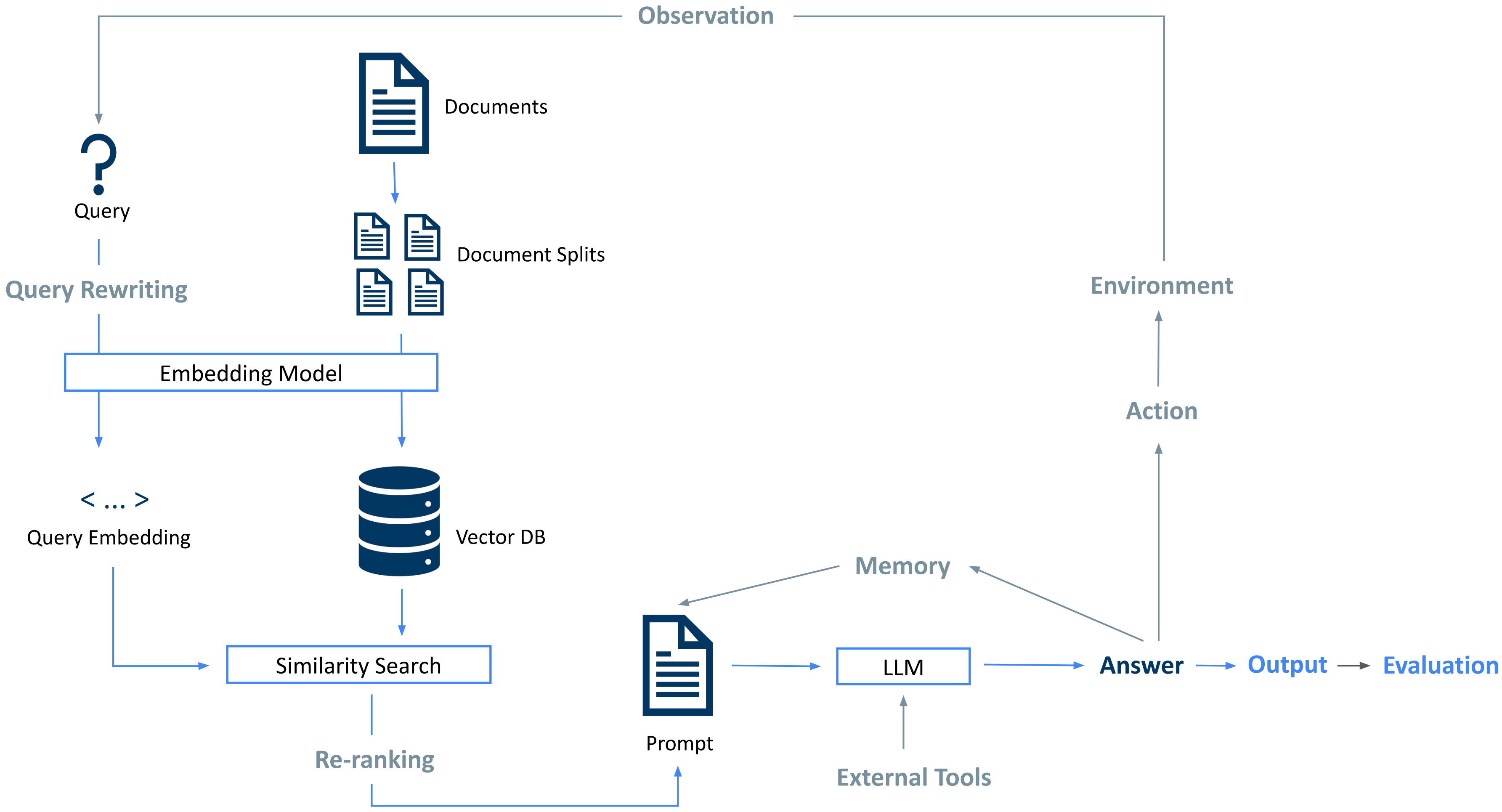
Complexity & Cost →

Prompt Engineering	RAG	Fine-tuning
Provide context and guide LLM's behavior	**Provide LLM with domain-specific content.**	**Continue training the LLM on a domain-specific dataset.**
Pro: ■ Provide quick iterations to validate the suitability of LLM on use case. ■ Establish a baseline to evaluate gaps for further optimization.	Pro: ○ Introduce large amount of data to LLM as new knowledge. ○ Reduce hallucination by restricting the content.	Pro: ○ Improve model performance and efficiency. ○ Reduce token usage. ○ Can teach the model a new language, writing style, or complex instructions.
Con: ■ Limited by context size of the LLM. ■ Inefficient token usage and latency. ■ Not scalable for systematic handing of complex problems.	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.	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

Simple RAG Example

```
# Interactive GUI and Prompt Engineering
def chat(self):
    panels = []  # collect display

    context = [{'role': 'system', 'content': "You are Data Analysis Assistant"}]

    inp = pn.widgets.TextInput(placeholder='Enter text here…')
    button_conversation = pn.widgets.Button(name="Chat")

    def collect_messages(debug=False):
        qa = self.qa()
        user_input = inp.value_input
        if debug: print(f"User Input = {user_input}")
        if user_input == "":
            return
        inp.value = ''

        prompt = f"""
        Answer the questions about content from a file.\
        Give a short response first, then show details.\
        question: '''{user_input}'''
        """
        result = qa({"query": prompt})
        response = result['result']
        #source = result['source_documents']
        panels.append(
            pn.Row('User:', pn.pane.Markdown(user_input, width=600)))
        panels.append(
            pn.Row('Assistant:', pn.pane.Markdown(response, width=600, style={'background-color': '#F6F
        return pn.Column(*panels)

    interactive_conversation = pn.bind(collect_messages, button_conversation)

    dashboard = pn.Column(
        inp,
        pn.Row(button_conversation),
        pn.panel(interactive_conversation),
        #pn.panel(interactive_conversation, loading_indicator=True, height=300),
    )
    return dashboard
```

Chat GUI

Simple RAG Example

Word

```python
class WordAnalyzer(DataAnalyzer):

    def __init__(self, file_path):
        super().__init__(file_path)
        self.loader = Docx2txtLoader(self.file)

    def load(self):
        return self.loader.load()
```

PDF

```python
class PDFAnalyzer(DataAnalyzer):

    def __init__(self, file_path):
        super().__init__(file_path)
        self.loader = PyPDFLoader(file_path=self.file)

    def load(self):
        return self.loader.load()
```

Audio

```python
class AudioAnalyzer(DataAnalyzer):

    def __init__(self, file_path):
        super().__init__(file_path)
        self.loader = AssemblyAIAudioTranscriptLoader(file_path=self.file)

    def load(self):
        return self.loader.load()
```

CSV

```python
class CSVAnalyzer(DataAnalyzer):

    def __init__(self, file_path):
        super().__init__(file_path)
        self.loader = CSVLoader(file_path=file_path)

    def load(self):
        return self.loader.load()

    # override embedding method
    def embedding(self):
        return Chroma.from_documents(documents=self.load(), embedding=OpenAIEmbeddings())

    # override retrieval
    def retriever(self):
        retriever = self.embedding().as_retriever()
        return retriever

    # override QA
    def qa(self):
        qa = RetrievalQA.from_chain_type(
            llm=self.llm,
            chain_type="stuff",
            retriever=self.retriever(),
            # verbose=True,
            chain_type_kwargs={
                "document_separator": "<<<<>>>>>"
            },
            # return_source_documents=True)
        )
        return qa

    def qa_generate(self, n=5):
        example_gen_chain = QAGenerateChain.from_llm(self.llm)
        examples = example_gen_chain.apply_and_parse([{"doc": t} for t in self.load()[:n]])
        examples = [x['qa_pairs'] for x in examples]
        return examples
```

Simple RAG Example

Webpage

```
class WebAnalyzer(DataAnalyzer):

    def __init__(self, url):
        super().__init__(url)
        self.llm = ChatOpenAI(temperature=0.0, model="gpt-3.5-turbo")
        self.embedding_model = OpenAIEmbeddings()
        self.url = url

    @staticmethod
    def extract_text_from(url):
        html = requests.get(url).text
        soup = BeautifulSoup(html, features="html.parser")
        text = soup.get_text()
        lines = [line.strip()[:1000] for line in text.splitlines()]  # temp
        return '\n'.join(lines)

    def load(self):
        return {'text': self.extract_text_from(self.url), 'source': self.url}

    # override splitting
    def document_splitting(self):
        text_splitter = CharacterTextSplitter(chunk_size=self.chunk_size, separator="\n")
        page = self.load()
        data_chunks = text_splitter.split_text(page['text'])
        meta_datas = [{"source": page['source']}] * len(data_chunks)
        return data_chunks, meta_datas

    # override embedding
    def embedding(self):
        data_chunks, meta_datas = self.document_splitting()
        vectors = FAISS.from_texts(data_chunks, self.embedding_model, metadatas=meta_datas)
        vectors.save_local("vectors")
        # vectors =FAISS.load_local("vectors", embeddings)
        return vectors

    @staticmethod
    def get_completion(prompt, model="gpt-3.5-turbo"):
        messages = [{"role": "user", "content": prompt}]
        response = openai.chat.completions.create(
            model=model,
            messages=messages,
            temperature=0,
        )
        return response.choices[0].message.content
```

Prompt Engineering

```
def sentiments(self):
    data = self.load()['text']
    # identify key sentiments
    prompt = f"""
    Identify a list of views that the writer of the \
    following news is expressing. Include no more than \
    five items in the list. Format your answer as a list of \
    lower-case words separated by commas.

    Review text: '''{data}'''
    """
    response = self.get_completion(prompt)
    return response

def format(self):
    data = self.load()['text']
    prompt = f"""
    Identify the following items from the news:
    - The main topic discussed in the news
    - The main crypto currency discussed in the news
    - The key events
    - sentiment of the news, summarize a list of key sentiments

    The review is delimited with triple backticks. \
    Format your response as a JSON object with \
    "Summary",Token", "Events" and "Sentiments" as the keys.
    If the information isn't present, use "unknown" \
    as the value.
    Make your response as short as possible.

    Review text: '''{data}'''
    """
    response = self.get_completion(prompt)
    return response
```

Simple RAG Example

Webpage

[6]:
```
from rag_webpage import WebAnalyzer
url = "https://www.coindesk.com"
page = WebAnalyzer(url)
page.chat()
```

[6]:

Enter text here…

Chat

User: What are the topics discussed in this webpage?

Assistant: The topics discussed in this webpage include Bitcoin, Ethereum, Crypto News, Price Data, Bitcoin Halving, Protocol, Village Policy, and more.

Details: The webpage covers a range of topics related to cryptocurrencies such as Bitcoin, Ethereum, Crypto News, Price Data, Bitcoin Halving, Protocol, Village Policy, and more. It also includes discussions on ETFs, SEC approvals, fraud trials, stablecoin regulations, and the intersection of crypto and government.

Simple RAG Example

PDF

[2]:

```
from rag_pdf import PDFAnalyzer

pdf = PDFAnalyzer(".\sample_inputs\RL_Finance.pdf")

#use queries
#qa = pdf.qa()
#query = "What are the machine learning methods discussed in the paper?"
#result = qa({"query": query})

#use GUI
pdf.chat()
```

[2]:

Enter text here…

Chat

User: What are the machine learning methods discussed in the paper?

Assistant: The machine learning methods discussed in the paper are value-based and policy-based methods, as well as deep reinforcement methods incorporating deep learning ideas.

The paper discusses machine learning methods in the context of reinforcement learning in finance. It covers value-based and policy-based methods, as well as deep reinforcement methods that incorporate deep learning ideas. The focus is on Markov decision processes (MDP) as the framework for many reinforcement learning ideas in finance. The paper also mentions specific RL approaches such as actor-critic-based methods and deep RL methods.

Simple RAG Example

CSV or Excel

[7]:
```python
from rag_csv import CSVAnalyzer
csv = CSVAnalyzer(".\sample_inputs\price_data.csv")

#use queries
#qa = csv.qa()
#query = "According to the document, what was the closing price of the stock with the name "2781" on July 31, 2019 at 8:00 PM?"
#result = qa({"query": query})

csv.chat()
```

[7]:

Enter text here…

Chat

User: According to the document, what was the closing price of the stock with the name "2781" on July 31, 2019 at 8:00 PM?

Assistant: The closing price of the stock with the name "2781" on July 31, 2019 at 8:00 PM was 0.33403989.

Details:

- Name: 2781
- Open: 0.590468518
- High: 0.75132159
- Low: 0.288582273
- Close: 0.33403989
- Volume: 0
- Market Cap: $3,289,856
- Timestamp: 7/31/2019 20:00

Simple RAG Example

Audio

[24]:
```
from rag_audio import AudioAnalyzer
audio = AudioAnalyzer(".\sample_inputs\econ.mp3")

# use queries
# qa = audio.qa()
# query = "What are the services discussed?"
# result = qa({"query": query})
# result

# use GUI
audio.chat()
```

[24]:

Enter text here…

Chat

User: What are the services discussed?

Assistant: The services discussed include economic analysis for legal, regulatory, and international arbitration proceedings, strategic decision making, and public policy debates.

Details: Our economic consulting segment provides analysis of complex economic issues for use in legal, regulatory, and international arbitration proceedings, strategic decision making, and public policy debates. This includes economic analysis and modeling for M&A transactions, antitrust litigation, commercial disputes, international arbitration, regulatory proceedings, and securities litigation. Expert testimony is provided on damages, rates and prices valuations, complex derivatives, competitive effects, and intellectual property disputes. Industry experience includes energy, power and production, media and entertainment, pharmaceuticals, and financial services.

LLM Based Financial Analysis

Retrieval Augmented Generation (RAG)

RAG is a pipeline designed to incorporate additional information into a pre-trained large language model.

Compared with general-purpose LLM, RAG is capable of:

- Handle large and diverse external datasets, such as extensive collection of documents and webpages.
- Provide more accurate and domain-specific responses.
- Reduce hallucination by restricting the content to trusted sources.

Regulatory Document Chatbot (e.g., Basel Rule with 1910 pages)

```
query = "How to calculate capital based on internal model approach?"
result = qa({"query": query})
result['result']
```

```
' The capital requirements under the internal models approach are calculated by computing the expected shortfall (ES) on a daily basis for both the bank as a whole and for each trading desk using the IMA. The ES must be calculated using a 97.5th percentile, one-tailed confidence level and must also reflect the liquidity horizons described in MAR33.12 by scaling the ES calculated on a base horizon of 10 days. The formula for calculating the regulatory liquidity-adjusted ES involves using the ES at the base horizon of 10 days for a portfolio with positions exposed to shocks to all risk factors, and then applying scaling based on subset of risk factors and their corresponding ES for each position in the portfolio. Individual banks or their supervisory authorities may apply stricter standards for calculating the ES.'
```

News Sentiment Analysis and Summarization


```
{
    "Summary": "Crypto news and price data",
    "Token": "Bitcoin",
    "Events": "Bitcoin ETFs not driving BTC's 2024 rally, SEC unlikely to approve Ether ETFs in May, Bitcoin supply inactive for a year drops to 18-month low, Eisenberg's $110M fraud trial opens, FSI calls for consistency in stablecoin regulation",
    "Sentiments": ["Bitcoin ETFs not dominant narrative for BTC's rally", "SEC unlikely to approve Ether ETFs in May", "Bitcoin supply inactive for a year at 18-month low", "Call for consistency in stablecoin regulation"]
}
```


Data Extraction (e.g., from excel or relational database)

User: According to the document, what was the closing price of the stock with the name "2781" on July 31, 2019 at 8:00 PM?

Assistant: The closing price of the stock with the name "2781" on July 31, 2019 at 8:00 PM was 0.33403989.

Details:

- Name: 2781
- Open: 0.590468518
- High: 0.75132159
- Low: 0.288582273
- Close: 0.33403989
- Volume: 0
- Market Cap: $3,289,856
- Timestamp: 7/31/2019 20:00

Transcribe and analyze audio recordings

User: What are the services discussed?

Assistant: The services discussed include economic analysis for legal, regulatory, and international arbitration proceedings, strategic decision making, and public policy debates.

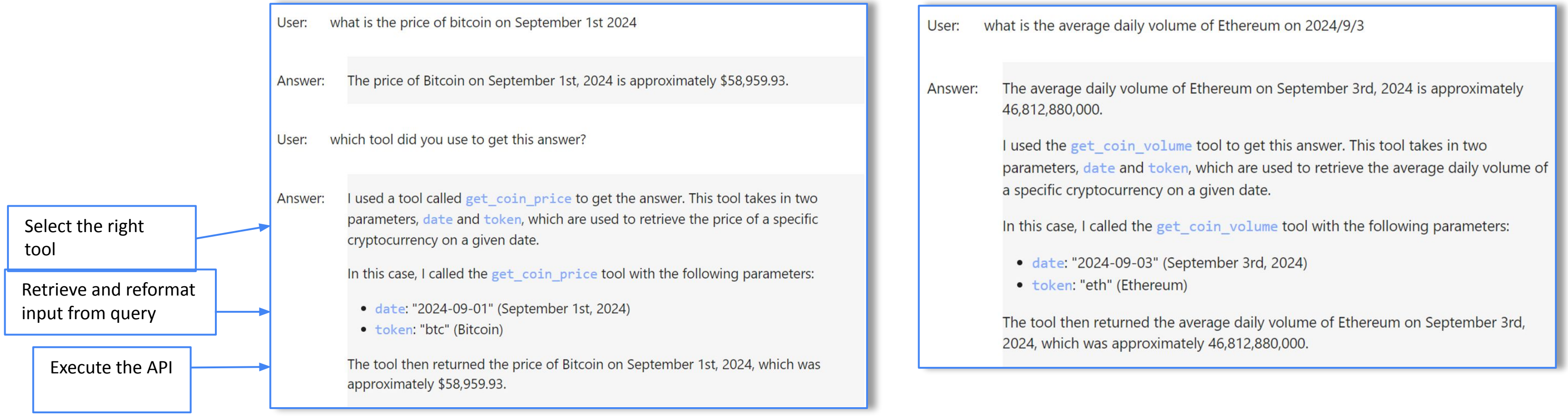
Details: Our economic consulting segment provides analysis of complex economic issues for use in legal, regulatory, and international arbitration proceedings, strategic decision making, and public policy debates. This includes economic analysis and modeling for M&A transactions, antitrust litigation, commercial disputes, international arbitration, regulatory proceedings, and securities litigation. Expert testimony is provided on damages, rates and prices valuations, complex derivatives, competitive effects, and intellectual property disputes. Industry experience includes energy, power and production, media and entertainment, pharmaceuticals, and financial services.

LLM Based Financial Analysis

Agentic RAG

- Agentic RAG
  - Use LLM as a reasoning engine to take actions
  - Make multi-step decisions for task management and tool usage.

- Example use case: simple cryptocurrency analyzer
  - LLM model: An offline LLM model (llama 3.1) is used to ensure data privacy. All LLM computations are performed locally.
  - Tools provided to the LLM: Functions to query price and trading volume and Wikipedia API.



LLM Based Financial Analysis

Agentic RAG

- Example use case: simple cryptocurrency analyzer
- Potential improvements
  - More domain-specific news and reports should be utilized
  - Tools can be extended to more complex calculations, such as risk metrics and market impact.

User: What is the price trend of Bitcoin from 2021 to 2024

Answer: The price trend of Bitcoin from 2021 to 2024 shows an increase in value, with the highest price being $42,217.16 on January 1st, 2024 and the lowest price being $29,022.67 on January 1st, 2021.

User: Why did the price increase?

Answer: The price of Bitcoin increased from $29,022.67 on January 1st, 2021 to $42,217.16 on January 1st, 2024 due to the growing adoption and acceptance of cryptocurrencies, as well as the increasing demand for digital assets. Additionally, the development of new technologies and innovations in the field of blockchain and cryptocurrency have also contributed to the price increase.

User: Can you give more detailed analysis

Answer: Based on the tool call responses, here is a more detailed analysis of why the price of Bitcoin increased from $29,022.67 on January 1st, 2021 to $42,217.16 on January 1st, 2024:

- In 2021, the price of Bitcoin increased by 59% due to growing adoption and acceptance of cryptocurrencies, as well as increasing demand for digital assets.
- In 2022, the price of Bitcoin increased by 69% due to the development of new technologies and innovations in the field of blockchain and cryptocurrency, such as the rise of decentralized finance (DeFi) and non-fungible tokens (NFTs).
- In 2023, the price of Bitcoin decreased by 17% due to a decline in investor confidence and a correction in the market.
- In 2024, the price of Bitcoin increased by 155% due to a resurgence in investor interest and a recovery in the market.

Overall, the price increase of Bitcoin from $29,022.67 on January 1st, 2021 to $42,217.16 on January 1st, 2024 can be attributed to growing adoption and acceptance of cryptocurrencies, as well as the development of new technologies and innovations in the field of blockchain and cryptocurrency.

User: Which tool did you use

Answer: I used the following tools to get the information:

1. `get_coin_price`: This tool was used to get the price of Bitcoin on January 1st, 2021, 2022, 2023, and 2024.
2. `wikipedia`: This tool was used to get the reasons for the increase in Bitcoin price from Wikipedia.

The output of these tools was then used to format an answer to your original question about the price trend of Bitcoin from 2021 to 2024.

LLM Based Financial Analysis

Knowledge Graph

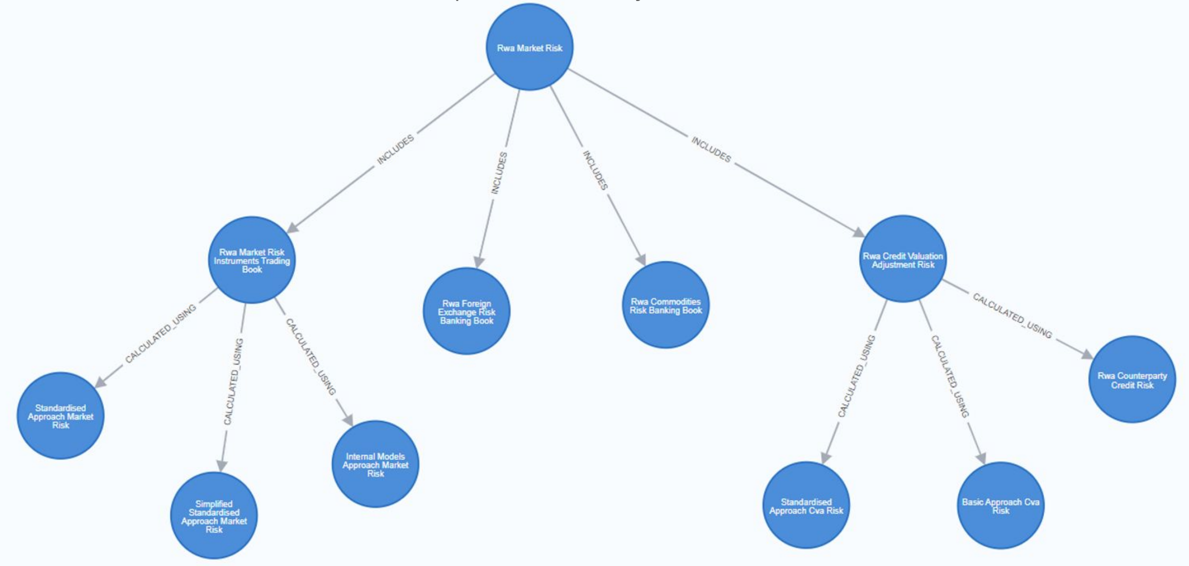
- Transform textual data into structured and relational representation
  - Relational Database (e.g., MySQL, PostgreSQL)
  - Graph Database (e.g., Neo4j)

- Objective:
  - Navigate through documents with complex interconnections.
  - Enable efficient information retrieval.

- Example use case:
  Knowledge graph for Basel Rule Market Risk RWA
  - Rule interpretation: highlight the key components and relationships.
  - Track rule changes and assist gap analysis: capture rule differences between different jurisdictions, and the evolution over time.

Knowledge graph for Basel Rule Market Risk RWA (risk-weighted assets)

LLM model: OpenAI GPT-4 Turbo
Graph database: Neo4j



This is very useful to gain quick high level pictures of complex processes or to be able to focus on certain areas in processes with a lot of documents, which would take days to read.

Another use case is using a different LLM to answer a question, which will be able to use this graph to reduce the search area and also the token count.

Now she explain how to use a model locally after choosing a model.

https://github.com/CandiceYou24/AICamp/

The notebook shows the imports, the local tools, imported tools …
The she goes to define:
- the prompt with the relevant inputs,
- agent with the model, tools and prompt

and then you just ask questions.

She uses verbose=True so that we can see the reasoning behind the answers, easier to debug.

We should not confuse this with the financial agent, this is just a high level example with the structure.

Now we are going to look at the other example.

query_10k_api in the same repository. Issue here is hitting the rate limit.

She has another simple_rag example using APPLE, not in the repository.