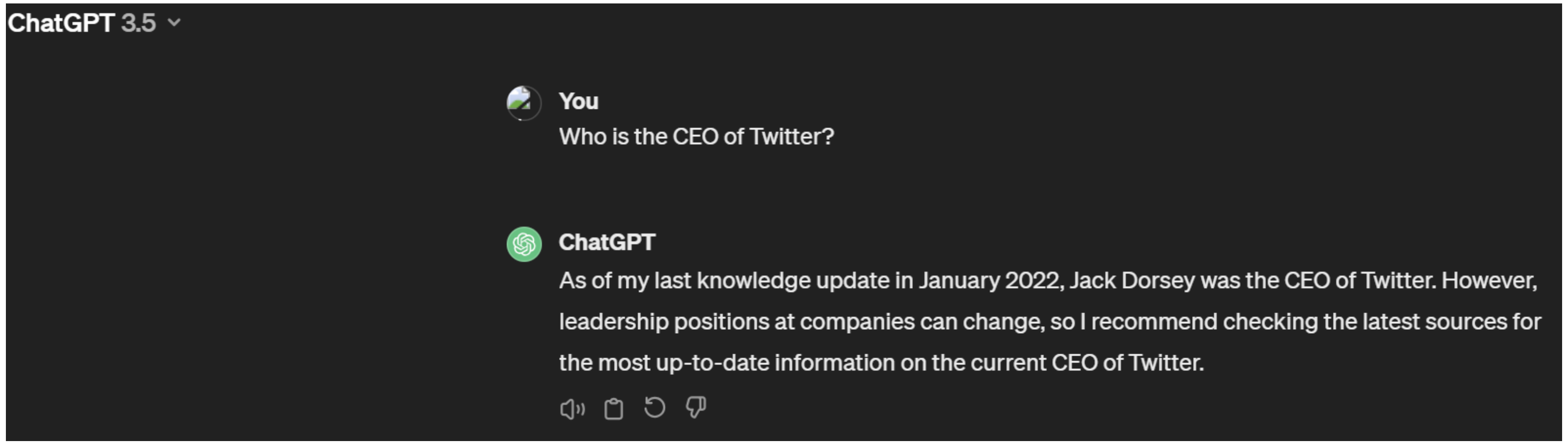
RAG Techniques

Palacode Narayana Iyer Anantharaman

6th Sep 2024

Why Retrieval Augmented Generation (RAG)?

- The datasets that were used for LLM training are not live and real time.



ChatGPT 3.5

You
Who is the CEO of Twitter?

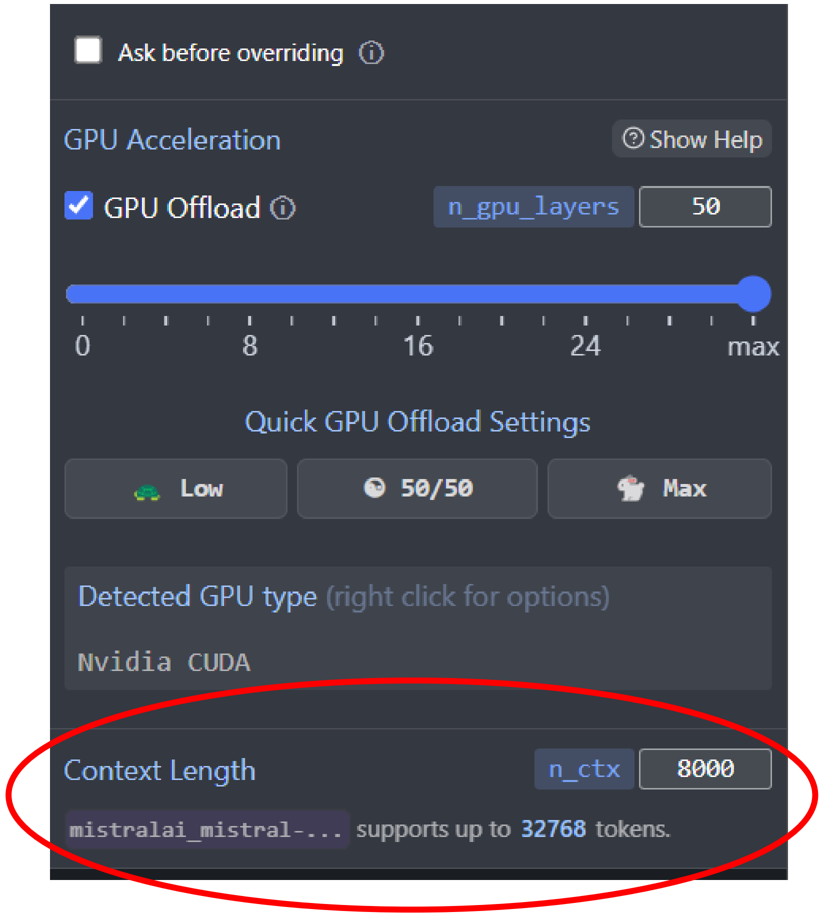
ChatGPT
As of my last knowledge update in January 2022, Jack Dorsey was the CEO of Twitter. However, leadership positions at companies can change, so I recommend checking the latest sources for the most up-to-date information on the current CEO of Twitter.

- Training datasets are usually public information. Private data are not included in foundational LLMs. Supporting custom data is a major requirement for the LLMs. For example:
    - Proprietary code, design documents
    - Company confidential financial documents

- One way to provide live data to LLM is by using the context window. But this has limitations.

Context Limitations

- LLM's are unaware of concepts outside of their training set
- Filling gaps in knowledge with assumptions
- Very hard to teach LLM's about new concepts



Ask before overriding

GPU Acceleration

GPU Offload ✓ n_gpu_layers 50

Detected GPU type (right click for options)

Nvidia CUDA

Context Length n_ctx 8000

mistralai_mistral-... supports up to 32768 tokens.

Naïve RAG: Illustration

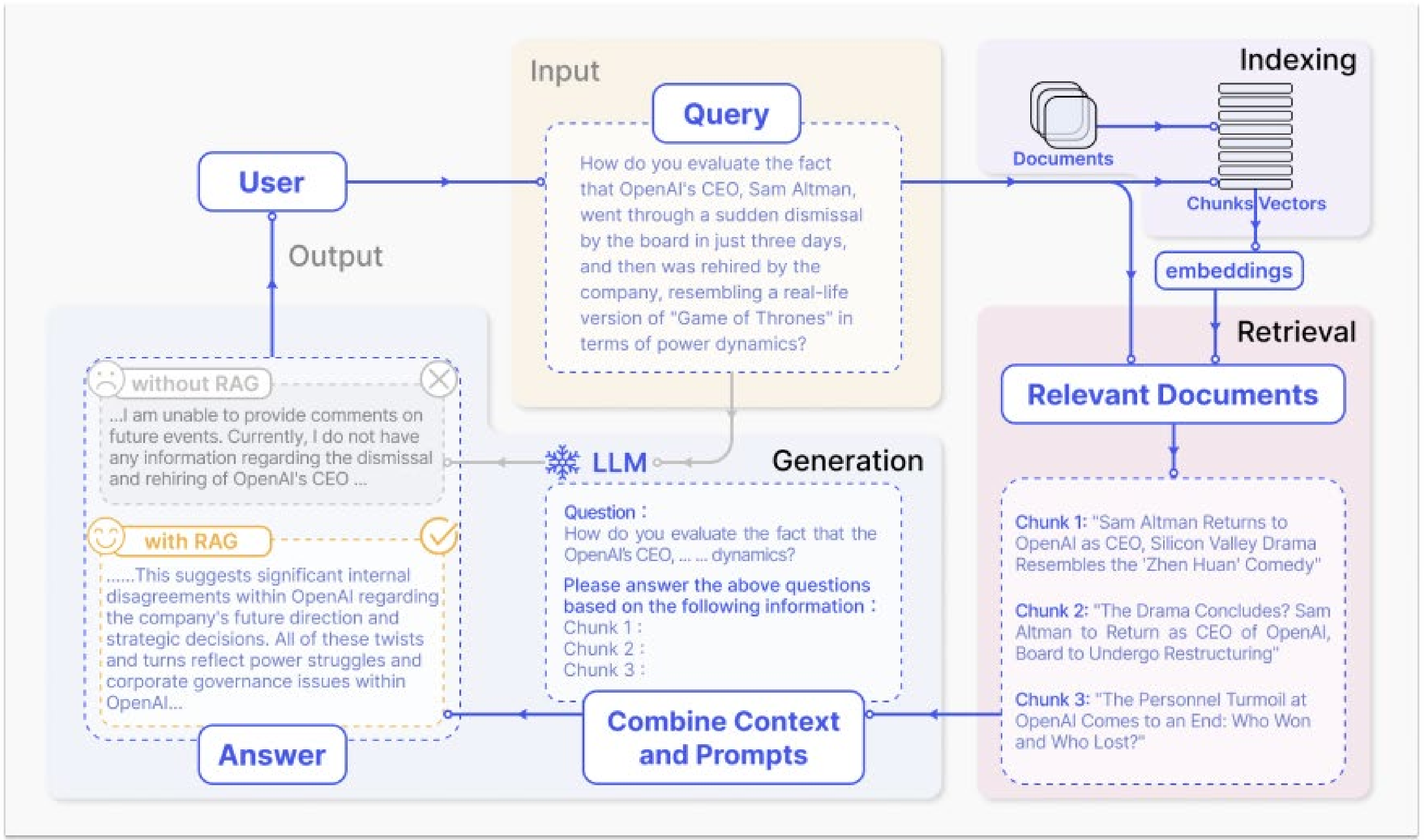

Input
Query
How do you evaluate the fact that OpenAI's CEO, Sam Altman, went through a sudden dismissal by the board in just three days, and then was rehired by the company, resembling a real-life version of "Game of Thrones" in terms of power dynamics?
User
Output
Indexing
Documents
Chunks
Vectors
embeddings
Retrieval
Relevant Documents
Chunk 1: "Sam Altman Returns to OpenAI as CEO, Silicon Valley Drama Resembles the 'Zhen Huan' Comedy"
Chunk 2: "The Drama Concludes? Sam Altman to Return as CEO of OpenAI, Board to Undergo Restructuring"
Chunk 3: "The Personnel Turmoil at OpenAI Comes to an End: Who Won and Who Lost?"
LLM
Generation
Question :
How do you evaluate the fact that the OpenAI's CEO, ... dynamics?
Please answer the above questions based on the following information :
Chunk 1 :
Chunk 2 :
Chunk 3 :
Combine Context and Prompts
without RAG
...I am unable to provide comments on future events. Currently, I do not have any information regarding the dismissal and rehiring of OpenAI's CEO ...
with RAG
......This suggests significant internal disagreements within OpenAI regarding the company's future direction and strategic decisions. All of these twists and turns reflect power struggles and corporate governance issues within OpenAI...
Answer

Fig. 2. A representative instance of the RAG process applied to question answering. It mainly consists of 3 steps. 1) Indexing. Documents are split into chunks, encoded into vectors, and stored in a vector database. 2) Retrieval. Retrieve the Top k chunks most relevant to the question based on semantic similarity. 3) Generation. Input the original question and the retrieved chunks together into LLM to generate the final answer.

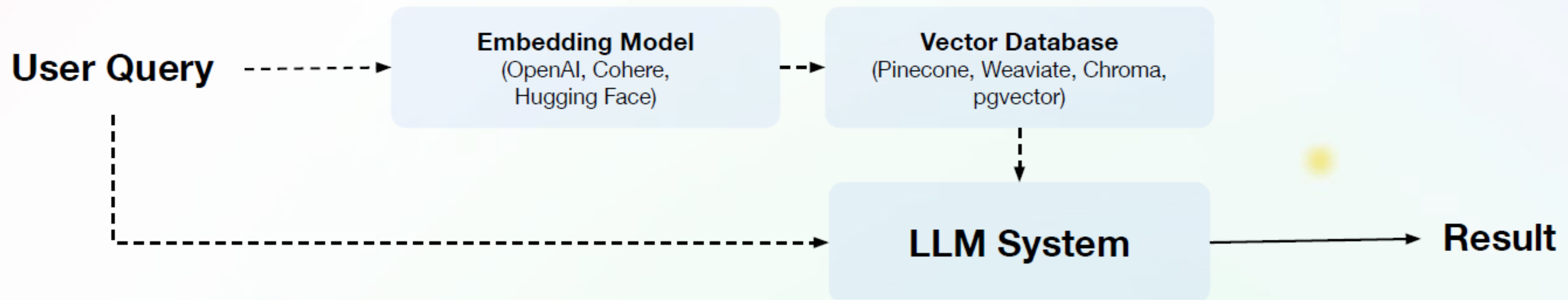
Fig Credits: RAG for LLM Models – A survey, Gao et al.

Retrieval Augmented Generation

Use-cases:

- **Knowledge Base question answering**
  - Library documentation
  - Technical documents
  - Code

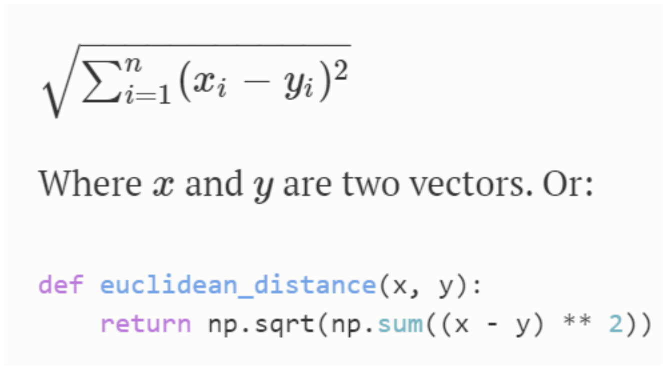
- **Technical Summarization**


User Query
Embedding Model
(OpenAI, Cohere, Hugging Face)
Vector Database
(Pinecone, Weaviate, Chroma, pgvector)
LLM System
Result

Ref: “Emerging architectures for LLM applications”, https://www.youtube.com/watch?v=Pft04KLw5Lk&t=670s

Euclidean Distance and Cosine Similarity

- Euclidean Distance is defined as:

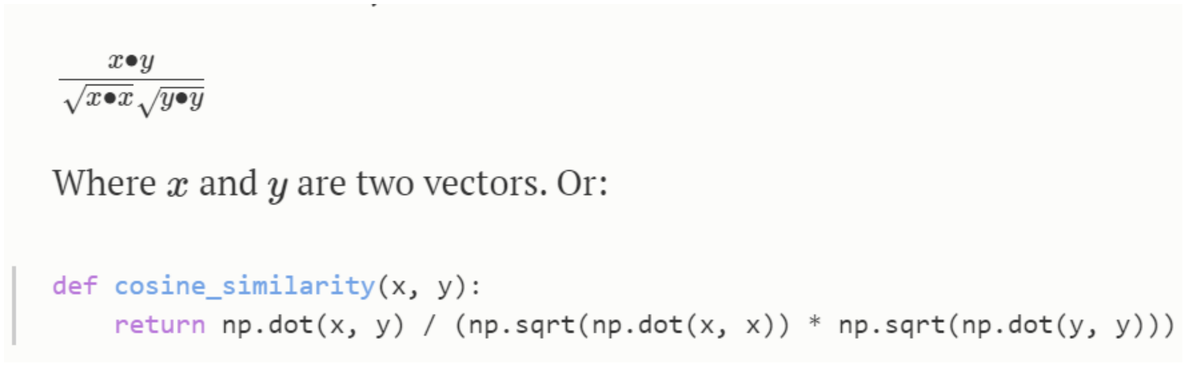


$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Where x and y are two vectors. Or:

```
def euclidean_distance(x, y):
    return np.sqrt(np.sum((x - y) ** 2))
```

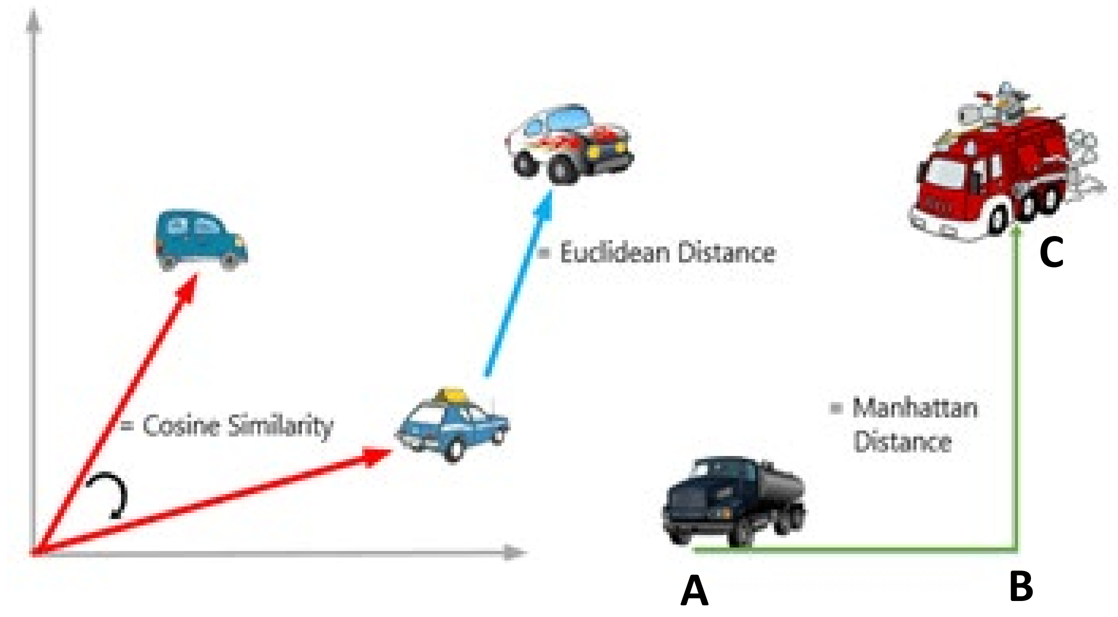
- Cosine Similarity:



$$\frac{x \bullet y}{\sqrt{x \bullet x}\sqrt{y \bullet y}}$$

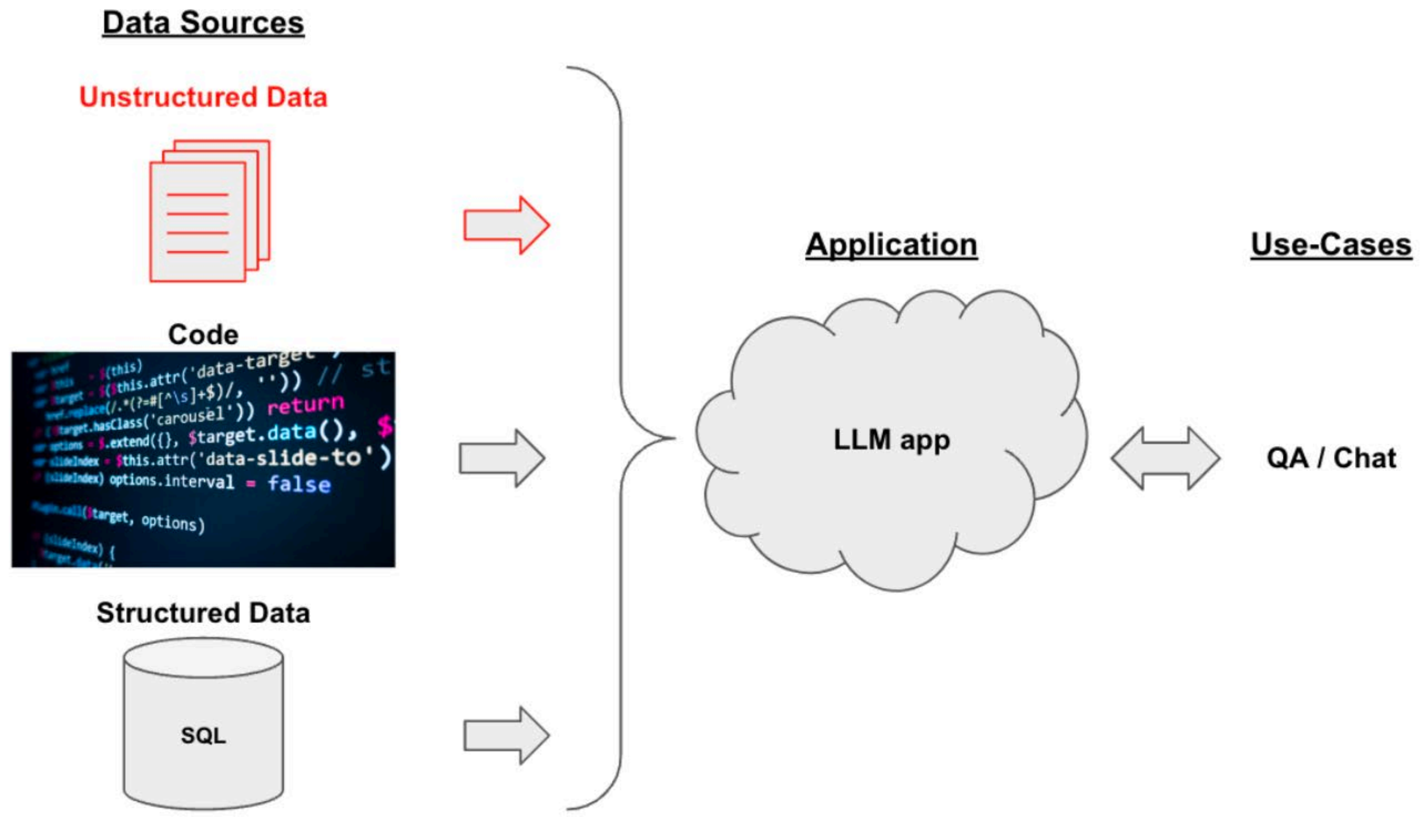
Where x and y are two vectors. Or:

```
def cosine_similarity(x, y):
    return np.dot(x, y) / (np.sqrt(np.dot(x, x)) * np.sqrt(np.dot(y, y)))
```

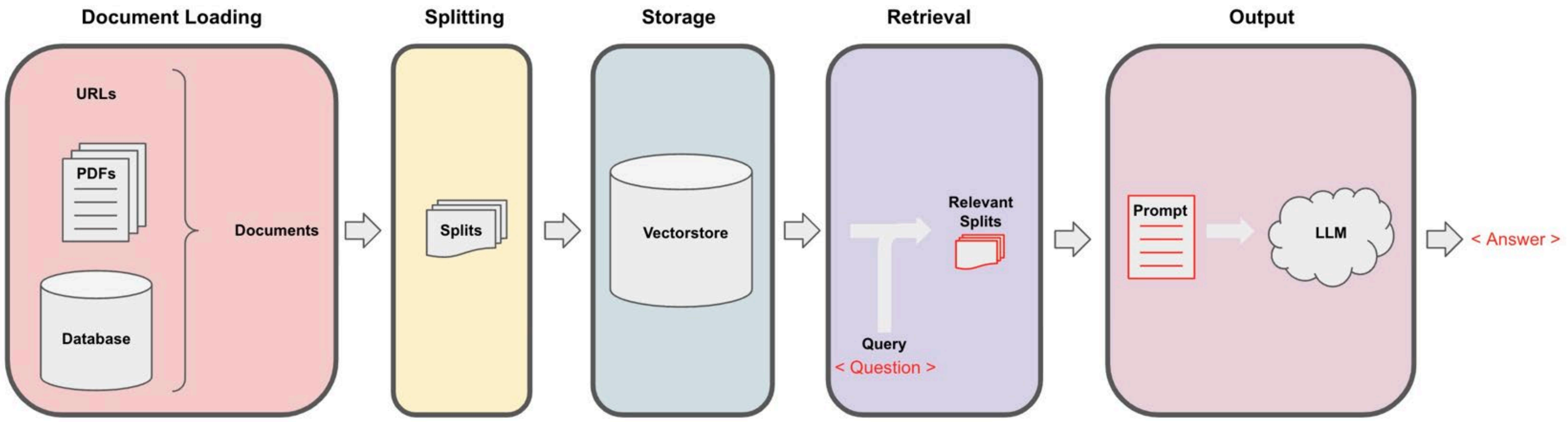


Question Answering on custom documents

- QA over structured data (e.g., SQL)
- QA over code (e.g., Python)



Langchain Pipeline for Question Answering



- The above subsystems can be viewed as independent modules that obey well defined interfaces
- This means that one can replace a module with another technique, so long as interfaces are respected

RAG Big Picture: Data Sources and Data formats

- We work with different data sources that arise from local file systems to web/cloud based resources.
- SaaS products, e-mails, Web Service APIs, Cloud databases etc are massive sources of data.
- Data can be unstructured (e.g. natural language texts in PDF documents) or structured (json data, excel tables storing numerical values), images, videos, audio, tables, source code, etc
- Data is represented in a variety of formats – pdfs, xlsx, pptx, py, json….
- Langchain doc loaders allow access to data using a common interface: Document class.

File Systems

Website URLs

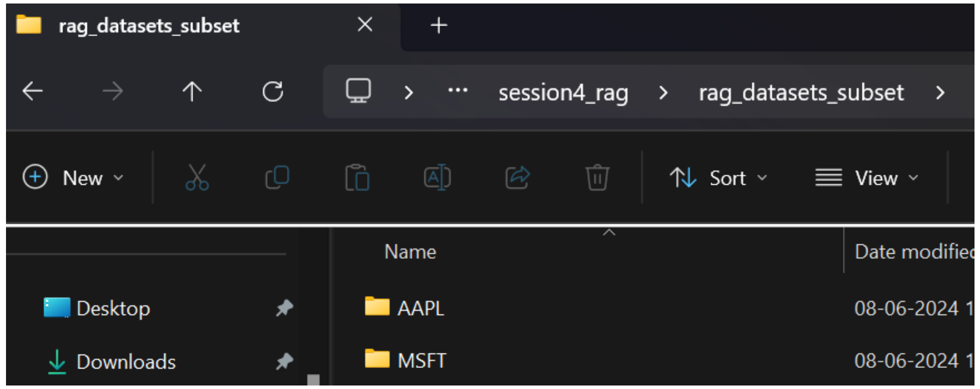
Web Service APIs

Custom Data Lakehouses

RAG Case Study #1 – Financial Analysis

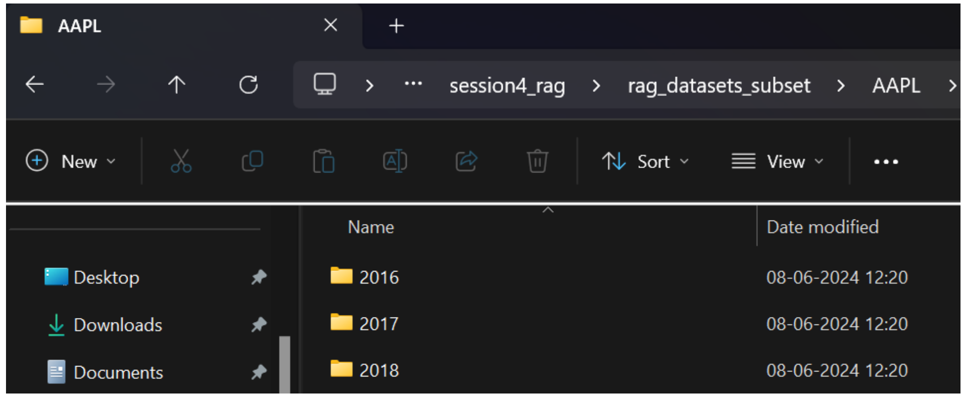
- We are given a dataset that has annual reports of Apple and Microsoft for 2018. Each annual report is split in to a number of PDF documents.
- We are required to build a Copilot Assistant, that takes a question from the user and provides the best possible answer.
- Extend this product to provide summarization.
- You can build an end to end product by including suitable UI and also graphics.
- With multimodal LLMs (e.g. Microsoft Phi-3V) it is possible to extend the problem to chat with charts and other visual components.

Dataset Organization



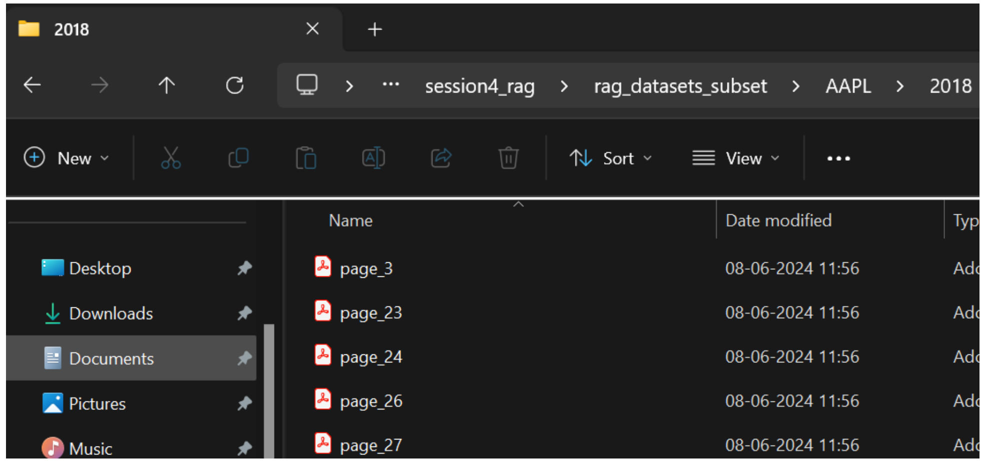
session4_rag > rag_datasets_subset >

Name	Date modifie
AAPL	08-06-2024 1
MSFT	08-06-2024 1



session4_rag > rag_datasets_subset > AAPL >

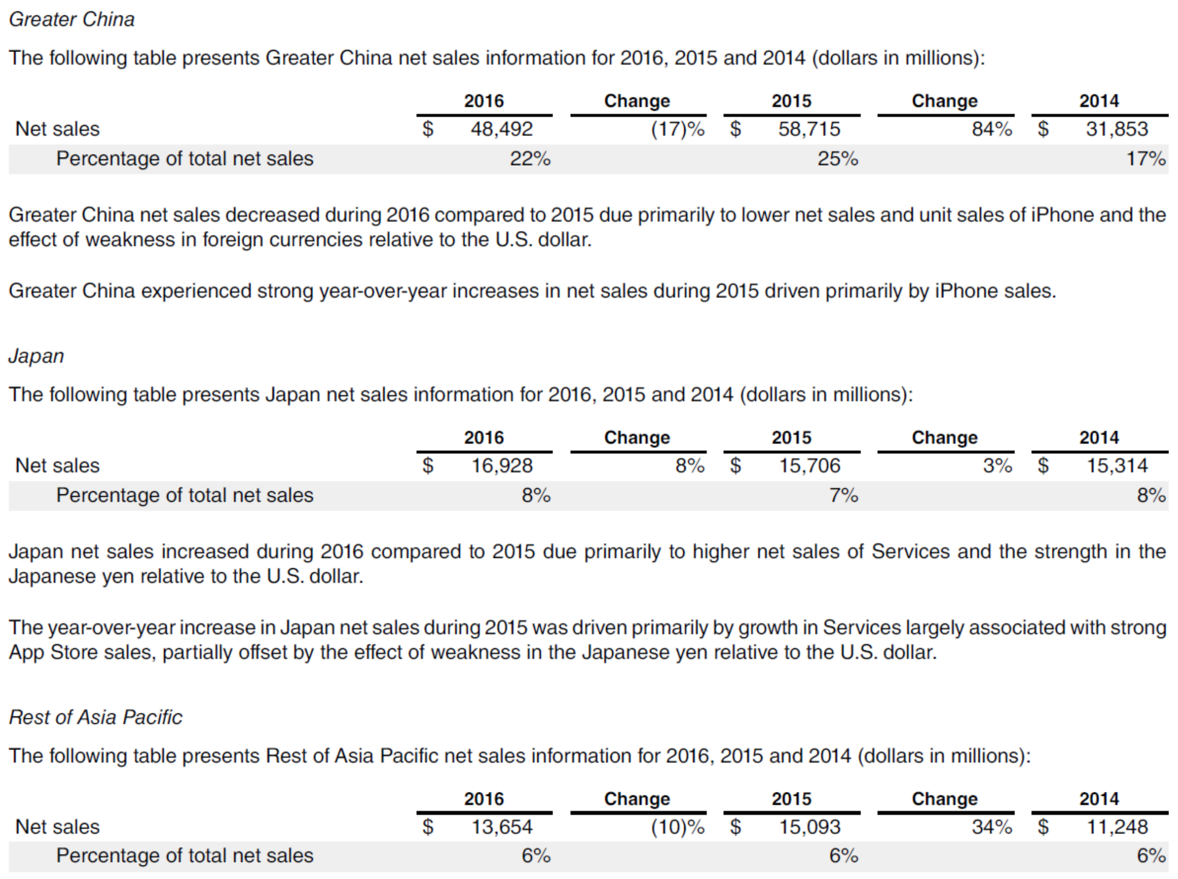
Name	Date modified
2016	08-06-2024 12:20
2017	08-06-2024 12:20
2018	08-06-2024 12:20



session4_rag > rag_datasets_subset > AAPL > 2018

Name	Date modified	Typ
page_3	08-06-2024 11:56	Ad
page_23	08-06-2024 11:56	Ad
page_24	08-06-2024 11:56	Ad
page_26	08-06-2024 11:56	Ad
page_27	08-06-2024 11:56	Ad

Sample page in PDF



Greater China

The following table presents Greater China net sales information for 2016, 2015 and 2014 (dollars in millions):

	2016	Change	2015	Change	2014
Net sales	$ 48,492	(17)%	$ 58,715	84%	$ 31,853
Percentage of total net sales	22%		25%		17%

Greater China net sales decreased during 2016 compared to 2015 due primarily to lower net sales and unit sales of iPhone and the effect of weakness in foreign currencies relative to the U.S. dollar.

Greater China experienced strong year-over-year increases in net sales during 2015 driven primarily by iPhone sales.

Japan

The following table presents Japan net sales information for 2016, 2015 and 2014 (dollars in millions):

	2016	Change	2015	Change	2014
Net sales	$ 16,928	8%	$ 15,706	3%	$ 15,314
Percentage of total net sales	8%		7%		8%

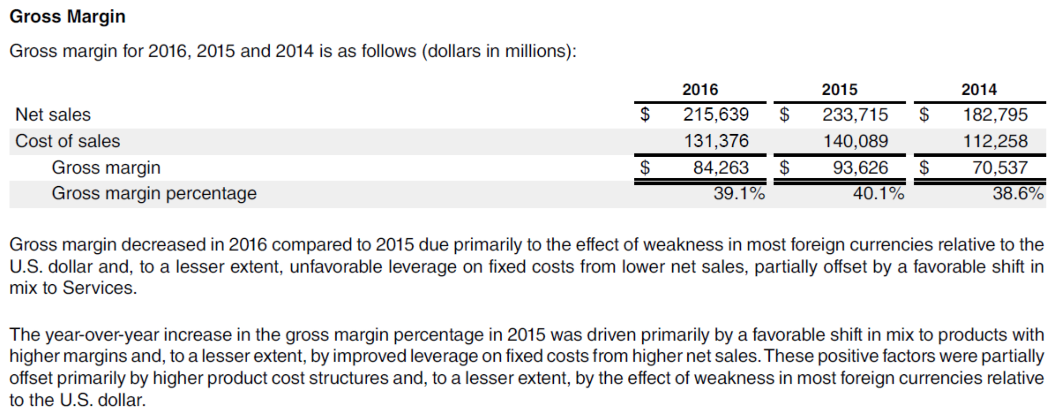
Japan net sales increased during 2016 compared to 2015 due primarily to higher net sales of Services and the strength in the Japanese yen relative to the U.S. dollar.

The year-over-year increase in Japan net sales during 2015 was driven primarily by growth in Services largely associated with strong App Store sales, partially offset by the effect of weakness in the Japanese yen relative to the U.S. dollar.

Rest of Asia Pacific

The following table presents Rest of Asia Pacific net sales information for 2016, 2015 and 2014 (dollars in millions):

	2016	Change	2015	Change	2014
Net sales	$ 13,654	(10)%	$ 15,093	34%	$ 11,248
Percentage of total net sales	6%		6%		6%



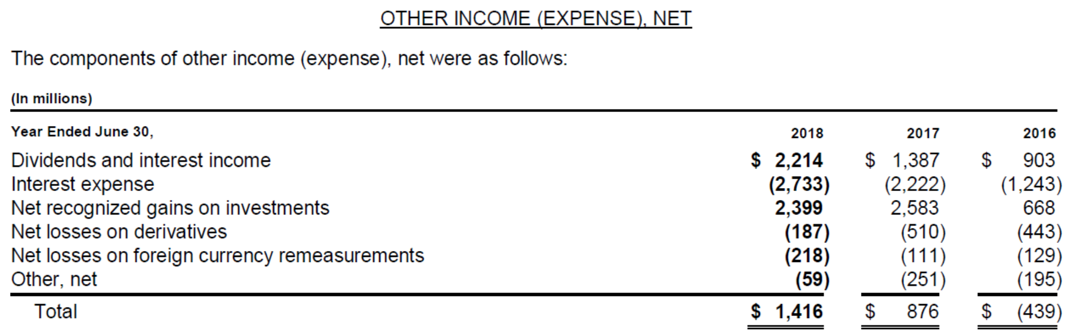
Gross Margin

Gross margin for 2016, 2015 and 2014 is as follows (dollars in millions):

	2016	2015	2014
Net sales	$ 215,639	$ 233,715	$ 182,795
Cost of sales	131,376	140,089	112,258
Gross margin	$ 84,263	$ 93,626	$ 70,537
Gross margin percentage	39.1%	40.1%	38.6%

Gross margin decreased in 2016 compared to 2015 due primarily to the effect of weakness in most foreign currencies relative to the U.S. dollar and, to a lesser extent, unfavorable leverage on fixed costs from lower net sales, partially offset by a favorable shift in mix to Services.

The year-over-year increase in the gross margin percentage in 2015 was driven primarily by a favorable shift in mix to products with higher margins and, to a lesser extent, by improved leverage on fixed costs from higher net sales. These positive factors were partially offset primarily by higher product cost structures and, to a lesser extent, by the effect of weakness in most foreign currencies relative to the U.S. dollar.



OTHER INCOME (EXPENSE), NET

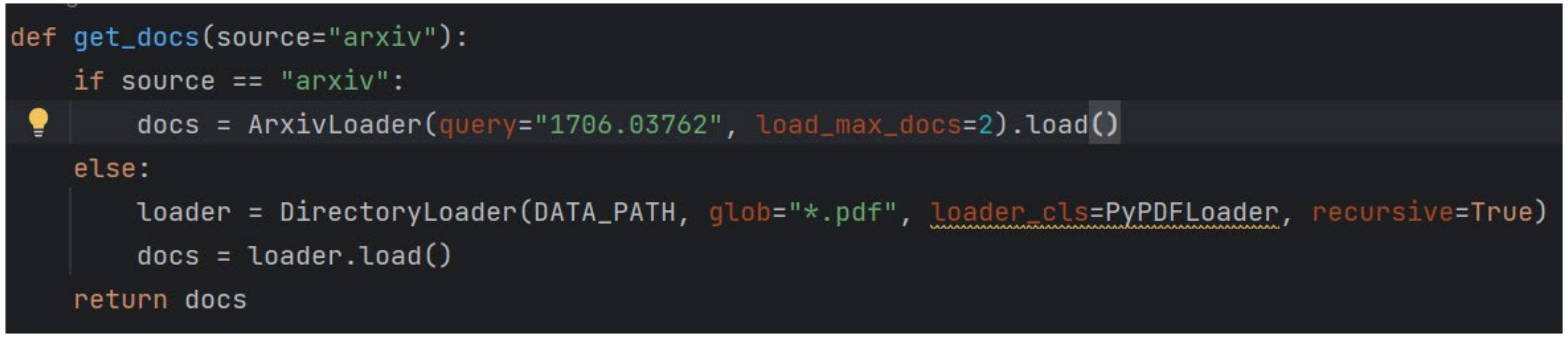
The components of other income (expense), net were as follows:

(In millions)

Year Ended June 30,	**2018**	2017	2016
Dividends and interest income	**$ 2,214**	$ 1,387	$ 903
Interest expense	**(2,733)**	(2,222)	(1,243)
Net recognized gains on investments	**2,399**	2,583	668
Net losses on derivatives	**(187)**	(510)	(443)
Net losses on foreign currency remeasurements	**(218)**	(111)	(129)
Other, net	**(59)**	(251)	(195)
Total	**$ 1,416**	$ 876	$ (439)

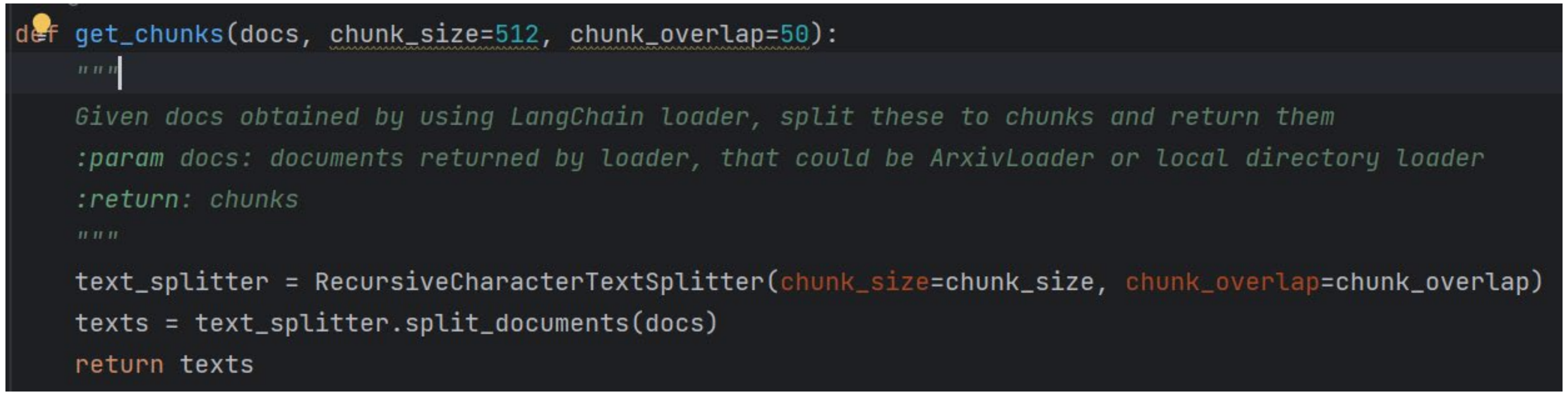
Naïve RAG Implementation: Ingesting

- Step#1: Load the required documents (In our case all docs are PDF)



```python
def get_docs(source="arxiv"):
    if source == "arxiv":
        docs = ArxivLoader(query="1706.03762", load_max_docs=2).load()
    else:
        loader = DirectoryLoader(DATA_PATH, glob="*.pdf", loader_cls=PyPDFLoader, recursive=True)
        docs = loader.load()
    return docs
```

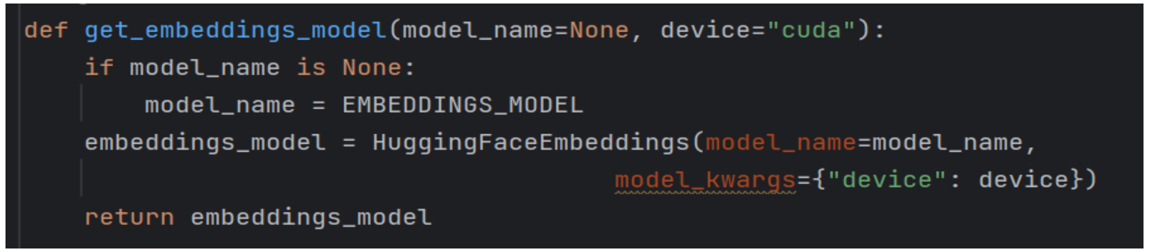
- Step#2: Split each document to multiple chunks



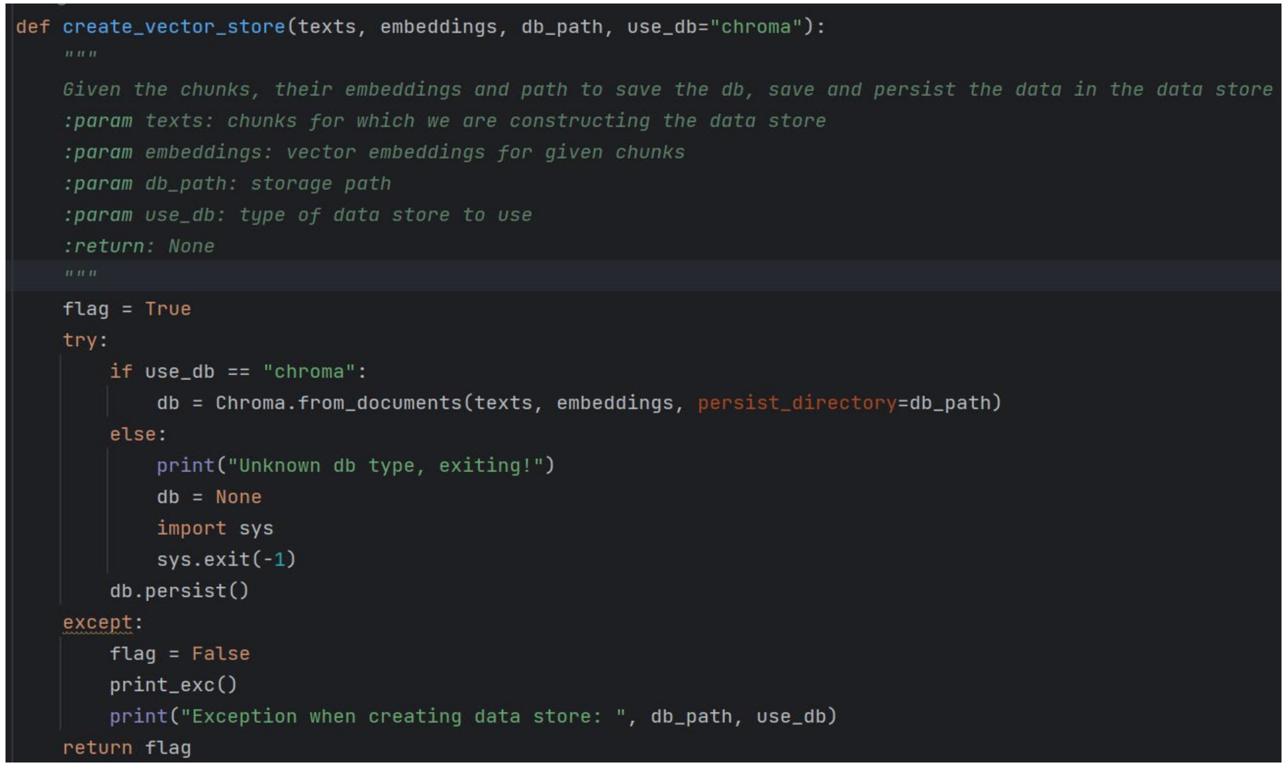
```python
def get_chunks(docs, chunk_size=512, chunk_overlap=50):
    """
    Given docs obtained by using LangChain loader, split these to chunks and return them
    :param docs: documents returned by loader, that could be ArxivLoader or local directory loader
    :return: chunks
    """
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_size, chunk_overlap=chunk_overlap)
    texts = text_splitter.split_documents(docs)
    return texts
```

Naïve RAG Implementation: Ingesting

- Step#3: Define an embedding model and embed each chunk
- Step#4: Embed and store each chunk in the vector store (ChromaDB)



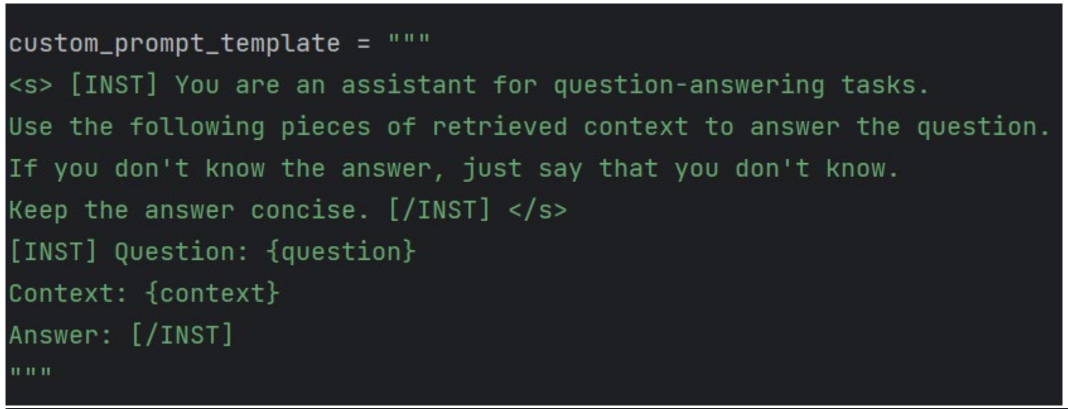
```python
def get_embeddings_model(model_name=None, device="cuda"):
    if model_name is None:
        model_name = EMBEDDINGS_MODEL
    embeddings_model = HuggingFaceEmbeddings(model_name=model_name,
                                             model_kwargs={"device": device})
    return embeddings_model
```



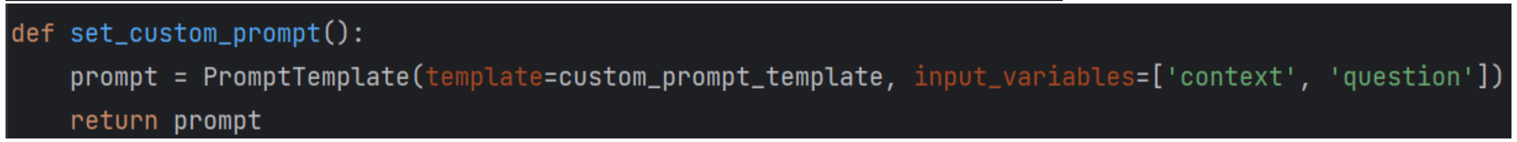
```python
def create_vector_store(texts, embeddings, db_path, use_db="chroma"):
    """
    Given the chunks, their embeddings and path to save the db, save and persist the data in the data store
    :param texts: chunks for which we are constructing the data store
    :param embeddings: vector embeddings for given chunks
    :param db_path: storage path
    :param use_db: type of data store to use
    :return: None
    """
    flag = True
    try:
        if use_db == "chroma":
            db = Chroma.from_documents(texts, embeddings, persist_directory=db_path)
        else:
            print("Unknown db type, exiting!")
            db = None
            import sys
            sys.exit(-1)
        db.persist()
    except:
        flag = False
        print_exc()
        print("Exception when creating data store: ", db_path, use_db)
    return flag
```

Naïve RAG Implementation: Model Creation

- Define required prompt

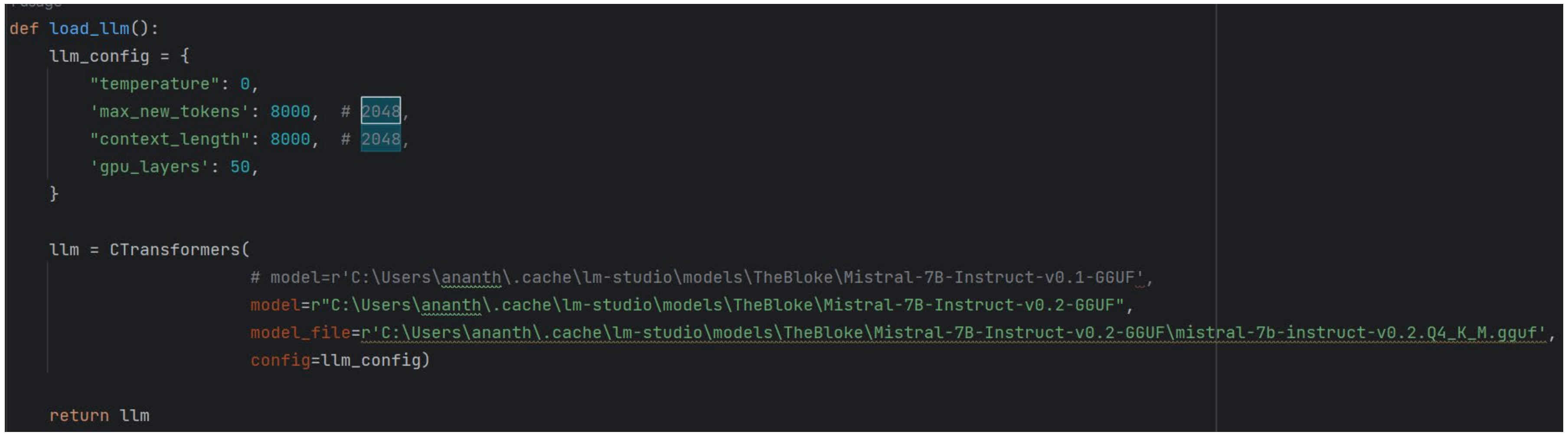


```
custom_prompt_template = """
<s> [INST] You are an assistant for question-answering tasks.
Use the following pieces of retrieved context to answer the question.
If you don't know the answer, just say that you don't know.
Keep the answer concise. [/INST] </s>
[INST] Question: {question}
Context: {context}
Answer: [/INST]
"""
```



```
def set_custom_prompt():
    prompt = PromptTemplate(template=custom_prompt_template, input_variables=['context', 'question'])
    return prompt
```

- Load LLM



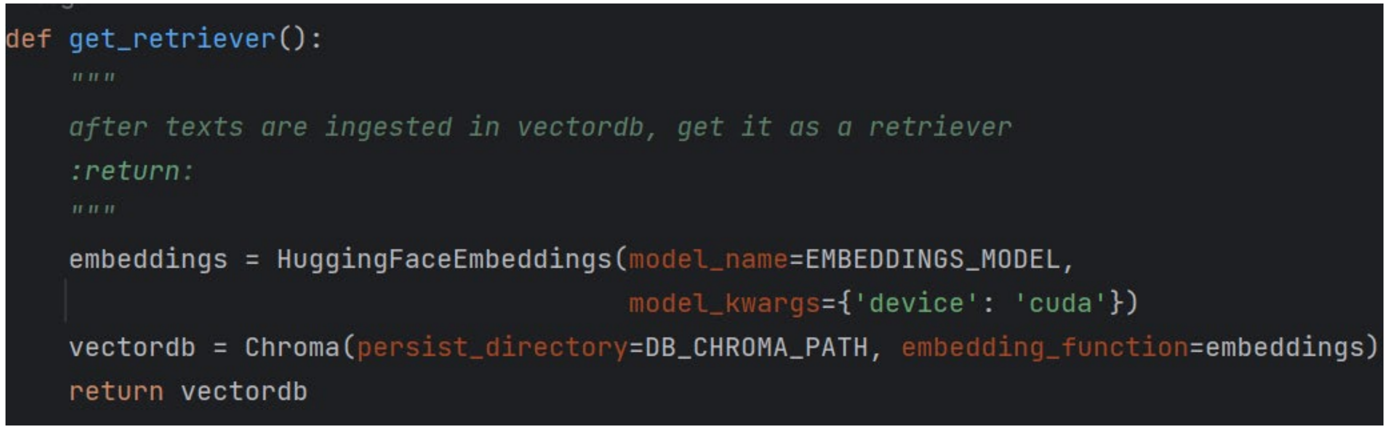
```
def load_llm():
    llm_config = {
        "temperature": 0,
        'max_new_tokens': 8000,  # 2048,
        "context_length": 8000,  # 2048,
        'gpu_layers': 50,
    }

    llm = CTransformers(
                        # model=r'C:\Users\ananth\.cache\lm-studio\models\TheBloke\Mistral-7B-Instruct-v0.1-GGUF',
                        model=r"C:\Users\ananth\.cache\lm-studio\models\TheBloke\Mistral-7B-Instruct-v0.2-GGUF",
                        model_file=r'C:\Users\ananth\.cache\lm-studio\models\TheBloke\Mistral-7B-Instruct-v0.2-GGUF\mistral-7b-instruct-v0.2.Q4_K_M.gguf',
                        config=llm_config)

    return llm
```

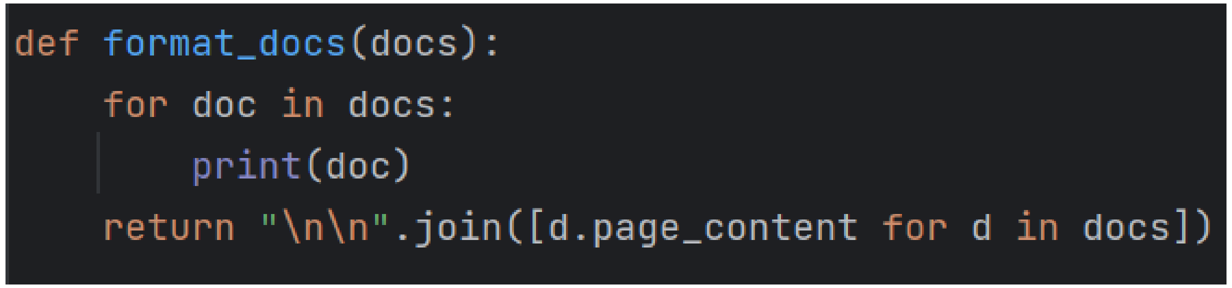
Naïve RAG Implementation: Model Creation

- Create a retriever (retrieves relevant docs from Vector Store, based on the query)



```python
def get_retriever():
    """
    after texts are ingested in vectordb, get it as a retriever
    :return:
    """
    embeddings = HuggingFaceEmbeddings(model_name=EMBEDDINGS_MODEL,
                                       model_kwargs={'device': 'cuda'})
    vectordb = Chroma(persist_directory=DB_CHROMA_PATH, embedding_function=embeddings)
    return vectordb
```

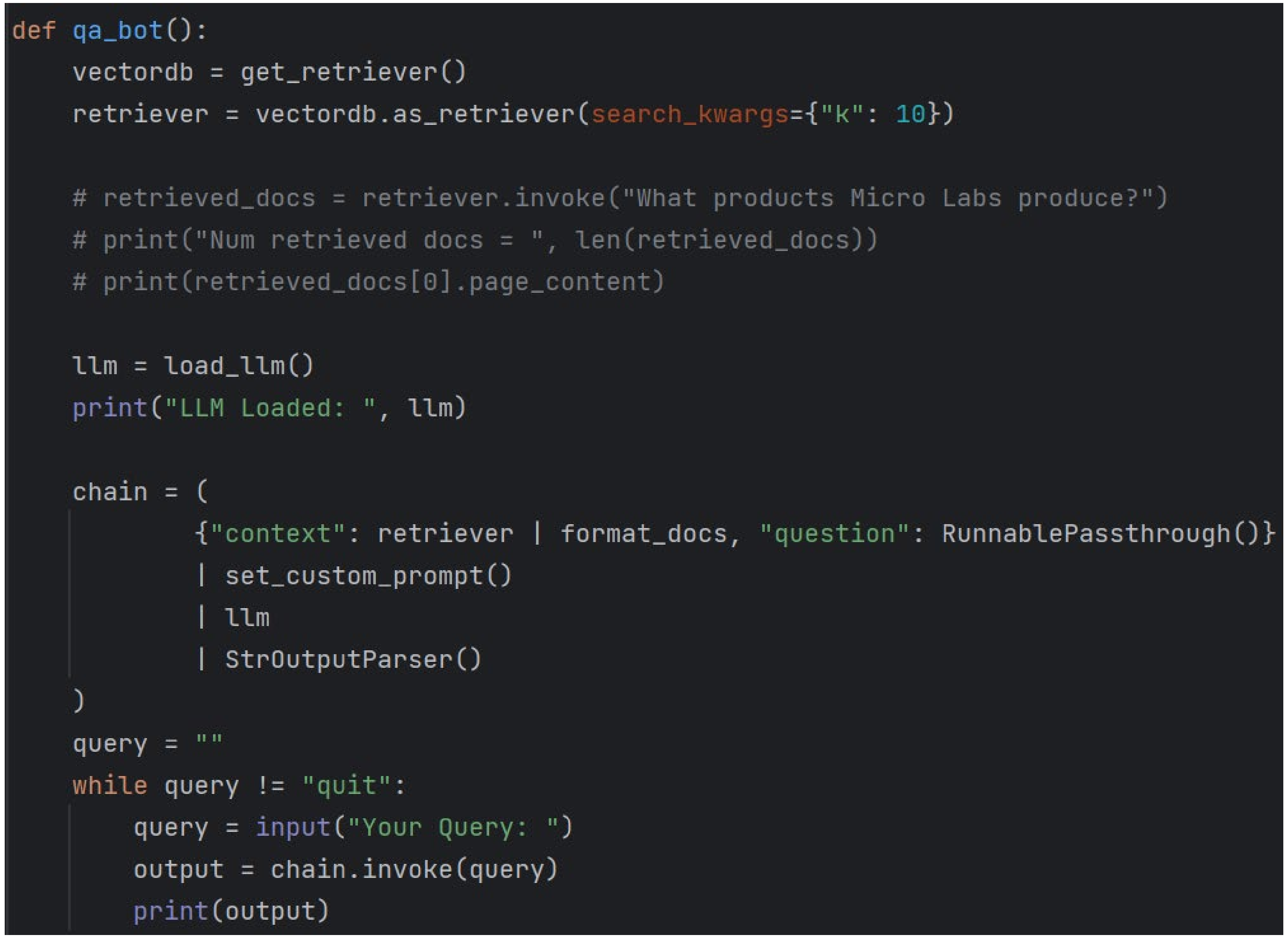
- Create a document formatter (formats retrieved docs in to a form suitable for LLM to process)



```python
def format_docs(docs):
    for doc in docs:
        print(doc)
    return "\n\n".join([d.page_content for d in docs])
```

Naïve RAG Implementation: Model

- Build the qa_bot



```python
def qa_bot():
    vectordb = get_retriever()
    retriever = vectordb.as_retriever(search_kwargs={"k": 10})

    # retrieved_docs = retriever.invoke("What products Micro Labs produce?")
    # print("Num retrieved docs = ", len(retrieved_docs))
    # print(retrieved_docs[0].page_content)

    llm = load_llm()
    print("LLM Loaded: ", llm)

    chain = (
            {"context": retriever | format_docs, "question": RunnablePassthrough()}
            | set_custom_prompt()
            | llm
            | StrOutputParser()
    )
    query = ""
    while query != "quit":
        query = input("Your Query: ")
        output = chain.invoke(query)
        print(output)
```

Demo: Financial Analysis

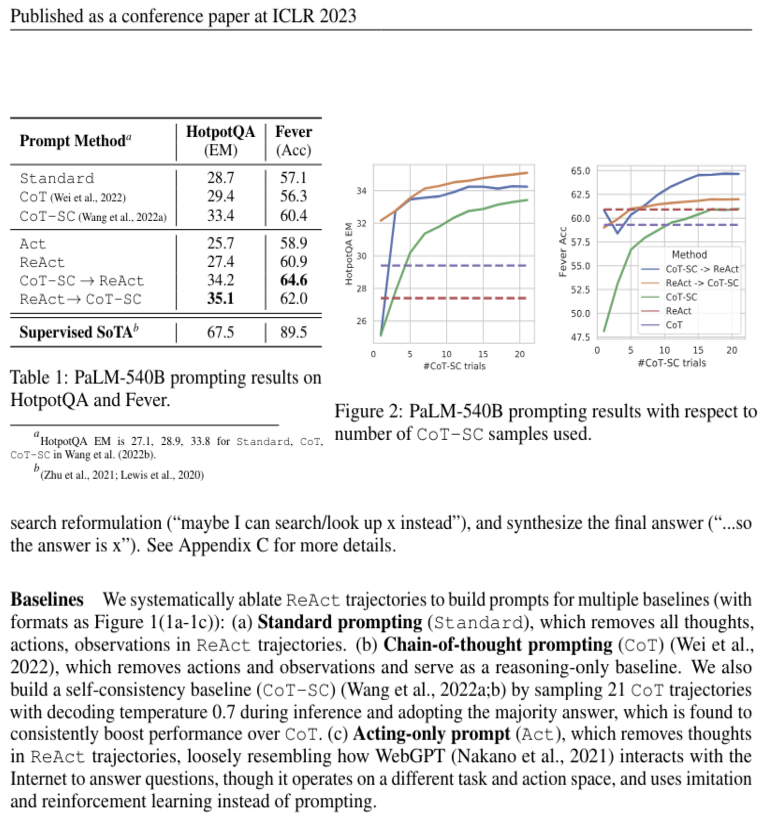
- We will build a product that allows a business strategist to analyze financial results of our company along with our key competitors and formulate our strategy.
  - How does our competitor grow over last 3 years?
  - What are their new investments? New products?
  - What acquisitions they have made?
  - Are they expanding their operations in different geos?
  - What are their retention levels?
  - Can we visualize these with a finance dashboard?

Some challenges

- If the matches are a large number of chunks, say 50 and we have set k = 20, we may miss vital information.
- Suppose we set k=100, then there is a possibility of overflowing the LLM context.
- Chunk size and overlap settings can affect the accuracy
- Suppose all chunks returned has only Apple’s documents and no Microsoft, we can’t compare them.
- When the number of source documents are very large and if the query requires answers contained in each document, one may have to deal with a large number of chunks.

Splitting/Chunking

- PDF Documents may contain images and tables besides text data.
- Chunking them with character or recursive character splitter is not enough.
- Similarly, splitting source code (e.g. Python, JS and so on) requires an understanding of the programming language.
- In such cases, a sophisticated semantic approach to chunking is required.
- Refer: [RetrievalTutorials/tutorials/LevelsOfTextSplitting/5_Levels_Of_Text_Splitting.ipynb at main · FullStackRetrieval-com/RetrievalTutorials · GitHub]



Published as a conference paper at ICLR 2023

Prompt Method[a]	HotpotQA (EM)	Fever (Acc)
Standard	28.7	57.1
CoT (Wei et al., 2022)	29.4	56.3
CoT-SC (Wang et al., 2022a)	33.4	60.4
Act	25.7	58.9
ReAct	27.4	60.9
CoT-SC → ReAct	34.2	**64.6**
ReAct→ CoT-SC	**35.1**	62.0
Supervised SoTA[b]	67.5	89.5

Table 1: PaLM-540B prompting results on HotpotQA and Fever.

[a] HotpotQA EM is 27.1, 28.9, 33.8 for Standard, CoT, CoT-SC in Wang et al. (2022b).
[b] (Zhu et al., 2021; Lewis et al., 2020)

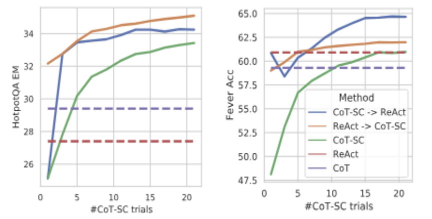


Figure 2: PaLM-540B prompting results with respect to number of CoT-SC samples used.

search reformulation ("maybe I can search/look up x instead"), and synthesize the final answer ("...so the answer is x"). See Appendix C for more details.

Baselines We systematically ablate ReAct trajectories to build prompts for multiple baselines (with formats as Figure 1(1a-1c)): (a) **Standard prompting** (Standard), which removes all thoughts, actions, observations in ReAct trajectories. (b) **Chain-of-thought prompting** (CoT) (Wei et al., 2022), which removes actions and observations and serve as a reasoning-only baseline. We also build a self-consistency baseline (CoT-SC) (Wang et al., 2022a;b) by sampling 21 CoT trajectories with decoding temperature 0.7 during inference and adopting the majority answer, which is found to consistently boost performance over CoT. (c) **Acting-only prompt** (Act), which removes thoughts in ReAct trajectories, loosely resembling how WebGPT (Nakano et al., 2021) interacts with the Internet to answer questions, though it operates on a different task and action space, and uses imitation and reinforcement learning instead of prompting.

Chunk Visualization Demo

- Refer: https://huggingface.co/spaces/m-ric/chunk_visualizer