



Hey Everyone! My name is...

LANGCHAIN

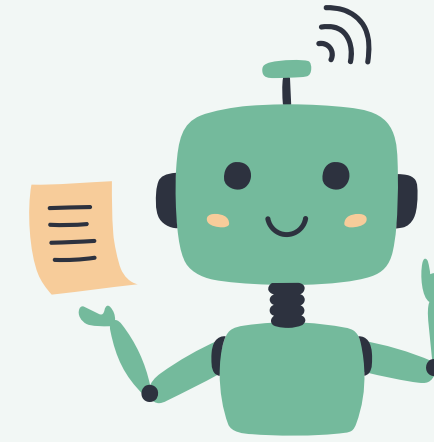
GROUP ONE
@AIAP BATCH THIRTEEN





**LangChain is a framework
designed to enhance the
usage & integration of LLMs
in various applications**

PROBLEM STATEMENT: WHY DO I EXIST?



1

**BOILERPLATE
TEXT IN PROMPTS**

2

**UNSTRUCTURED
RESPONSES**

3

**LLM ECOSYSTEM
SWITCHING**

4

**SHORT MEMORY OF
LLMS**

5

**INTEGRATING INTO
PIPELINES**

6

**PASSING DATA TO
LLMS**

1

LLMS UTILITIES

Provision of common interface and utilities for all LLMs.

2

PROMPT MANAGEMENT

Provision of prompt template function to set up repetitive, boilerplate text prompts.

3

MEMORY TYPES

Contains a collection of memory implementations

HOW CAN I HELP?



4

INDEXES

Supports users' need to retrieve, transform, store and query documents.

5

CHAINS

Provision of standard interface for a sequence of calls to integrate LLMs with other tools.

6

AGENTS

Provision of a standard interface for agents, which helps LLM makes decision on actions to be taken.

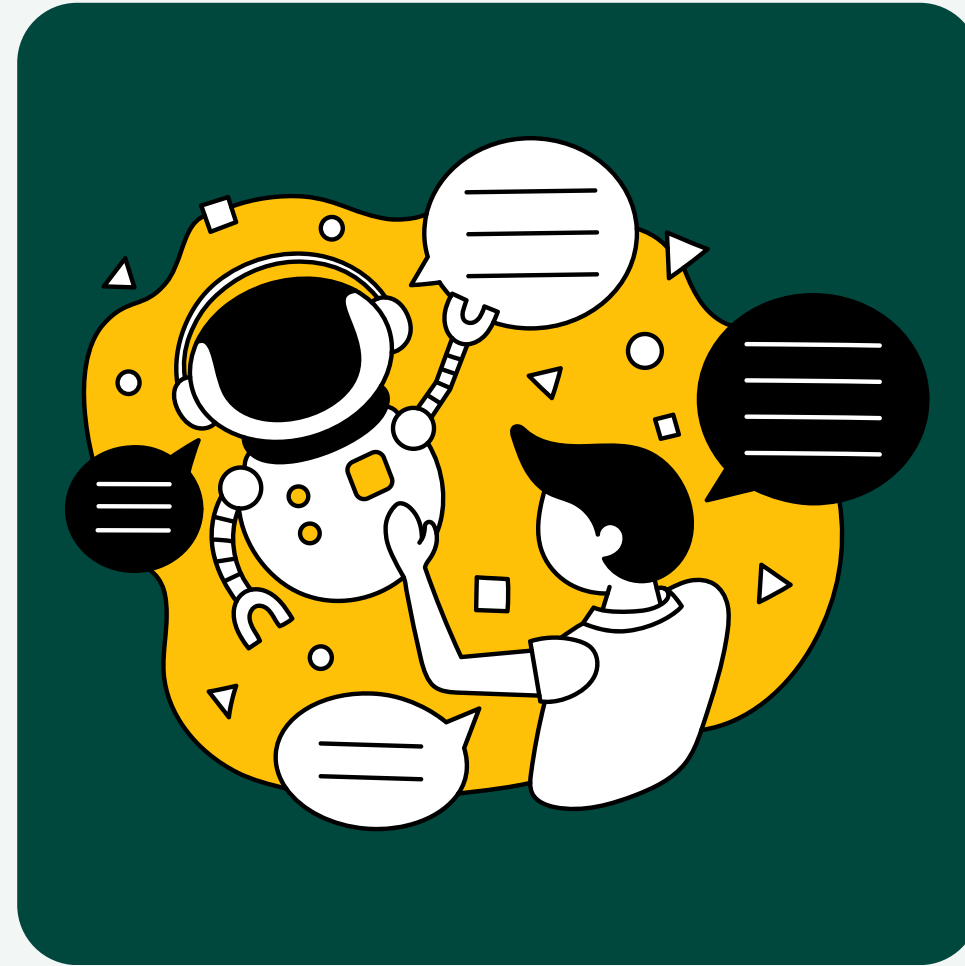
HOW CAN I HELP?



USE CASES



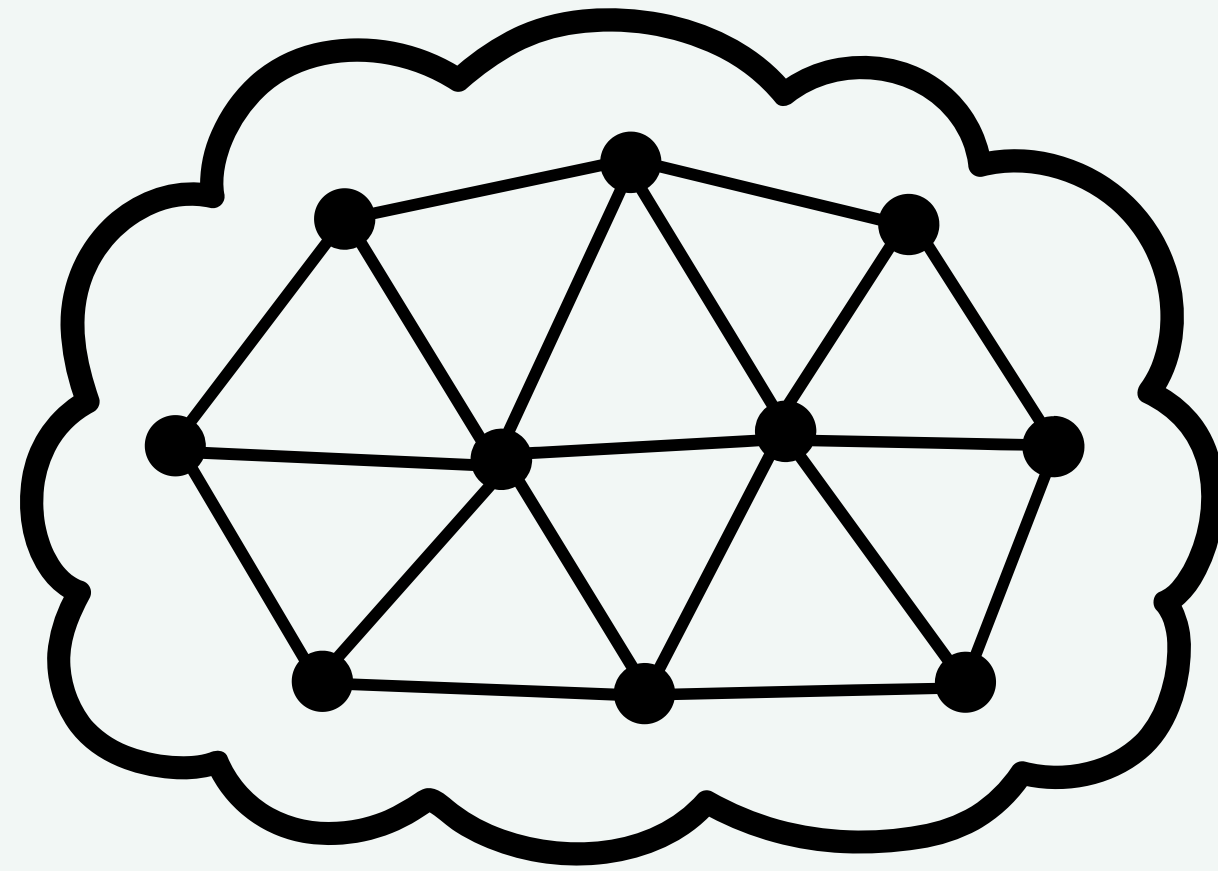
**QUERYING
DATASETS WITH
NATURAL LANGUAGE**



**INTERACTING WITH
APIS**



**BUILDING A
CHATBOT**



LangChain Components

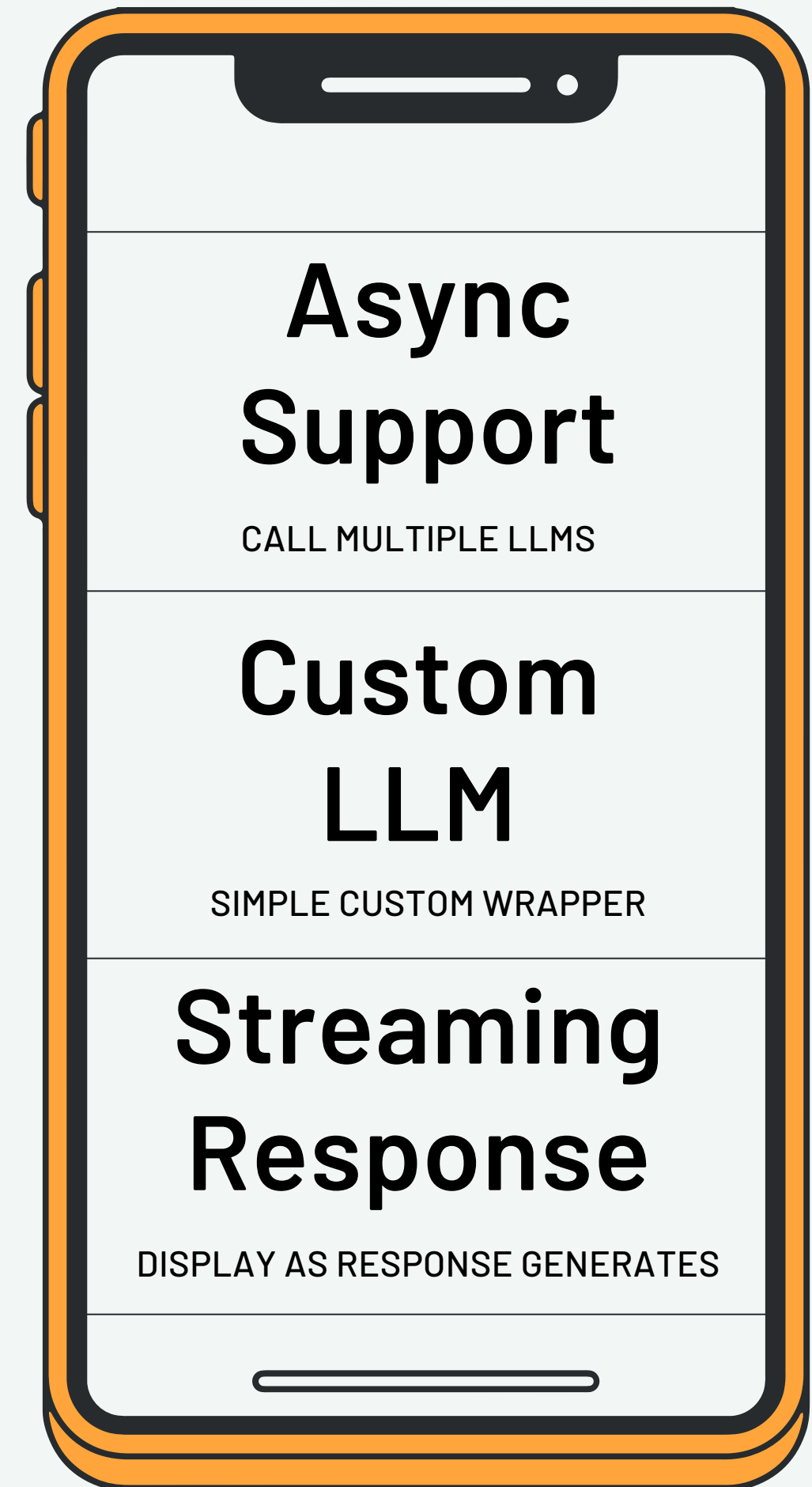
LLMS UTILITIES

A Standard interface with sub-modules to interact with a variety of LLMs from various LLM providers

Async Support: Call multiple LLMs concurrently

Custom LLM: For users to customise a simple wrapper to use their own LLM

Streaming Response: Process or display response as it is being generated



PROMPT TEMPLATES

Since much of the boilerplate text in prompts would be identical to the next, it would be ideal to just **write it once**

LangChain solves this problem by providing several classes and functions to make constructing and working with prompts easy

```
from langchain import PromptTemplate

template = """
I want you to act as a naming consultant for new
companies.
What is a good name for a company that makes
{product}?
"""

prompt = PromptTemplate(
    input_variables=["product"],
    template=template,
)

prompt.format(product="colorful socks")
```

```
# -> I want you to act as a naming consultant for new
companies.
# -> What is a good name for a company that makes
colorful socks?
```

MEMORY TYPES

LLM in itself is stateless. However, its response is highly dependent on its previous conversation with the user

ConversationBufferMemory:

Stores all interactions

ConversationBufferWindowMemory:

Only store last k interaction(s)

Input:

"Which data source types could be used to give context to the model?"

History:

Human: Good morning AI!

AI: Good morning! How can I help you?

Human: I'd like to explore the potential of integrating LLM with external knowledge

AI: Interesting! Integrating LLM with external knowledge can help them better understand the context and generate more accurate results.

Human: Which data source types could be used to give context to the model?

Output:

"There are a variety of data sources that..."

MEMORY TYPES

LLM in itself is stateless. However, its response is highly dependent on its previous conversation with the user

ConversationSummaryMemory:

Stores all interactions in a summarised form

Input:

"Which data source types could be used to give context to the model?"

History:

"The human greeted the AI with a good morning, to which the AI responded with a good morning and asked how it could help. The human expressed interest in exploring the potential of integrating Large Language Models with external knowledge, to which the AI responded positively and asked for more information."

Output:

"There are a variety of data sources that..."

MEMORY TYPES

LLM in itself is stateless. However, its response is highly dependent on its previous conversation with the user

ConversationKnowledgeGraphMemory:

Recognises entities and stores information about the entities

Input:

"My name is human and I like coffee!"

History:

""

Output:

"Hi Human! It's nice to meet you..."

Input:

"I also like tea! What do you like?"

History:

""

Output:

"I like a variety of beverages, but..."

conversation.memory.kg.get_triples()

[('Human', 'coffee', 'likes'), ('Human', 'tea', 'likes')]

4 main elements
when working with
indexes

Document Loaders

LOAD DOCS FROM
VARIOUS SOURCES

Text Splitters

SPLIT LARGE TEXT INTO
SMALLER CHUNKS

Vector Stores

CREATE AND STORE
EMBEDDINGS FOR EACH DOC

Retrievers

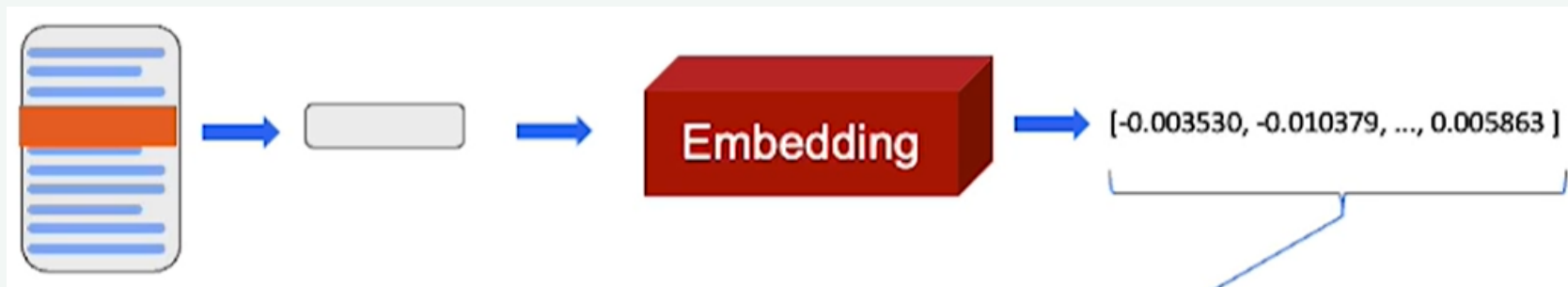
FETCH RELEVANT DOCS
TO COMBINE WITH LLMS

INDEXES

**Structures documents so that LLMs can
best interact with them**

Common Use Case - Question & Answering

For question answering over many documents, it is almost always best to create an index over the data. This can then be used to access the most relevant documents for a given question without having to pass all the documents to the LLM, saving both time and money.



Embeddings

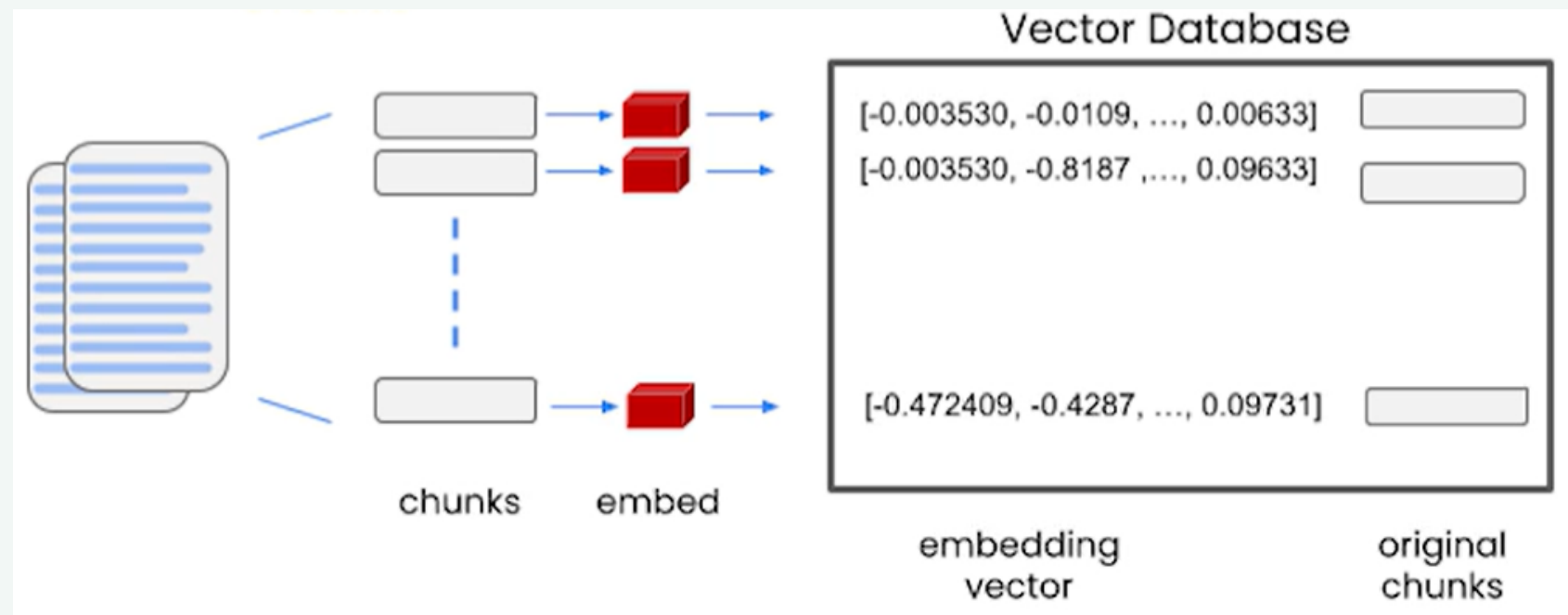
Embeddings are vector representations of words or phrases that capture their meaning and context. Text with similar content will have similar vectors. Knowing which pieces of text are similar will be useful in figuring out which pieces of text to include when passing to the LLM to answer the question.

INDEXES

Why use indexes?

A problem faced when querying a LLM on a large document set is that LLM's can only inspect a few thousand words at a time. This is where embeddings and VectorStores come in.





Creating Index

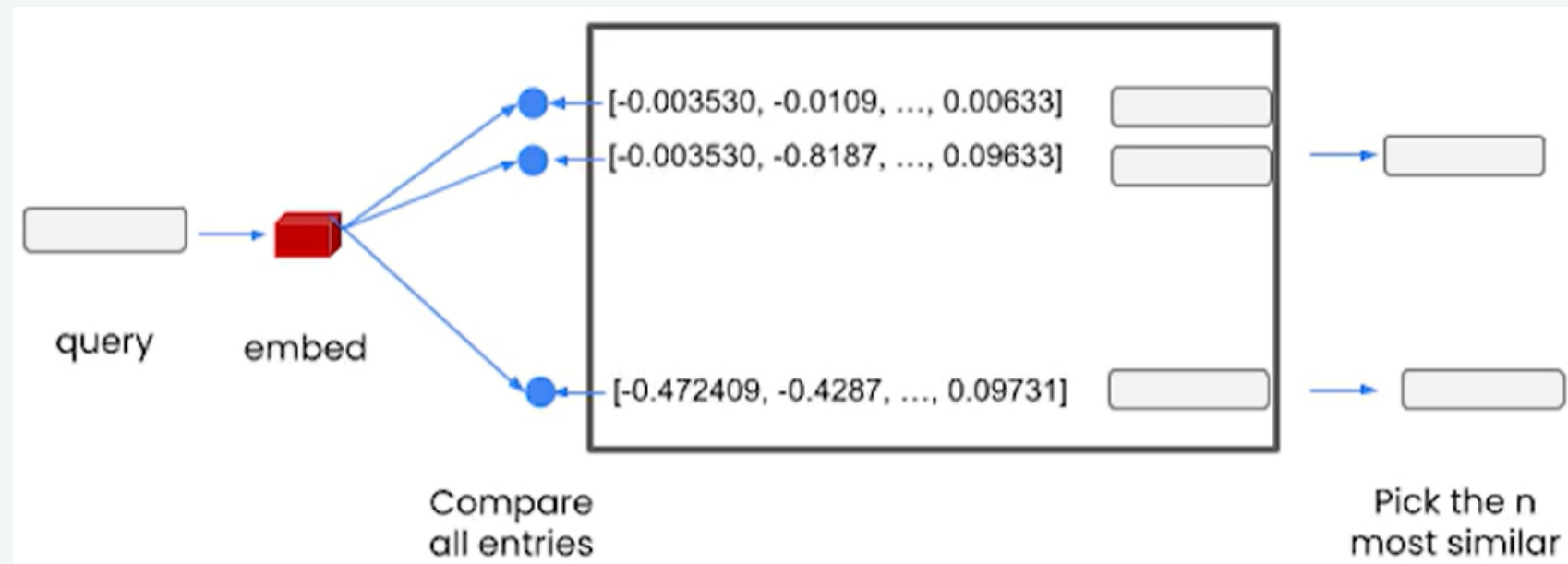
As the LLM may not be able to take in a whole document, the document is broken into smaller **chunks** so that only the most relevant ones are passed to the LLM. An **embedding** for each of the chunks is created and then stored in a **Vector Database**. This is how the **Index** is created.

INDEXES

Why use indexes?

A problem faced when querying a LLM on a large document set is that LLM's can only inspect a few thousand words at a time. This is where embeddings and VectorStores come in.





Finding Similar Vectors

The **Index** can now be used during runtime to find the pieces of text most relevant to an incoming query. An embedding for the query is first created and then compared to all the vectors in the database. The ***n*** most similar vectors are then picked and returned.

INDEXES

Why use indexes?

A problem faced when querying a LLM on a large document set is that LLM's can only inspect a few thousand words at a time. This is where embeddings and VectorStores come in.





Returning the Final Answer

The returned values can now fit in the LLM context. They are then passed, using the prompt, to the LLM for processing to get back the final answer.

INDEXES

Why use indexes?

A problem faced when querying a LLM on a large document set is that LLM's can only inspect a few thousand words at a time. This is where embeddings and VectorStores come in.



CHAINS

An end-to-end wrapper that allows us to combine multiple components in a particular way to accomplish a common use case

Composing components together in a chain is simple yet powerful. It drastically simplifies and makes the implementation of complex applications more modular, making it easier to debug, maintain, and improve your applications.

LLMChain

A SIMPLE CHAIN THAT ADDS
SOME FUNCTIONALITY AROUND
LANGUAGE MODELS

Document Chains (Index-related chains)

USED FOR INTERACTING WITH
INDEXES SO THAT YOU CAN
COMBINE YOUR OWN DATA
(STORED IN THE INDEXES) WITH
LLMS

CHAINS

LLMChain

Components:

- Prompt Template
- Model (LLM or ChatModel)
- Output Parser (Optional)

This chain takes in user input, uses the PromptTemplate to format the input key values provided (and memory key values if available) into a prompt and passes it to the LLM. The output from the LLM will be passed to the OutputParser (if provided) to parse the output into a final format.

```
from langchain import PromptTemplate, OpenAI, LLMChain

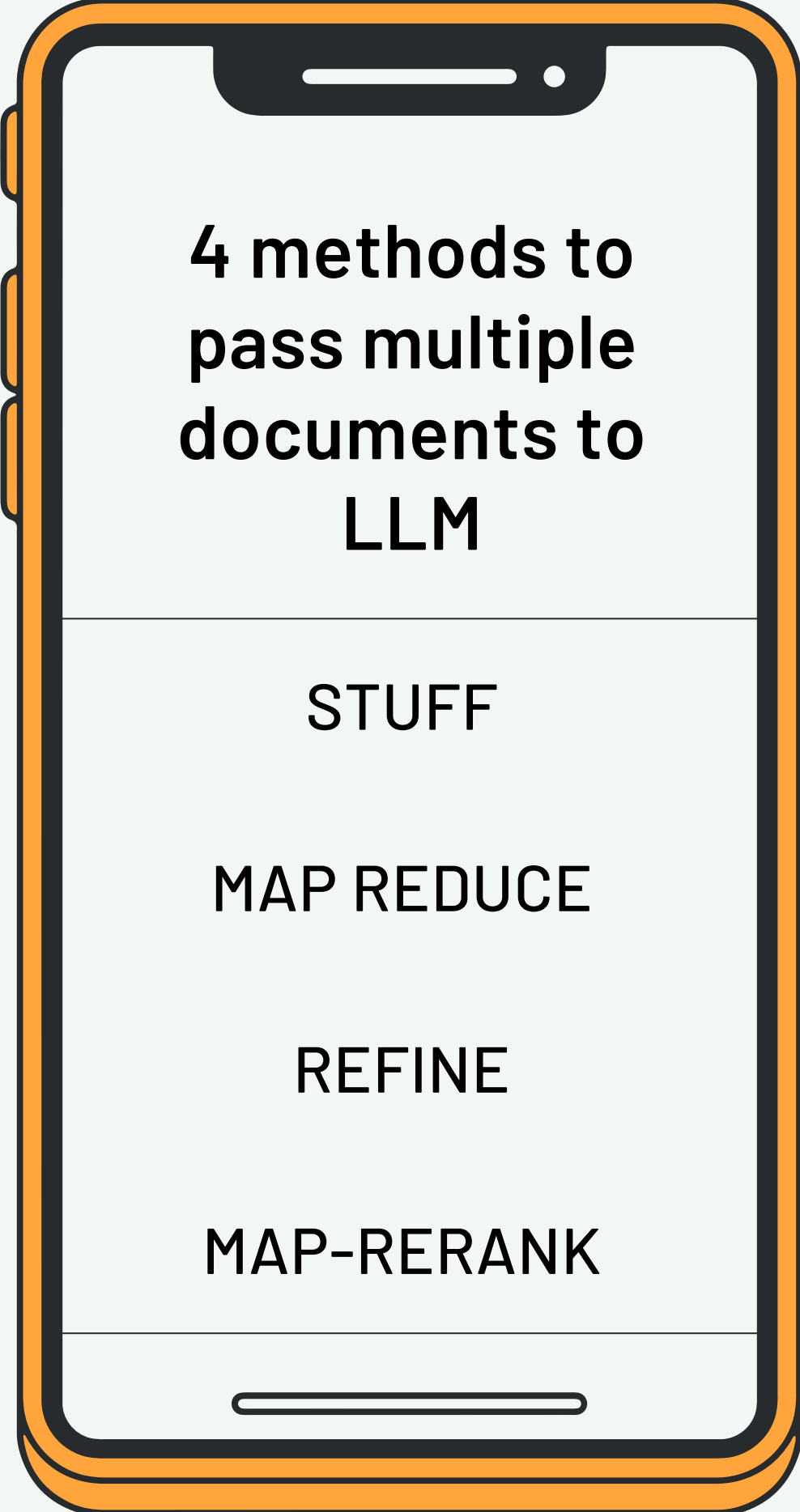
prompt_template = "What is a good name for a company that  
makes {product}?"

llm = OpenAI(temperature=0)
llm_chain = LLMChain(
    llm=llm,
    prompt=PromptTemplate.from_template(prompt_template)
)
llm_chain("colorful socks")
```

CHAINS

Document Chains (Index-related chains)

These are the core chains for working with Documents. They are useful for summarizing documents, answering questions over documents, extracting information from documents, and more.



**4 methods to
pass multiple
documents to
LLM**

STUFF

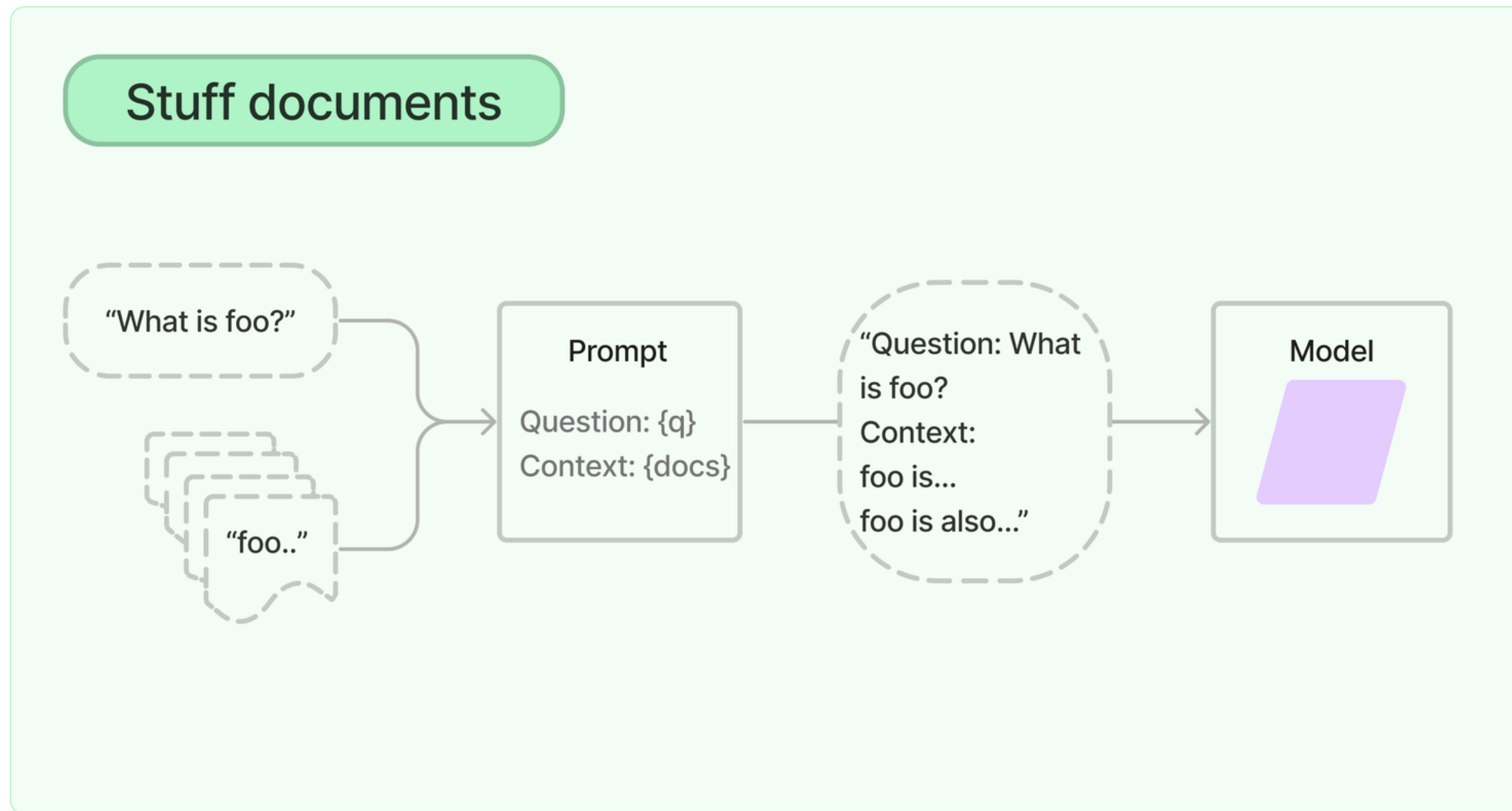
MAP REDUCE

REFINE

MAP-RERANK

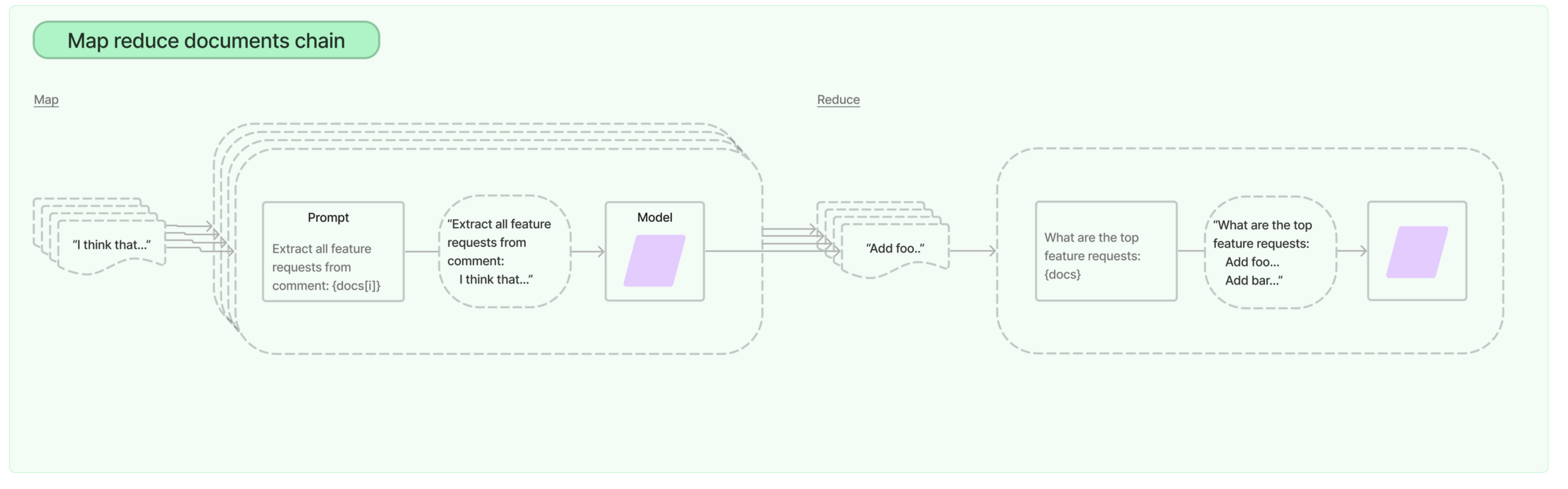
CHAINS

STUFF



CHAINS

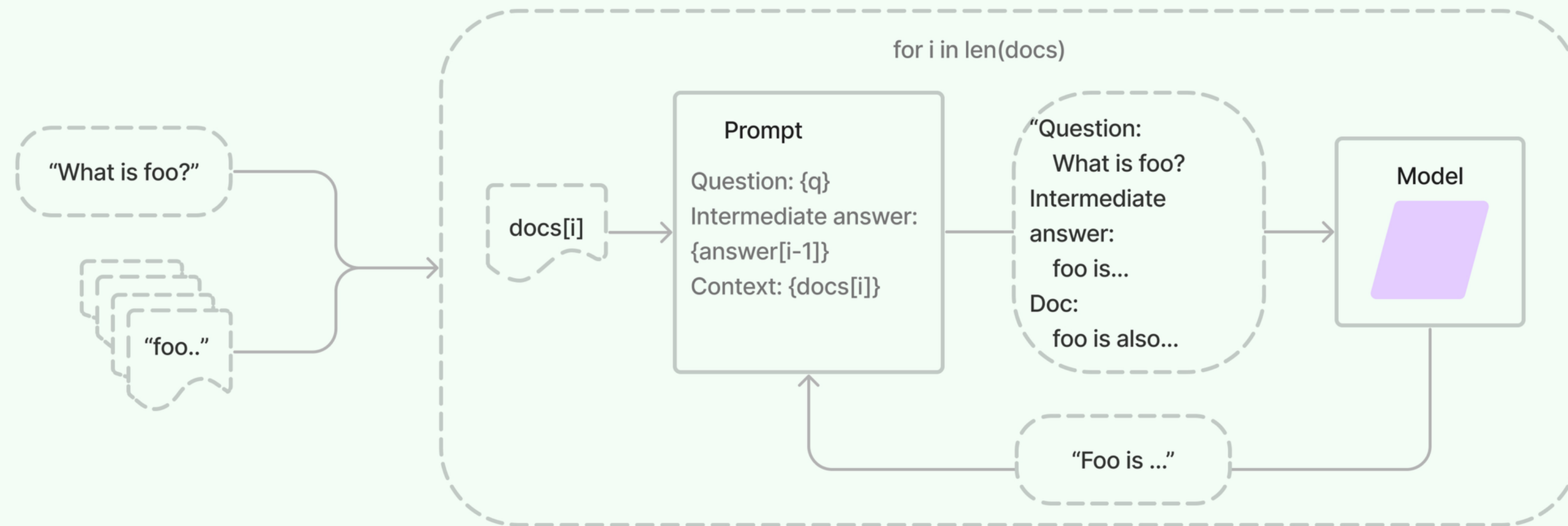
MAP REDUCE



CHAINS

REFINE

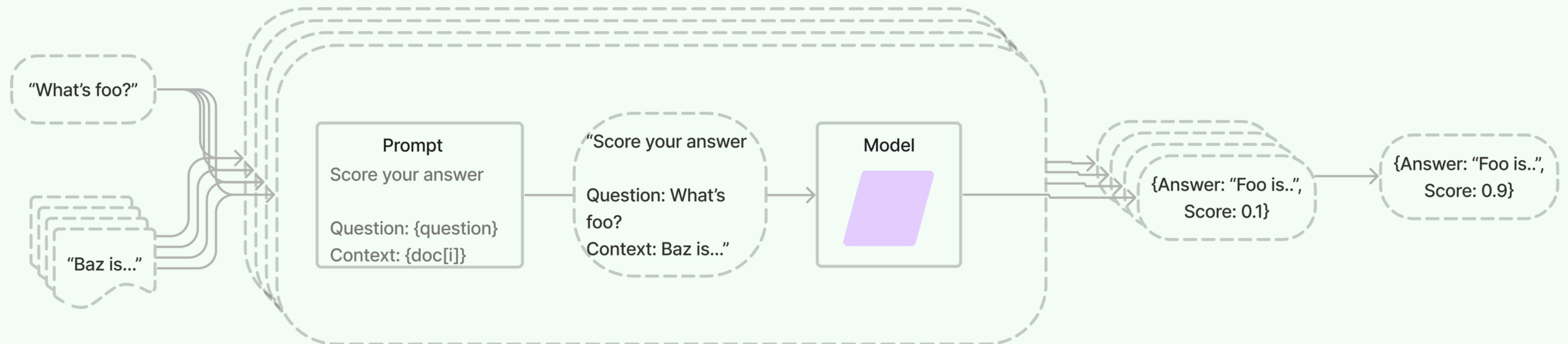
Refine documents chain



CHAINS

MAP RE-RANK

Map re-rank documents chain



CHAINS

PROS & CONS

Method	Pros	Cons
Stuff	<ul style="list-style-type: none">• Only makes a single call to LLM• LLM has access to all data at once when generating text	<ul style="list-style-type: none">• Most LLMs have a context length limit• Not workable for large / many documents
Map Reduce	<ul style="list-style-type: none">• Can scale to larger /more docs than Stuff• Calls to the LLM on individual docs are independent and can be parallelized• Good for summarization tasks	<ul style="list-style-type: none">• Requires many more calls to the LLM than Stuffing• Loses some information during the final combined call as individual documents are treated independently
Refine	<ul style="list-style-type: none">• Can pull in more relevant context since individual doc are not treated independently• Less loss of information when compared to Map Reduce method	<ul style="list-style-type: none">• Requires many more calls to the LLM than Stuffing• Calls are not independent - cannot be processed in parallel• Dependent on the ordering of the docs
Map-Rerank	<ul style="list-style-type: none">• Similar pros as Map Reduce method.• Requires fewer calls, compared to Map Reduce method	<ul style="list-style-type: none">• Cannot combine information between docs - useful only when you expect that a single simple answer exists in a single doc

CHAINS

OFF-THE-SHELVES CHAINS

These chains are structured assembly of components for accomplishing specific higher-level tasks which can be used out of the box, making it easy to get started.

Document QA

Used for question answering over a list of documents.

Retrieval QA

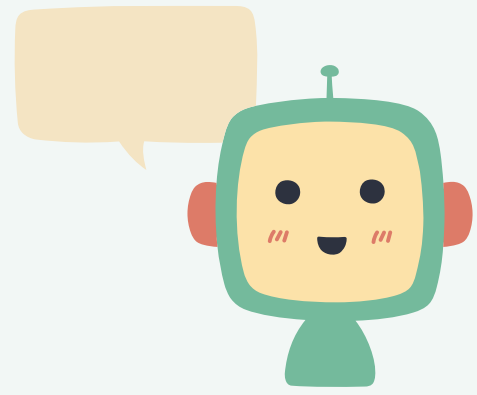
Used for question answering over an index.

Simple Sequential Chain

A simple chain that allows you to join multiple single-input/single-output chains into one chain.

Sequential Chain

A more general form of sequential chains that allows for multiple inputs/outputs. All outputs from all previous chains will be available to the next chain.



Agents

- CONVERSATIONAL CHAT AGENT
- CREATE JSON AGENT
- CREATE SQL AGENT
- CREATE PANDAS DATAFRAME AGENT
- CUSTOM AGENT



Tools

- WOLFRAM ALPHA FOR COMPUTATION
- PYTHONREPLTOOL FOR CODE INTERPRETATION AND EXECUTION
- WIKIPEDIA FOR FACT CHECKING
- CUSTOM MADE TOOLS, ETC

AGENTS

What can Agents do?

LangChain provides agents that have access to a suite of tools. Depending on the user's input, an agent can decide which tools to call.

DEMONSTRATION

QUESTION & ANSWER

Involves fetching multiple documents, and then asking the LLM a question about the documents' content.

DATA ANALYSIS

Manipulating and analysing Pandas Dataframe using natural language

LIMITATIONS

The LangChain framework is relatively new and is actively evolving. The documentation is actively being updated as we are writing this article.

Therefore, there could be a need to actively update the codebase of an application built on top of LangChain as functions within the library could get deprecated.



Conclusion

LangChain represents a significant breakthrough in how we build LLM-powered applications. However, as with any field that is rapidly evolving, it could potentially render some tools to be outdated quickly and other tools to evolve and become more prevalent in the industry.

We need to constantly educate ourselves with how they can be used and what are the best practices in implementing them.

FOR MORE INFORMATION



github.com/cweiqiang/-aiap13_group1_sharing

