Building Text-Based Applications with the ChatGPT API & LangChain

Lucas Soares

10-05-2024

Quick 'Interactivity Notes'

- The presentation will be organized into the following structure:

Quick 'Interactivity Notes'

- The presentation will be organized into the following structure:
  - **Presentation Block**

Quick 'Interactivity Notes'

- The presentation will be organized into the following structure:
  - **Presentation Block**
  - **Notebook Demo**

Quick 'Interactivity Notes'

- The presentation will be organized into the following structure:
  - **Presentation Block**
  - **Notebook Demo**
  - **Quick Q&A + Summary**

Quick 'Interactivity Notes'

- The presentation will be organized into the following structure:
  - **Presentation Block**
  - **Notebook Demo**
  - **Quick Q&A + Summary**
  - **Optional Exercise** During Q&A (for those that don't have questions and want to try something out)

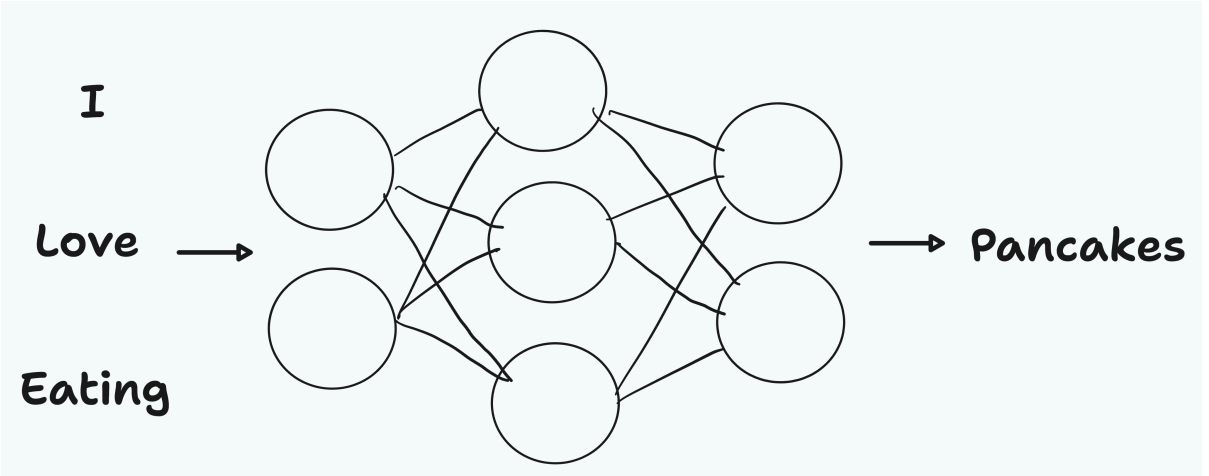
Quick 'Interactivity Notes'

- The presentation will be organized into the following structure:
  - **Presentation Block**
  - **Notebook Demo**
  - **Quick Q&A + Summary**
  - **Optional Exercise** During Q&A (for those that don't have questions and want to try something out)
- Repeat

Large Language Models

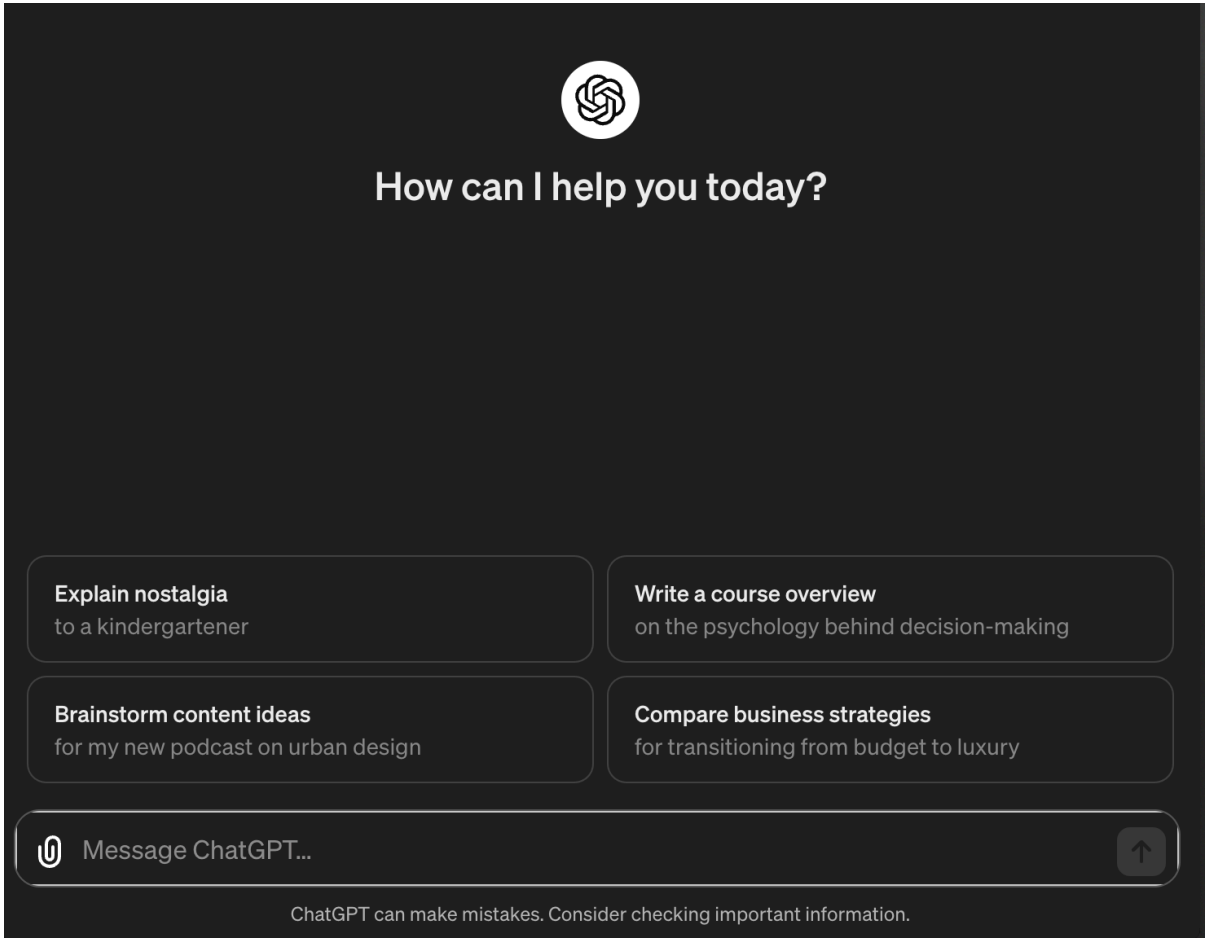
Large Language Models

Large Language Models Predict the Next Word



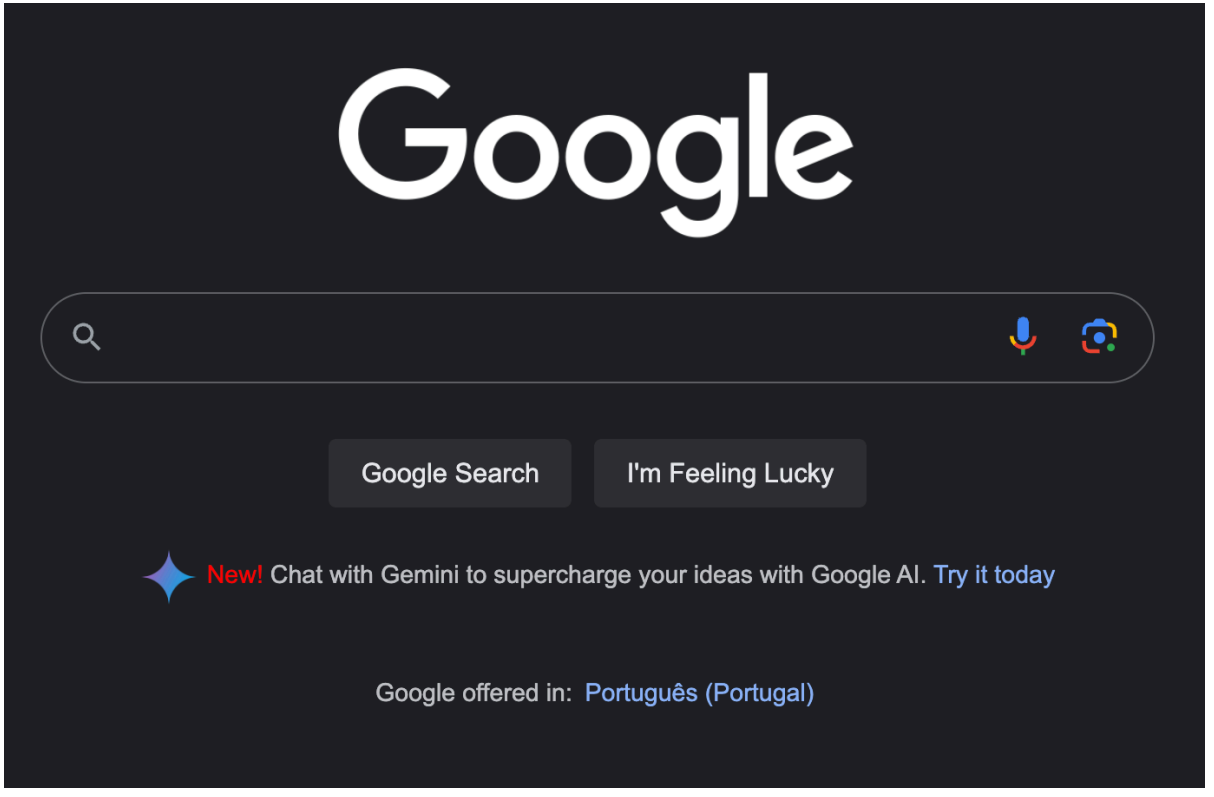
Applications of Large Language Models

- Conversational ChatBots



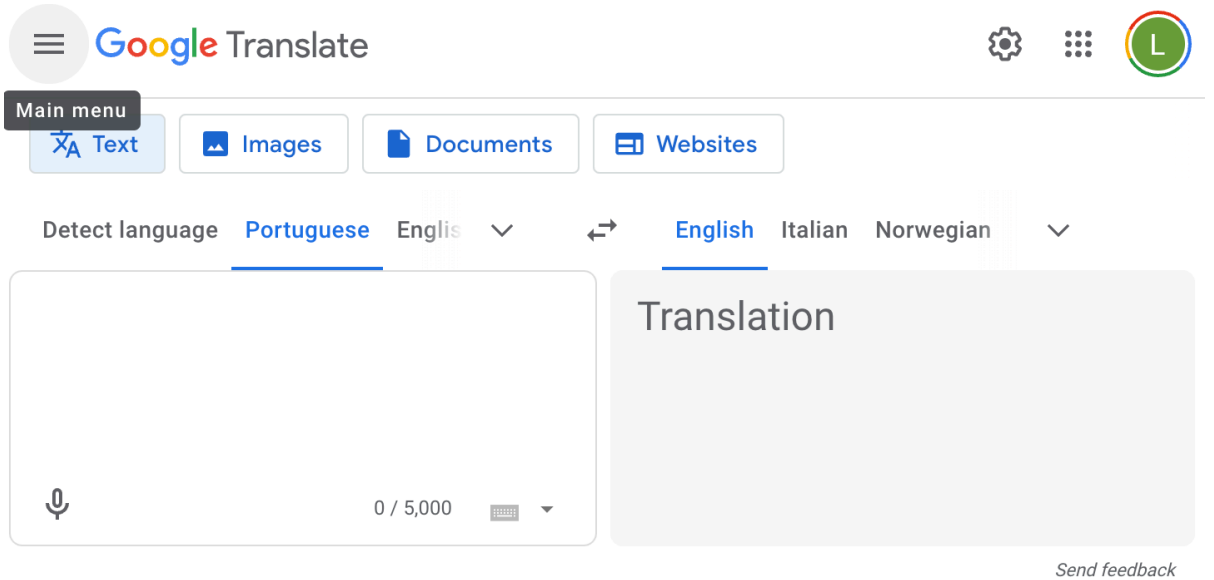
Applications of Large Language Models

- Search Engines



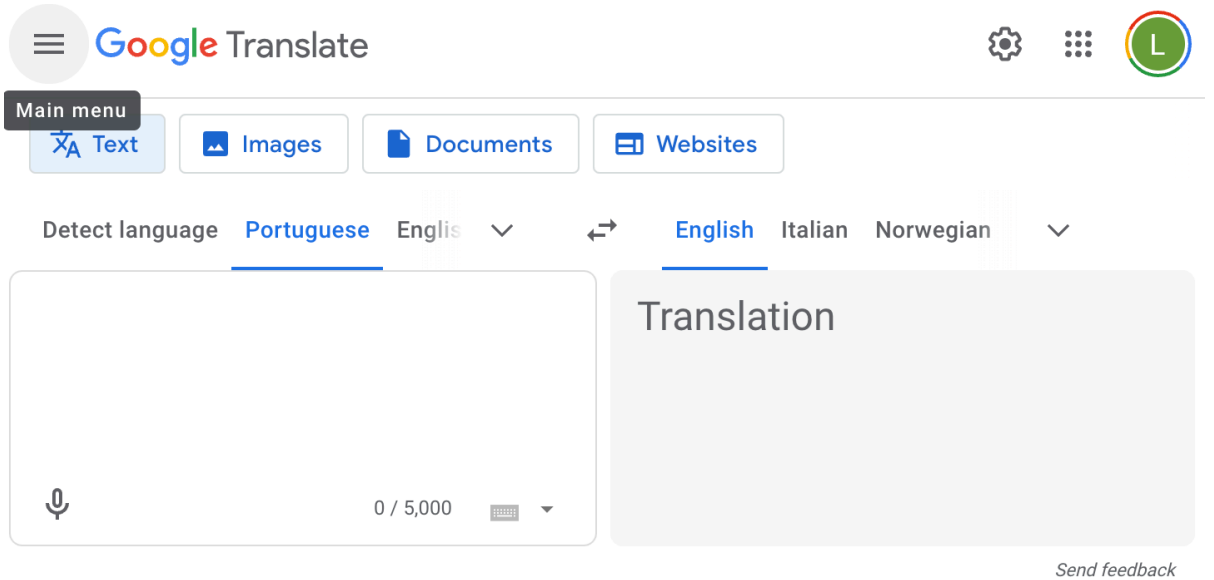
Applications of Large Language Models

- Translation



Applications of Large Language Models

- Translation



- And so much more from Q&A over PDFs to personalized tutoring.

Transformers

(Super quickly)

Transformers

(Super quickly)

They generate text one word at a time. For example:

Transformers

(Super quickly)

They generate text one word at a time. For example:

1. Input: "I love eating __"

Transformers

(Super quickly)

They generate text one word at a time. For example:

1. Input: "I love eating __"
2. Tokenize: I love eating

Transformers

(Super quickly)

They generate text one word at a time. For example:

1. Input: "I love eating __"
2. Tokenize: I love eating
3. Embedding: [0.78, 1.45, 0.35 .....]

Transformers

(Super quickly)

They generate text one word at a time. For example:

1. Input: "I love eating __"
2. Tokenize: I love eating
3. Embedding: [0.78, 1.45, 0.35 .....]
4. Positional encoding:
    1. I 2. love 3. eating

Transformers

(Super quickly)

They generate text one word at a time. For example:

1. Input: "I love eating __"
2. Tokenize: I love eating
3. Embedding: [0.78, 1.45, 0.35 .....]
4. Positional encoding:
    1. I 2. love 3. eating
5. Attention: Understands context → "I" (Lucas) and "love": things that Lucas loves. Cross references that against things that "Lucas" eats leading to ... → "PANCAKES!!!"

Transformers

(Super quickly)

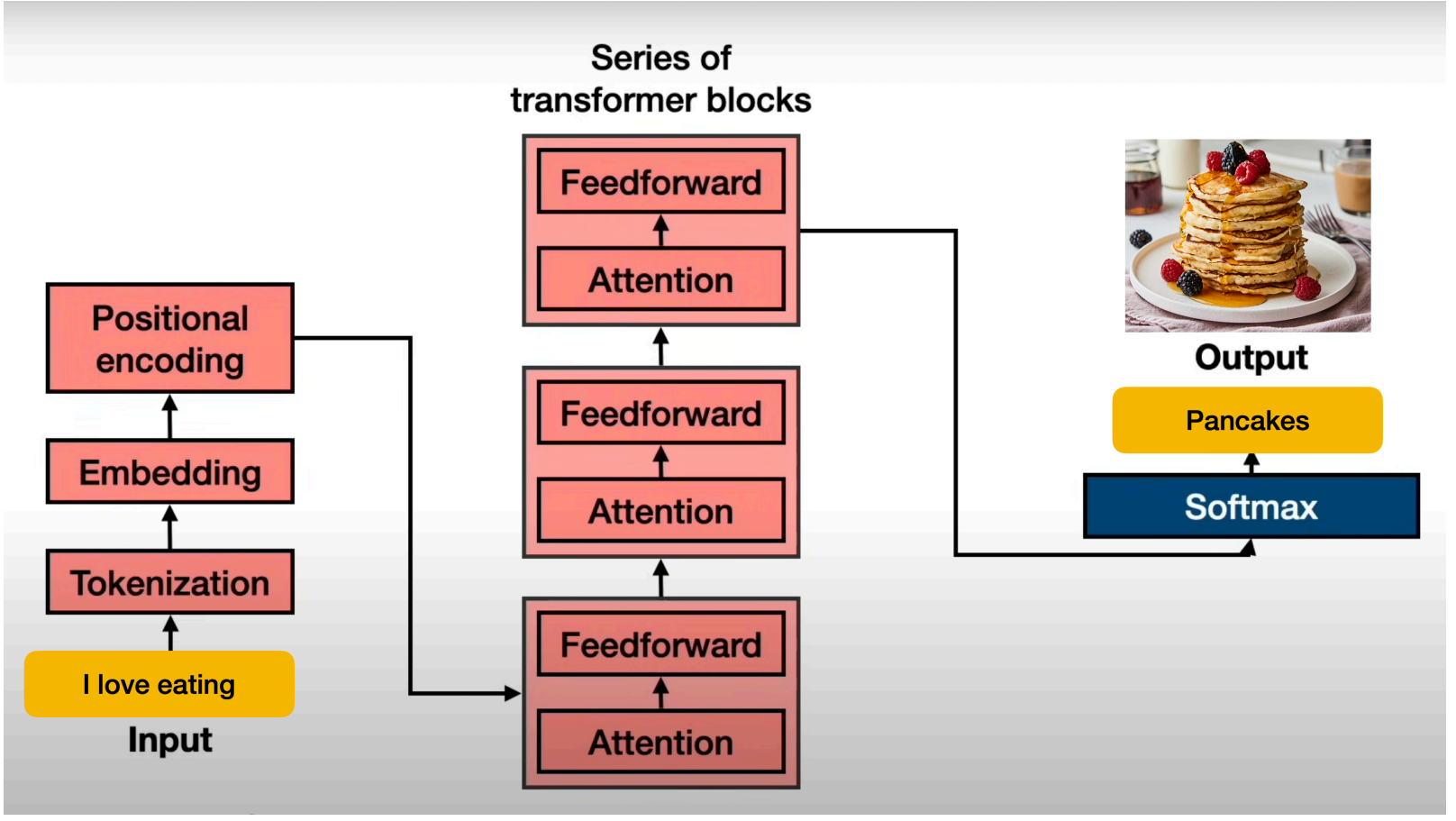
They generate text one word at a time. For example:

1. Input: "I love eating __"
2. Tokenize: I love eating
3. Embedding: [0.78, 1.45, 0.35 .....]
4. Positional encoding:

    1. I 2. love 3. eating

5. Attention: Understands context → "I" (Lucas) and "love": things that Lucas loves. Cross references that against things that "Lucas" eats leading to ... → "PANCAKES!!!"

Transformers



[1] What are Transformers and How do They Work? By Serrano Academy

Benefits of Large Language Models

- Multi-task Capability
    - LLMs find applications in content generation, question answering, translation, tutoring, and personal assistants.

Benefits of Large Language Models

- Multi-task Capability
    - LLMs find applications in content generation, question answering, translation, tutoring, and personal assistants.
- Fine-tuning
    - LLMs can usually be fine tuned with a relatively small amount of data, making them adaptable to a wide range of tasks.

Benefits of Large Language Models

- Multi-task Capability
  - LLMs find applications in content generation, question answering, translation, tutoring, and personal assistants.
- Fine-tuning
  - LLMs can usually be fine tuned with a relatively small amount of data, making them adaptable to a wide range of tasks.
- Scalability
  - LLMs demonstrate excellent scalability to very large capacity networks and huge datasets.

Limitations and Ethical Considerations

LLMs are far from perfect

Limitations and Ethical Considerations

LLMs are far from perfect

- **Knowledge Limit**: LLMs have a cutoff point for their knowledge.

Limitations and Ethical Considerations

LLMs are far from perfect

- **Knowledge Limit**: LLMs have a cutoff point for their knowledge.
- **Understanding Limit**: LLMs do not understand text in the same way humans do. They don't have beliefs or desires; they simply predict what comes next based on their training.

Limitations and Ethical Considerations

LLMs are far from perfect

- **Knowledge Limit**: LLMs have a cutoff point for their knowledge.
- **Understanding Limit**: LLMs do not understand text in the same way humans do. They don't have beliefs or desires; they simply predict what comes next based on their training.
- **Misuse**: LLMs can hallucinate and produce false or harmful content

Limitations and Ethical Considerations

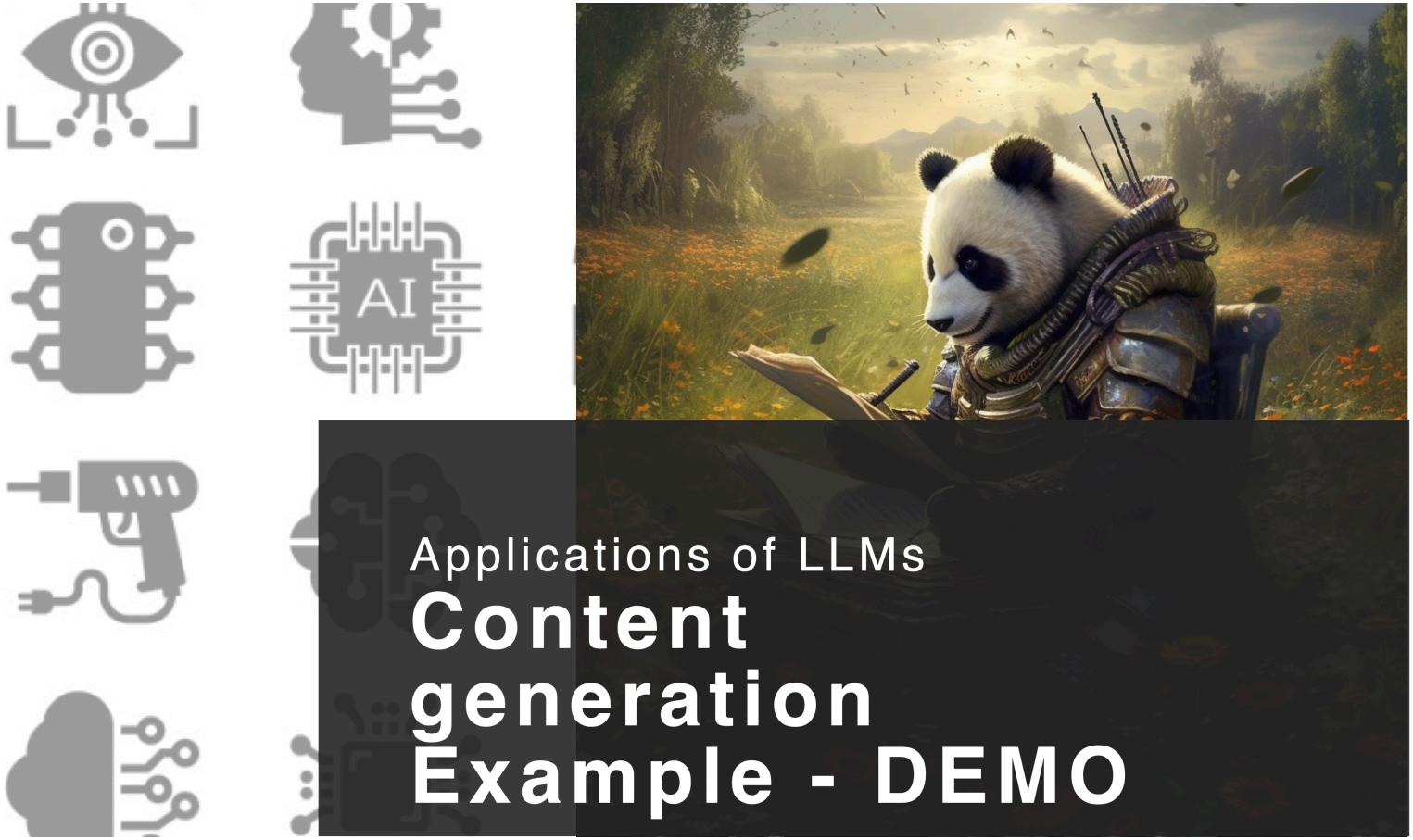
LLMs are far from perfect

- **Knowledge Limit**: LLMs have a cutoff point for their knowledge.
- **Understanding Limit**: LLMs do not understand text in the same way humans do. They don't have beliefs or desires; they simply predict what comes next based on their training.
- **Misuse**: LLMs can hallucinate and produce false or harmful content
- **Reproducibility**: Unpredictability of LLM behavior. Watkins 2023

Limitations and Ethical Considerations

LLMs are far from perfect

- **Knowledge Limit**: LLMs have a cutoff point for their knowledge.
- **Understanding Limit**: LLMs do not understand text in the same way humans do. They don't have beliefs or desires; they simply predict what comes next based on their training.
- **Misuse**: LLMs can hallucinate and produce false or harmful content
- **Reproducibility**: Unpredictability of LLM behavior. Watkins 2023
- **Data Privacy and Bias**: Ethical considerations should extend to the acquisition of data for training additional models. Models may have biases; their use should be transparent and biases mitigated. Watkins 2023



Applications of LLMs

Content generation Example - DEMO

Demo App - Panda Letters

Q&A & Summary

- **LLMs predict the next word** and generate human-like text.
- **LLMs power diverse applications** including conversational chatbots, search engines, and translation services, demonstrating their versatility in handling various language tasks.
- **Transformer architecture** processes text sequentially, utilizing mechanisms like tokenization, embedding, positional encoding, and attention to understand context and generate relevant outputs.
- **Advantages of LLMs**: LLMs are multi-taskers capable of content generation, question answering, and more, with the ability to be fine-tuned for specific tasks and scale effectively to handle large datasets and complex models.
- **Benefits and Limitations**: Despite their benefits (multi-tasking, fine-tuning etc..), LLMs have limitations, including a fixed knowledge base, lack of true understanding, potential for misuse, and ethical concerns around data privacy and inherent biases.
- Optional Exercise
    - Set up your OpenAI API key and prepare for the practical demos

Break

Introduction to the ChatGPT API

Introduction to the ChatGPT API

- The ChatGPT API allows you to use OpenAI's models to generate human-like text based on the input you provide.

Introduction to the ChatGPT API

- The ChatGPT API allows you to use OpenAI's models to generate human-like text based on the input you provide.
- Basic structure of calling the API:

```
from openai import OpenAI
client = OpenAI()

def get_response(prompt_question):
    response = client.chat.completions.create(
        model="gpt-3.5-turbo-0125",
        messages=[{"role": "system", "content": "You are a helpful research and
                {"role": "user", "content": prompt_question}]
    )

    return response.choices[0].message.content
```

Introduction to the ChatGPT API

- The ChatGPT API allows you to use OpenAI's models to generate human-like text based on the input you provide.
- Basic structure of calling the API:

```
from openai import OpenAI
client = OpenAI()

def get_response(prompt_question):
    response = client.chat.completions.create(
        model="gpt-3.5-turbo-0125",
        messages=[{"role": "system", "content": "You are a helpful research and
                {"role": "user", "content": prompt_question}]
    )

    return response.choices[0].message.content
```

- Required parameters: `model`, `messages`

[2] OpenAI Docs

Notebook Demo 1.0 - Intro to ChatGPT API & Prompt Basics

Prompt Basics

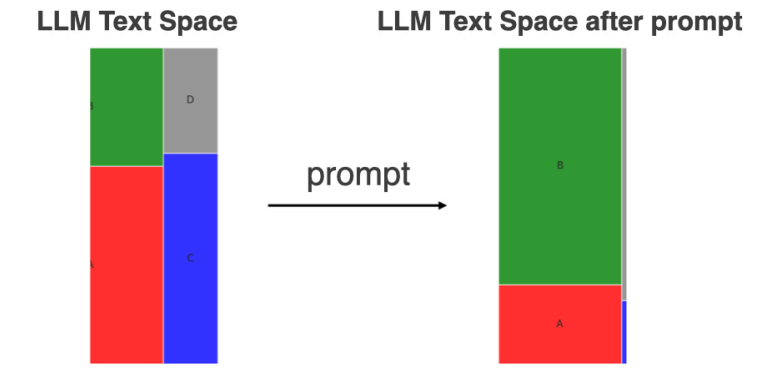
A prompt is a piece of text that conveys to the LLM the user's intention.

Prompt Basics

- A prompt is a piece of text that conveys to the LLM the user's intention.
- Question → Instruction → Behavior

Prompt Basics

- A prompt is a piece of text that conveys to the LLM the user's intention.
- Question → Instruction → Behavior
- It **constrains the space of possibilities** in the LLM's text space



Prompt Basics

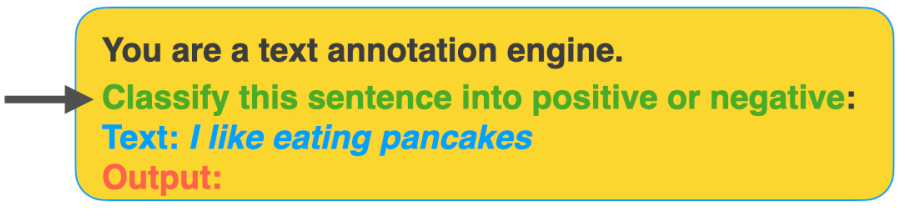
Components of the prompt

You are a text annotation engine.
Classify this sentence into positive or negative:
Text: *I like eating pancakes*
Output:

Prompt Basics

Components of the prompt: instruction

- **Instruction**: where you describe what you want



Prompt Basics

Components of the prompt: instruction, context

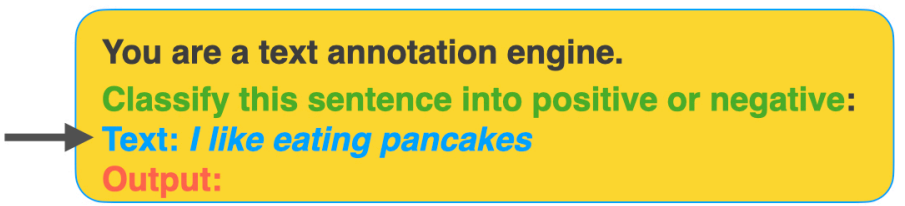
- **Instruction:** where you describe what you want
- **Context:** additional information to help with performance.

You are a text annotation engine.
Classify this sentence into positive or negative:
Text: ***I like eating pancakes***
Output:

Prompt Basics

Components of the prompt: instruction, context input data

- **Instruction:** where you describe what you want
- **Context:** additional information to help with performance.
- **Input data:** example of data you wish the model to process



Prompt Basics

Components of the prompt: instruction, context input data , output indicator

- **Instruction:** where you describe what you want
- **Context:** additional information to help with performance.
- **Input data:** example of data you wish the model to process
- **Output indicator:** Priming model to output what you want (e.g structure, length, etc..)

You are a text annotation engine.
Classify this sentence into positive or negative:
Text: ***I like eating pancakes***
Output:

Q&A & Summary

- **Prompts** are the inputs to LLMs__
- **Prompts** allow users to **guide the model's responses** and generate specific outputs.
- **Components of a prompt:** instruction, context, input data, and output indicator.
- **ChatGPT API:** programatic access to GPT models by OpenAI
- **ChatGPT API:** required parameters: `model`, `messages`
- **Optional Exercise During Q&A**

  `Create a prompt that summarizes a piece of text in a format/structure of your choice.`

Break

Break

Fine Tuning ChatGPT Applications

A primer on how to fine tune ChatGPT on your data

Fine Tuning:

Fine Tuning ChatGPT Applications

A primer on how to fine tune ChatGPT on your data

Fine Tuning: process of “specialising an LLM on your specific dataset”

Fine Tuning ChatGPT Applications

A primer on how to fine tune ChatGPT on your data

- **Fine Tuning:** process of “specialising an LLM on your specific dataset”
- **Why?**

Fine Tuning ChatGPT Applications

A primer on how to fine tune ChatGPT on your data

- **Fine Tuning:** process of “specialising an LLM on your specific dataset”
- **Why?** Better results on specific tasks, lower latency… **See OpenAI docs**

Fine Tuning ChatGPT Applications

A primer on how to fine tune ChatGPT on your data

- **Fine Tuning:** process of “specialising an LLM on your specific dataset”
- **Why?** Better results on specific tasks, lower latency… **See OpenAI docs**
- **How?**

Fine Tuning ChatGPT Applications

A primer on how to fine tune ChatGPT on your data

- **Fine Tuning:** process of “specialising an LLM on your specific dataset”
- **Why?** Better results on specific tasks, lower latency… **See OpenAI docs**
- **How?** By preparing a custom `.jsonl` dataset on your specific use case

Fine Tuning ChatGPT Applications

A primer on how to fine tune ChatGPT on your data

- **Fine Tuning:** process of “specialising an LLM on your specific dataset”
- **Why?** Better results on specific tasks, lower latency… **See OpenAI docs**
- **How?** By preparing a custom `.jsonl` dataset on your specific use case

The goal of fine tuning is to adapt the model to your specific use case by providing it with a few examples of successful interactions

[4] OpenAI's Fine Tuning Docs

Fine Tuning ChatGPT Applications

Common Use Cases

Fine Tuning ChatGPT Applications

Common Use Cases

- Setting style, tone, format

OpenAI docs

Fine Tuning ChatGPT Applications

Common Use Cases

- Setting style, tone, format
- Correcting failures

[OpenAI docs]

Fine Tuning ChatGPT Applications

Common Use Cases

- Setting style, tone, format
- Correcting failures
- Improving reliability

[OpenAI docs]

Fine Tuning ChatGPT Applications

Common Use Cases

- Setting style, tone, format
- Correcting failures
- Improving reliability
- Handling edge cases

[OpenAI docs]

Fine Tuning ChatGPT Applications

Common Use Cases

- Setting style, tone, format
- Correcting failures
- Improving reliability
- Handling edge cases
- Extracting structured output

[OpenAI docs]

Fine Tuning ChatGPT Applications

Common Use Cases

- Setting style, tone, format
- Correcting failures
- Improving reliability
- Handling edge cases
- Extracting structured output
- Function Calling

[OpenAI docs]

Fine Tuning ChatGPT Applications

Basic Steps

OpenAI docs

Fine Tuning ChatGPT Applications

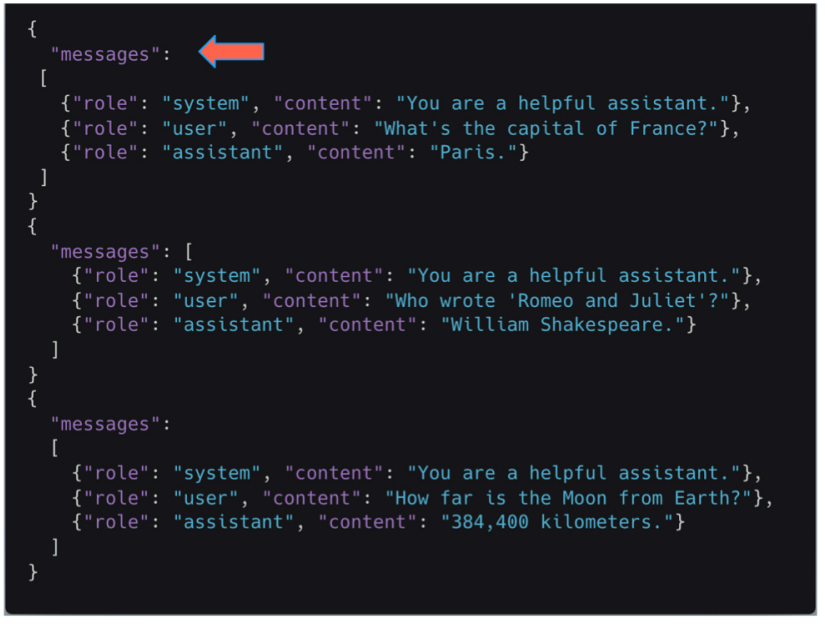
Basic Steps

- Prepare and upload training data

[OpenAI docs]

Fine Tuning ChatGPT Applications

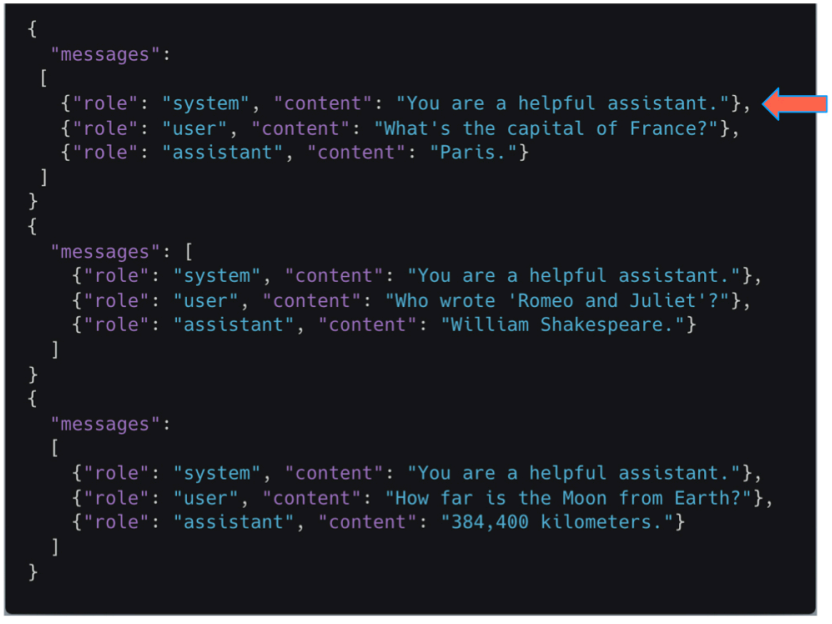
Prepare and upload training data



```
{
  "messages":
 [
   {"role": "system", "content": "You are a helpful assistant."},
   {"role": "user", "content": "What's the capital of France?"},
   {"role": "assistant", "content": "Paris."}
 ]
}
{
  "messages": [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"},
    {"role": "assistant", "content": "William Shakespeare."}
  ]
}
{
  "messages":
  [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "How far is the Moon from Earth?"},
    {"role": "assistant", "content": "384,400 kilometers."}
  ]
}
```

Fine Tuning ChatGPT Applications

Prepare and upload training data



```
{
  "messages":
 [
   {"role": "system", "content": "You are a helpful assistant."},
   {"role": "user", "content": "What's the capital of France?"},
   {"role": "assistant", "content": "Paris."}
 ]
}
{
  "messages": [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"},
    {"role": "assistant", "content": "William Shakespeare."}
  ]
}
{
  "messages":
  [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "How far is the Moon from Earth?"},
    {"role": "assistant", "content": "384,400 kilometers."}
  ]
}
```

Fine Tuning ChatGPT Applications

Prepare and upload training data

```
{
  "messages":
 [
   {"role": "system", "content": "You are a helpful assistant."},
   {"role": "user", "content": "What's the capital of France?"},
   {"role": "assistant", "content": "Paris."}
 ]
}
{
  "messages": [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"},
    {"role": "assistant", "content": "William Shakespeare."}
  ]
}
{
  "messages":
  [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "How far is the Moon from Earth?"},
    {"role": "assistant", "content": "384,400 kilometers."}
  ]
}
```

Fine Tuning ChatGPT Applications

Prepare and upload training data

```
{
  "messages":
 [
   {"role": "system", "content": "You are a helpful assistant."},
   {"role": "user", "content": "What's the capital of France?"},
   {"role": "assistant", "content": "Paris."}
 ]
}
{
  "messages": [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"},
    {"role": "assistant", "content": "William Shakespeare."}
  ]
}
{
  "messages":
  [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "How far is the Moon from Earth?"},
    {"role": "assistant", "content": "384,400 kilometers."}
  ]
}
```

Fine Tuning ChatGPT Applications

Prepare and upload training data

```
{
  "messages":
 [
   {"role": "system", "content": "You are a helpful assistant."},
   {"role": "user", "content": "What's the capital of France?"},
   {"role": "assistant", "content": "Paris."}
 ]
}
{
  "messages": [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"},
    {"role": "assistant", "content": "William Shakespeare."}
  ]
}
{
  "messages":
  [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "How far is the Moon from Earth?"},
    {"role": "assistant", "content": "384,400 kilometers."}
  ]
}
```

Fine Tuning ChatGPT Applications

Prepare and upload training data

```
{
  "messages":
 [
   {"role": "system", "content": "You are a helpful assistant."},
   {"role": "user", "content": "What's the capital of France?"},
   {"role": "assistant", "content": "Paris."}
 ]
}
{
  "messages": [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"},
    {"role": "assistant", "content": "William Shakespeare."}
  ]
}
{
  "messages":
  [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "How far is the Moon from Earth?"},
    {"role": "assistant", "content": "384,400 kilometers."}
  ]
}
```

Fine Tuning ChatGPT Applications

Basic Steps

- Prepare and upload training data
- Train a new fine-tuned model

OpenAI docs

Fine Tuning ChatGPT Applications

Fine Tuning Jobs

```python
from openai import OpenAI
# OpenAI API key should be set as
# environment variable - OPENAI_API_KEY
client = OpenAI()
client.files.create(
  file=open("dataset.jsonl", "rb"),
  purpose="fine-tune"
)
client.fine_tuning.jobs.create(
  training_file="file-id",
  model="gpt-3.5-turbo-1106"
)
```

Fine Tuning ChatGPT Applications

Basic Steps

- Prepare and upload training data
- Train a new fine-tuned model
- Use your fine-tuned model

[OpenAI docs]

Fine Tuning ChatGPT Applications

Fine Tuning Jobs

```python
def get_response(prompt, fine_tuned_model_id="ft:gpt-3.5-turbo-1106:personal::8Oq91T4e"):
    client = OpenAI()
    response = client.chat.completions.create(model=fine_tuned_model_id,
                                              messages=
                                              [
                                                  {"role": "system", "content": "You are a helpful assistant."},
                                                  {"role": "user", "content": prompt}
                                              ],
                                              temperature=0.0,
                                              n = 1
                                              )
    return response.choices[0].message.content
```

Disclaimer: Example code is provided in the repository notebooks

Notebook Demo 3.0 - Fine Tuning ChatGPT API

What is Langchain?

What is Langchain?

- **LangChain is a framework that facilitates creation of LLM-based applications**

What is Langchain?

- **LangChain is a framework that facilitates creation of LLM-based applications**
- **Main features**:

What is Langchain?

- **LangChain is a framework that facilitates creation of LLM-based applications**
- **Main features**:
  - components

What is Langchain?

- **LangChain is a framework that facilitates creation of LLM-based applications**
- **Main features**:
  - components
  - off-the-shelf-chains

What is Langchain?

- **LangChain is a framework that facilitates creation of LLM-based applications**
- **Main features**:
  - components
  - off-the-shelf-chains
- LangChain gives you building blocks for creating powerful LLM applications

[5] LangChain Docs

LangChain Components

LangChain Components

Models

LangChain Components

Models

- Abstractions over the LLM APIs like the ChatGPT API

LangChain Components

Models

- Abstractions over the LLM APIs like the ChatGPT API

```
from langchain_openai import ChatOpenAI

chat_model = ChatOpenAI(model="gpt-3.5-turbo-0125")

output = chat_model.invoke("I am teaching a live-training\
  about LLMs!")

print(output.content)
```

[5] LangChain Docs

LangChain Components

LangChain Components

Prompt Templates

LangChain Components

Prompt Templates

- Abstractions over standard prompts to LLMs

LangChain Components

Prompt Templates

- Abstractions over standard prompts to LLMs

```
from langchain_core.prompts import ChatPromptTemplate

prompt = ChatPromptTemplate.from_template(
  """Show me 5 examples of this concept: {concept}"""
  )

prompt.format(concept="animal")

# Output
# 'Human: Show me 5 examples of this concept: animal'
```

[5] LangChain Docs

LangChain Components

LangChain Components

Output Parsers

LangChain Components

Output Parsers

- Translates raw output from LLM to a workable format

LangChain Components

Output Parsers

- Translates raw output from LLM to a workable format

```
from langchain_core.output_parsers import StrOutputParser
output_parser = StrOutputParser()
```

[5] LangChain Docs

Chains in LangChain

Chains in LangChain

Chain = Model + Prompt + Output Parser

Chains in LangChain

Chain = Model + Prompt + Output Parser

- Chains are the building blocks in LangChain

Chains in LangChain

Chain = Model + Prompt + Output Parser

- Chains are the building blocks in LangChain
- They are used to compose abstractions that go from simple to complex components

Chains in LangChain

Chain = Model + Prompt + Output Parser

- Chains are the building blocks in LangChain
- They are used to compose abstractions that go from simple to complex components

```
llm = ChatOpenAI(model="gpt-3.5-turbo-0125")
prompt = ChatPromptTemplate.from_template("""
output_parser = StrOutputParser()
Write 5 concepts that are fundamental to learn about {topic}.
                                            """)
chain = prompt | llm | output_parser
chain.invoke({"topic": "Artificial Neural Networks"})
```

[5] LangChain Docs

LCEL - LangChain Expression Language

Composing Chains with LCEL

LCEL - LangChain Expression Language

Composing Chains with LCEL

- LCEL is a powerful, declarative language designed to streamline the composition of complex chains in the LangChain framework.

LCEL - LangChain Expression Language

Composing Chains with LCEL

- LCEL is a powerful, declarative language designed to streamline the composition of complex chains in the LangChain framework.
- Pipe syntax

LCEL - LangChain Expression Language

Composing Chains with LCEL

- LCEL is a powerful, declarative language designed to streamline the composition of complex chains in the LangChain framework.
- Pipe syntax


```
chain = prompt | llm | output_parser
```

LCEL - LangChain Expression Language

Composing Chains with LCEL

- LCEL is a powerful, declarative language designed to streamline the composition of complex chains in the LangChain framework.
- Pipe syntax


```
chain = prompt | llm | output_parser
```

- Allows you to build complex chain pipelines with a simple standard interface

[5] LangChain Docs

LCEL - Runnables

- To facilitate creation of custom chains, LangChain has implemented a **"Runnable" protocol**.

LCEL - Runnables

- To facilitate creation of custom chains, LangChain has implemented a **"Runnable" protocol**.
- The standard interface includes `stream`, `invoke`, and `batch` methods. Async methods are also available

LCEL - Runnables

- To facilitate creation of custom chains, LangChain has implemented a **"Runnable" protocol**.
- The standard interface includes `stream`, `invoke`, and `batch` methods. Async methods are also available
- The input type and output type vary by component:

Component	Input Type	Output Type
Prompt	Dictionary	PromptValue
ChatModel	Single string, list of chat messages or a PromptValue	ChatMessage
LLM	Single string, list of chat messages or a PromptValue	String
OutputParser	The output of an LLM or ChatModel	Depends on the parser
Retriever	Single string	List of Documents
Tool	Single string or dictionary, depending on the tool	Depends on the tool

[5] LangChain Docs

Notebook Demo 4.0 - Intro to LangChain

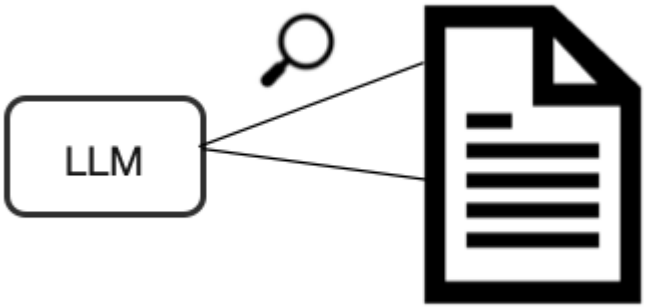
Q&A & Summary

- Fine tuning ChatGPT API: process of training a pre-trained model on a specific task or dataset.
- Fine tuning steps for the ChatGPT API: prepare the data, train the model, use the model.
- **LLMs can predict the next word in a sequence**. ("I Like eating...? ;P ")
- **Chains** are the building blocks in LangChain, composed of Models, Prompt Templates, and Output Parsers.
- **LCEL** is a declarative language that users the Unix pipe symbol to build complex chain pipelines with a simple standard interface.
- Optional Exercise During Q&A

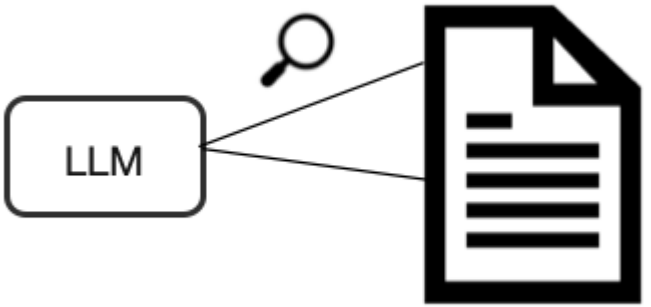
  ```
  Create a simple chain that can answer questions about a specific topic.
  ```

Break 5 minutes

LangChain for Chat Over Documents

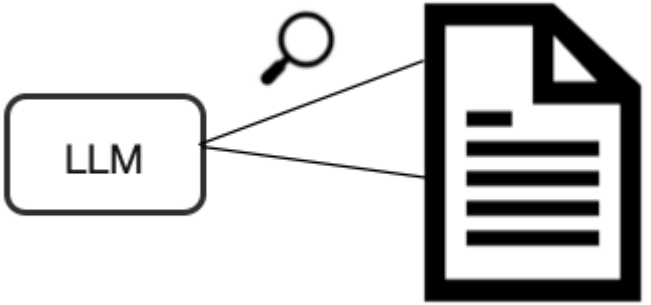


LangChain for Chat Over Documents



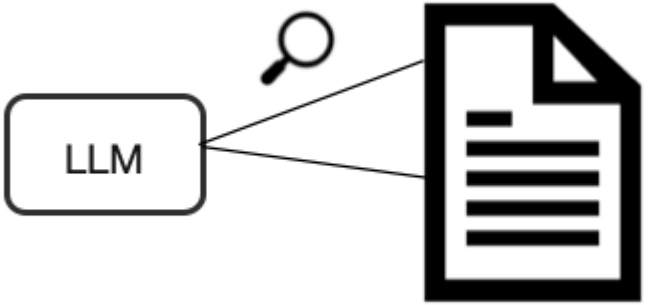
- RAG = **R**etrieval **A**ugmented **G**eneration

LangChain for Chat Over Documents



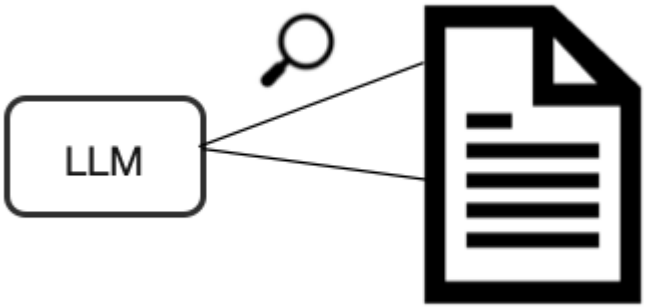
- RAG = **R**etrieval **A**ugmented **G**eneration
- It's about connecting LLMs to documents like PDFs, Text files, HTML, etc.

LangChain for Chat Over Documents



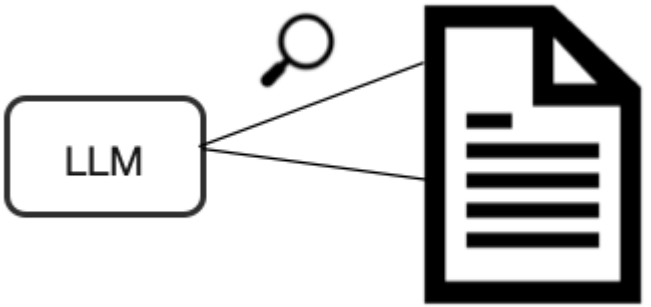
- RAG = **R**etrieval **A**ugmented **G**eneration
- It's about connecting LLMs to documents like PDFs, Text files, HTML, etc.
- How do we get around the context length limitations of LLMs?

LangChain for Chat Over Documents



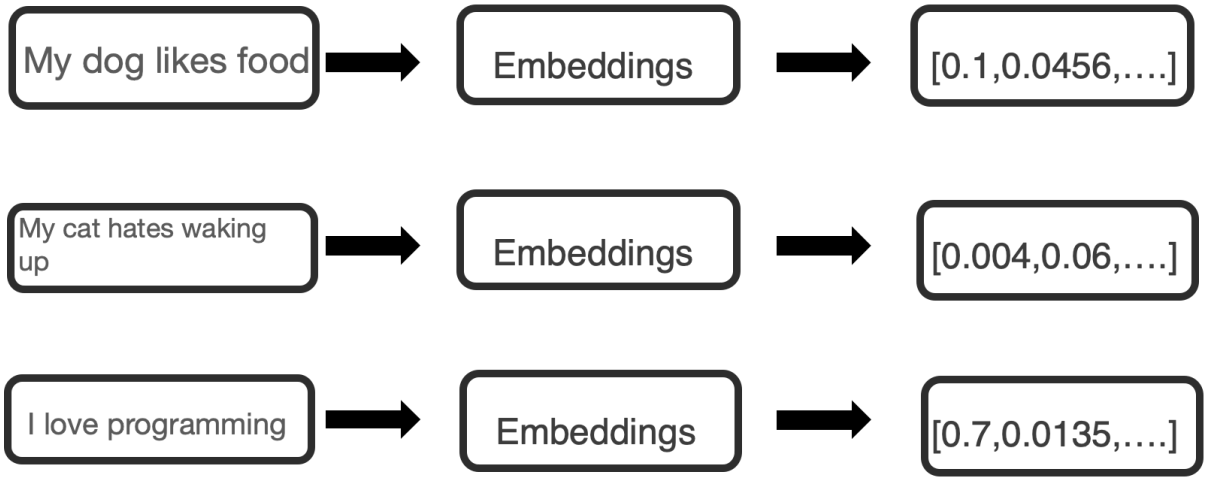
- RAG = **R**etrieval **A**ugmented **G**eneration
- It's about connecting LLMs to documents like PDFs, Text files, HTML, etc.
- How do we get around the context length limitations of LLMs?
- Quick Answer is **Embeddings**!

LangChain for Chat Over Documents

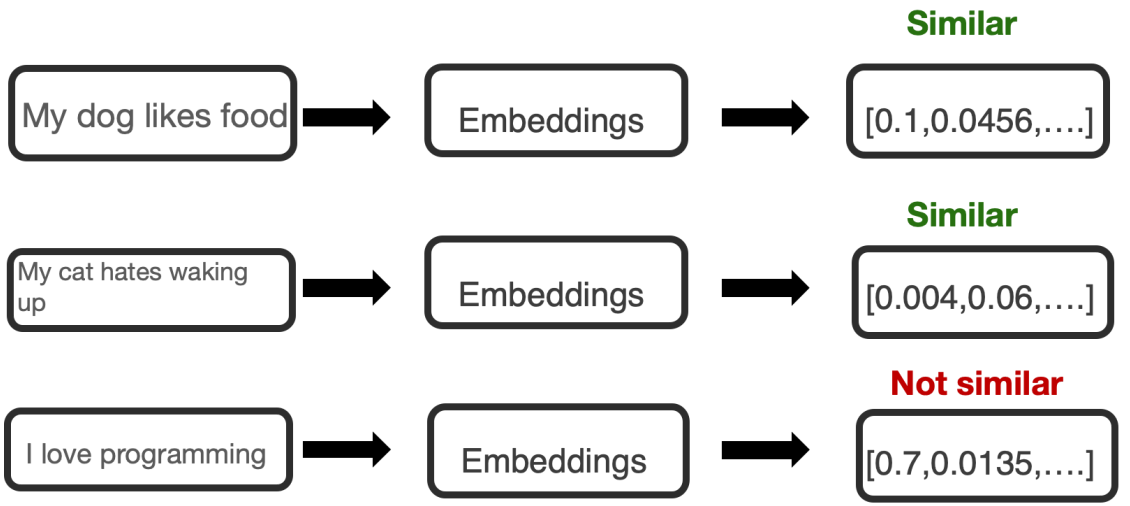


- Embeddings are vectorized representations of text

LangChain for Chat Over Documents

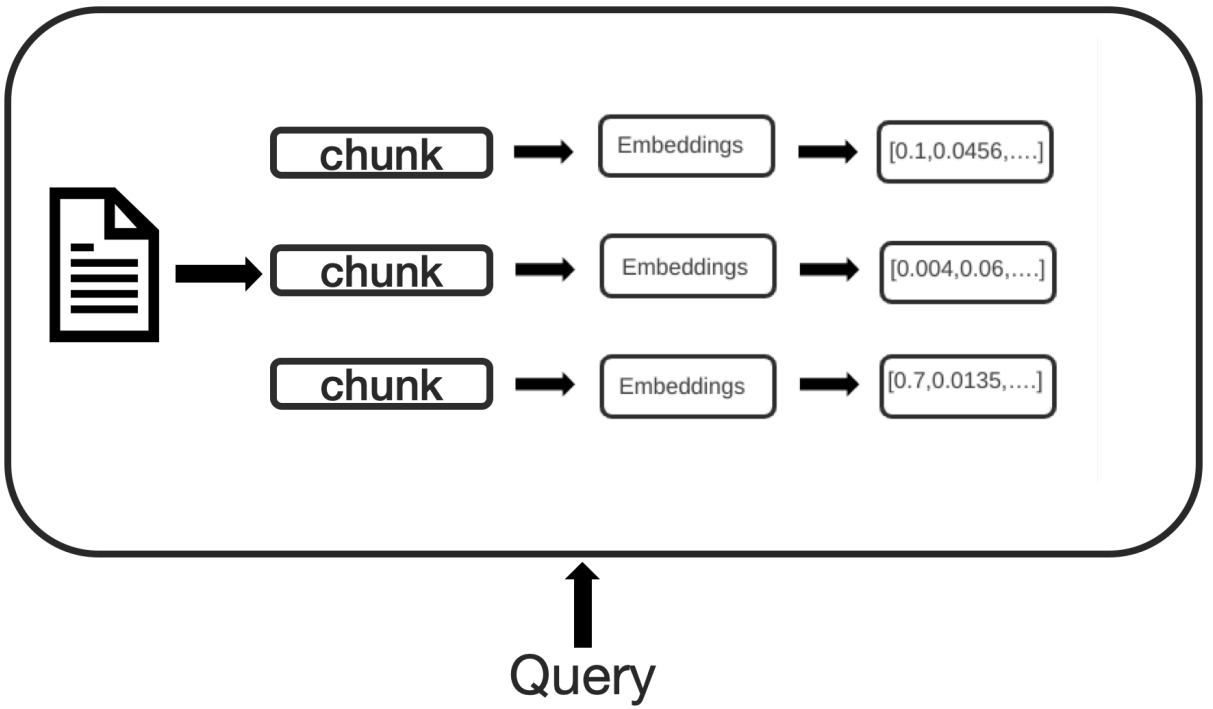


LangChain for Chat Over Documents



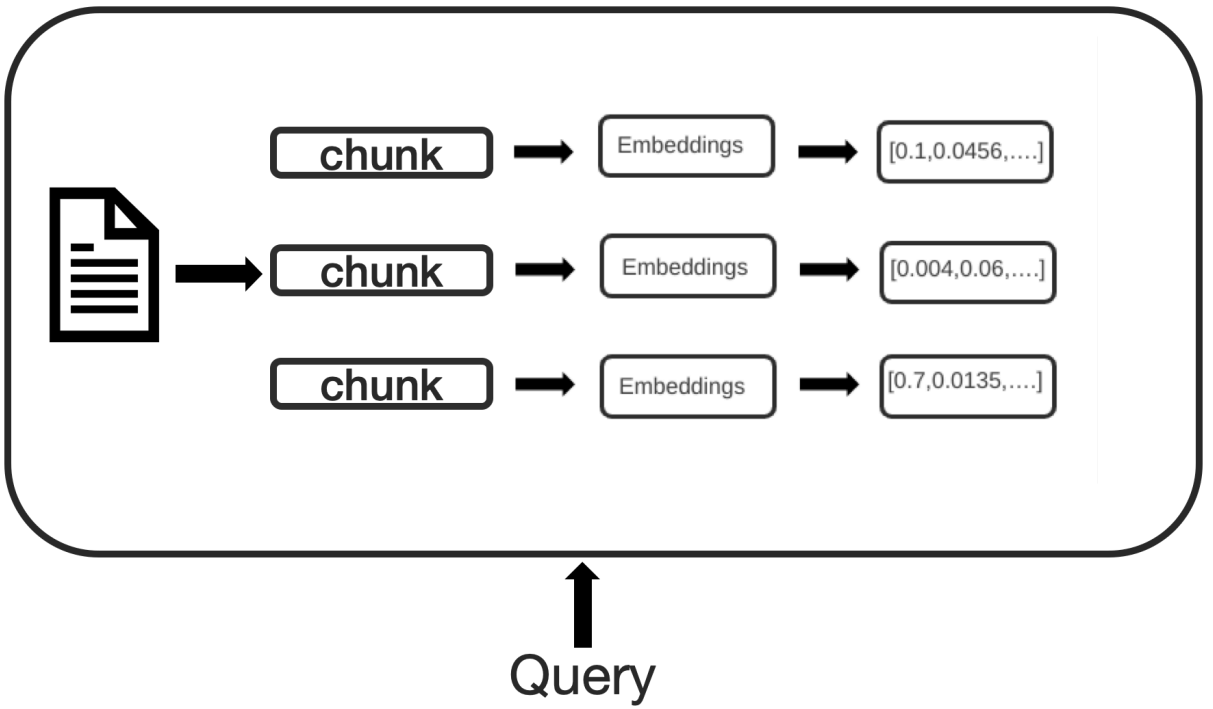
- Embeddings capture content and meaning

LangChain for Chat Over Documents



- Embeddings capture content and meaning
- Vector DBs

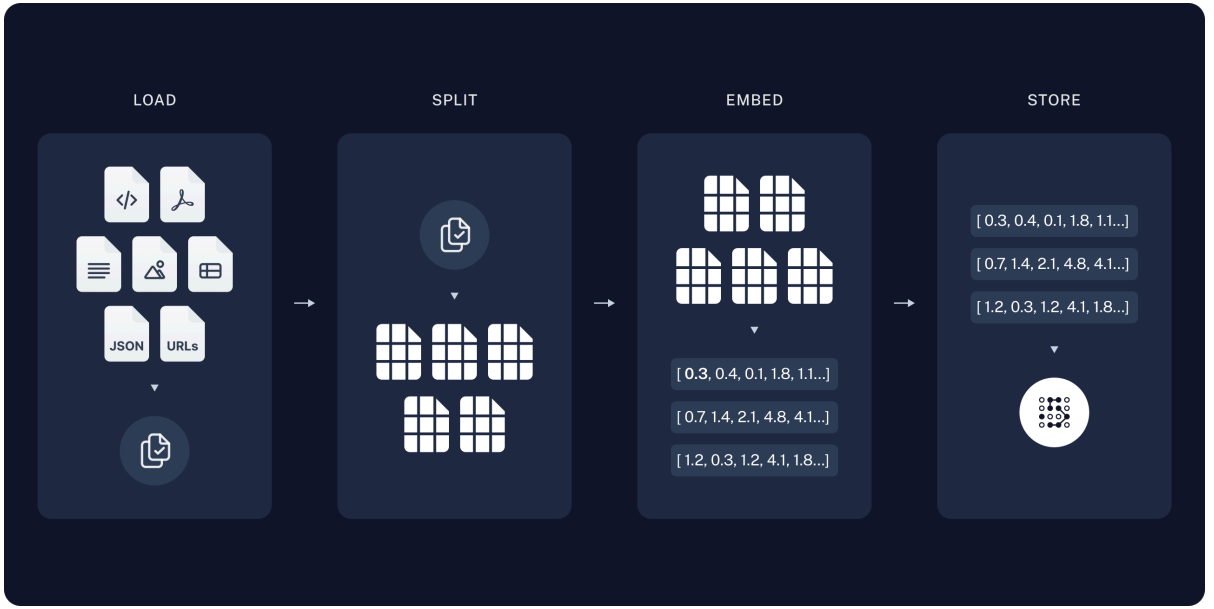
LangChain for Chat Over Documents



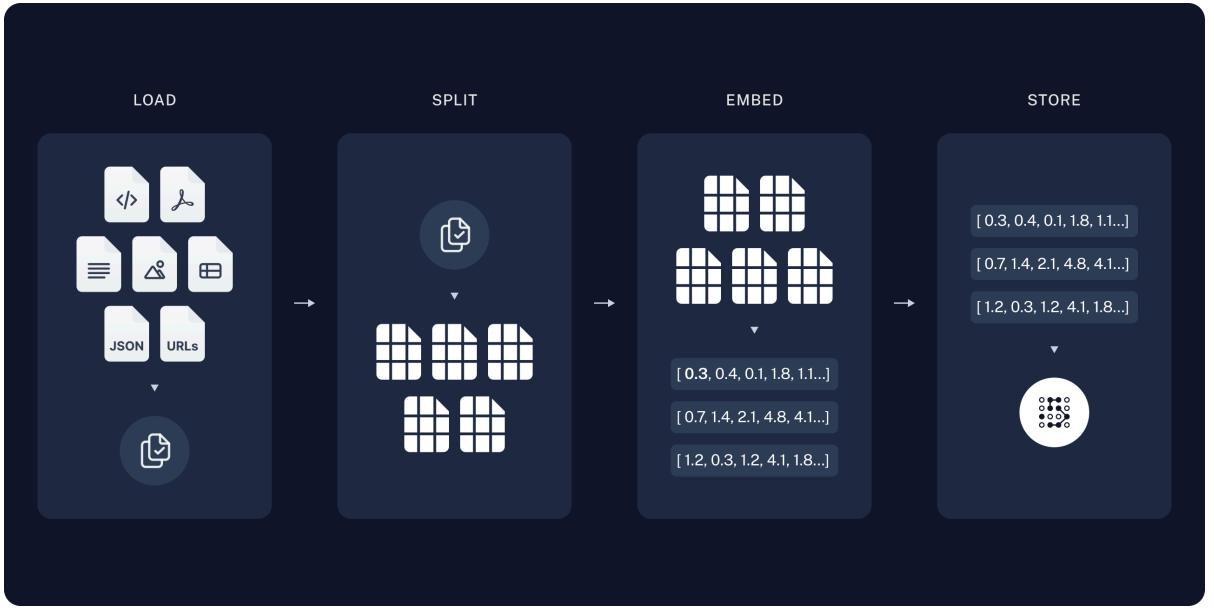
- Embeddings capture content and meaning
- Vector DBs
- How to build RAG systems with LangChain?

LangChain for Chat Over Documents

LangChain for Chat Over Documents

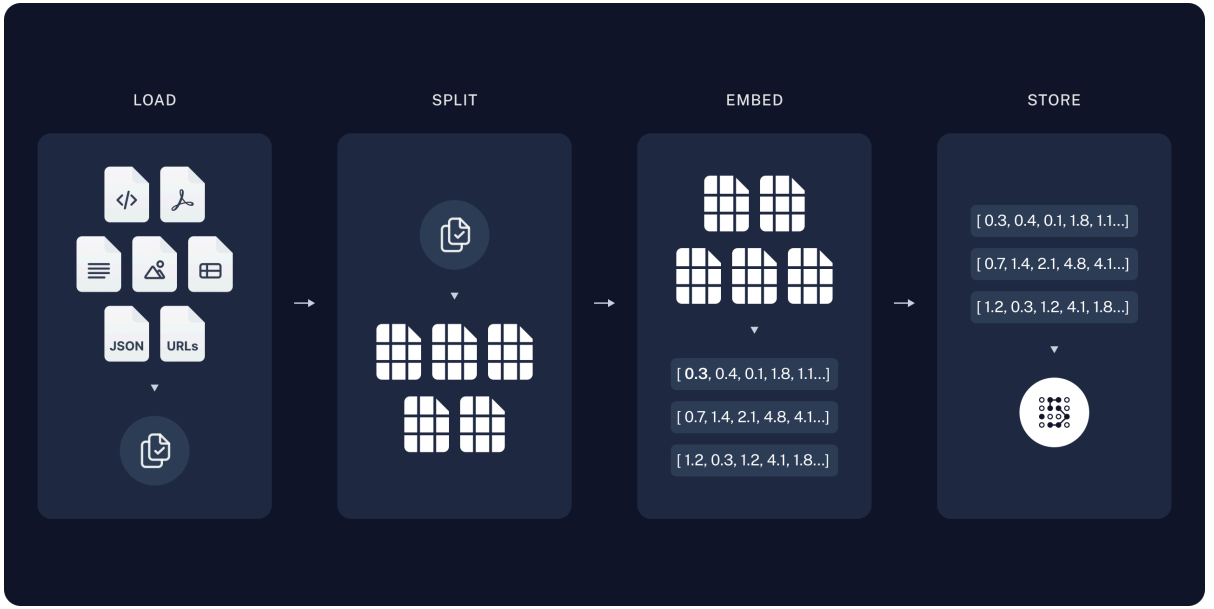


LangChain for Chat Over Documents



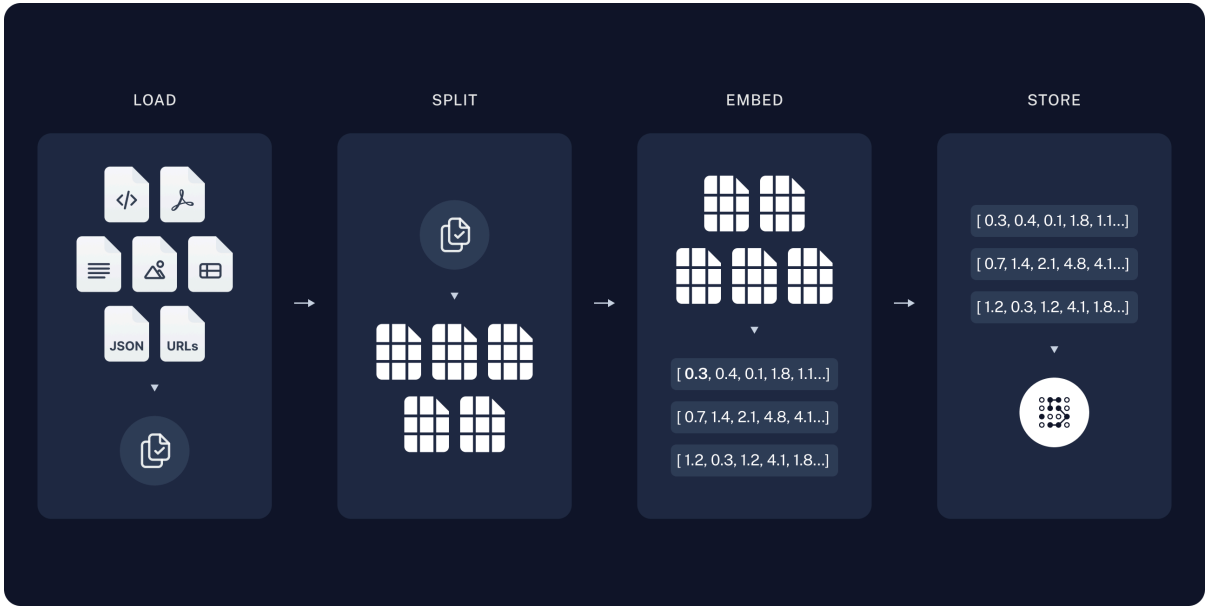
- Load

LangChain for Chat Over Documents



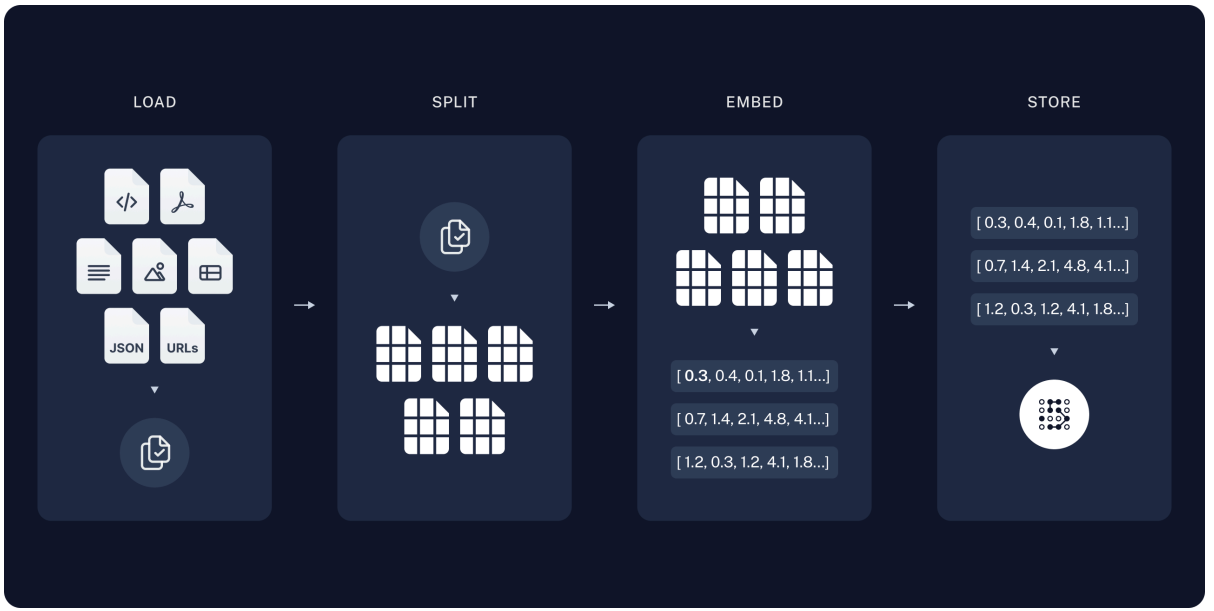
- Load
- Split

LangChain for Chat Over Documents



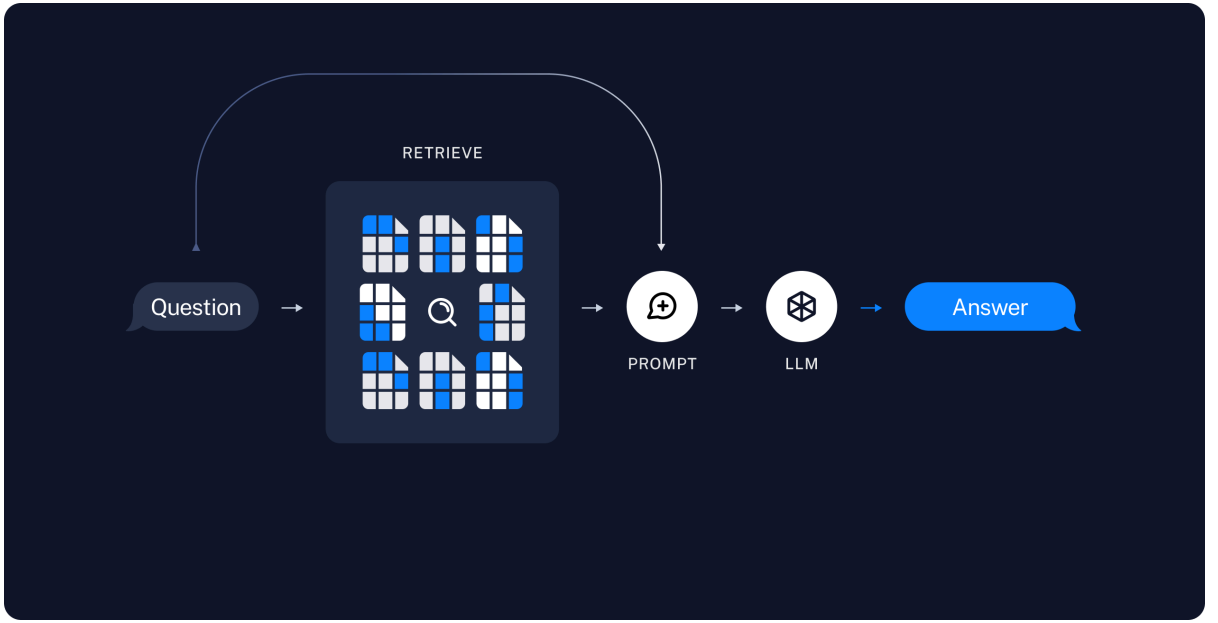
- Load
- Split
- Embed

LangChain for Chat Over Documents

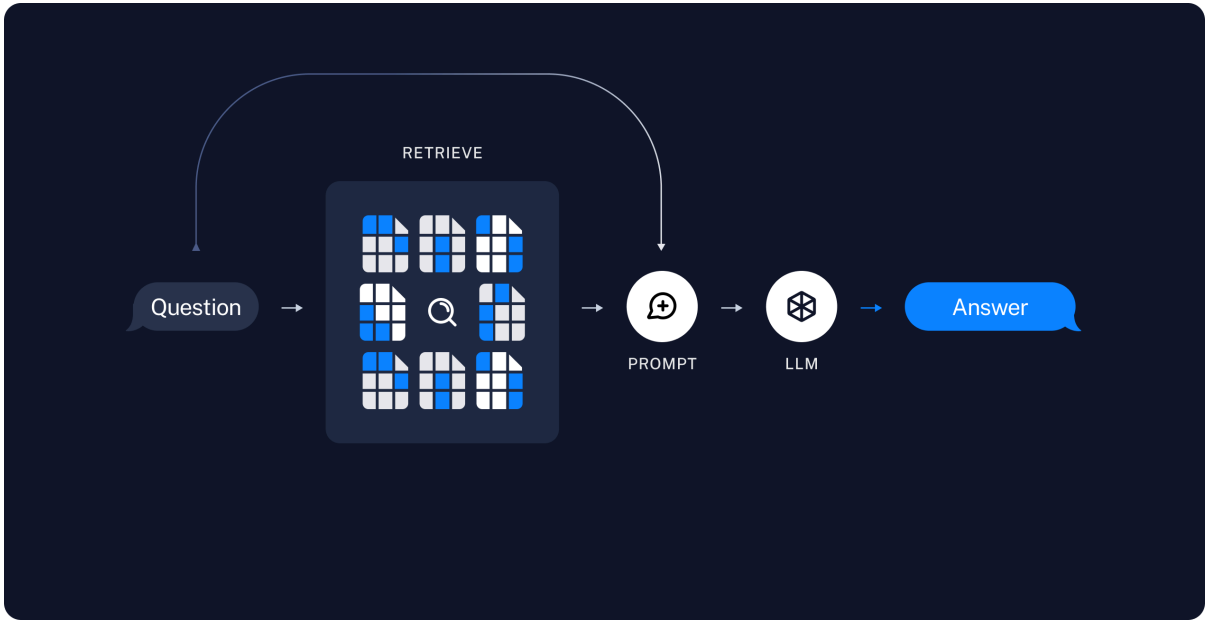


- Load
- Split
- Embed
- Store

LangChain for Chat Over Documents

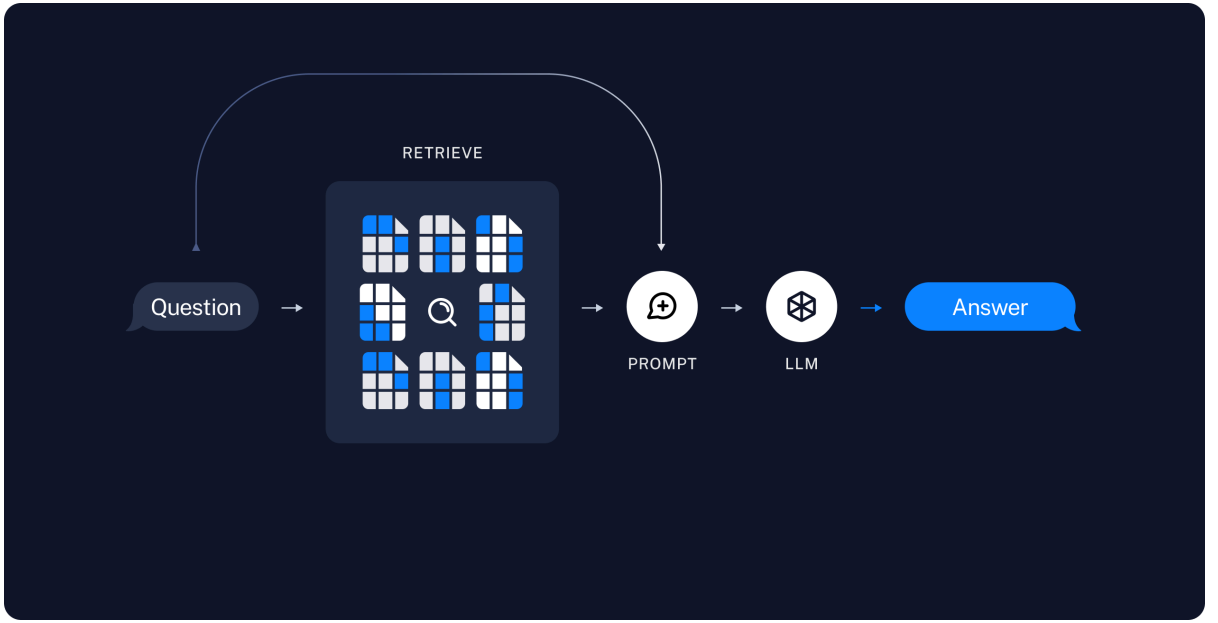


LangChain for Chat Over Documents



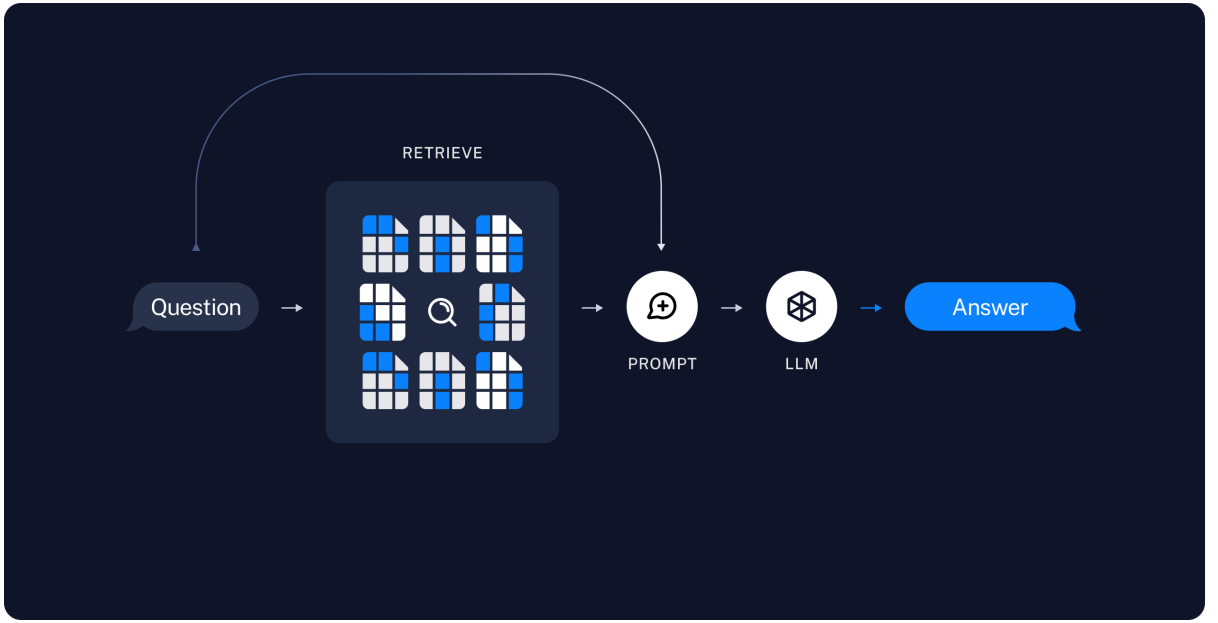
- Retrieval Piepeline

LangChain for Chat Over Documents



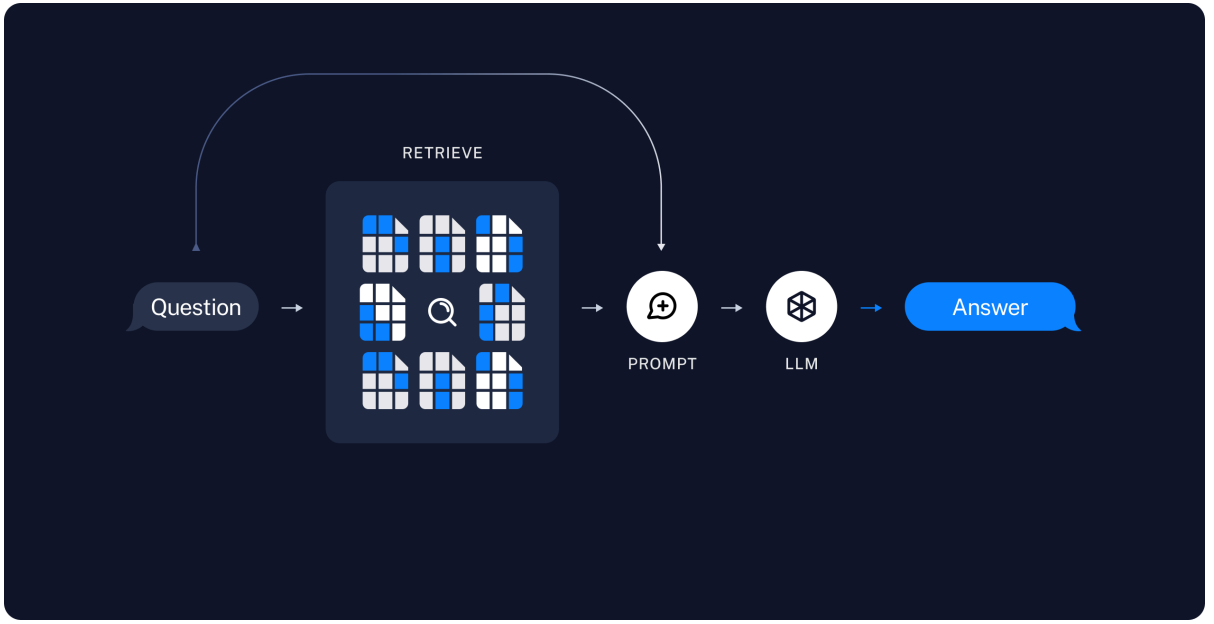
- Retrieval Piepeline
  - Input Question

LangChain for Chat Over Documents



- Retrieval Piepeline
  - Input Question
  - Retrieve Relevant Documents

LangChain for Chat Over Documents



- Retrieval Piepeline
  - Input Question
  - Retrieve Relevant Documents
  - LLM uses the prompt question + retrieved data to produce a final answer

LangChain for Chat Over Documents

- Sample Code

```
from langchain import hub
from langchain_community.vectorstores import Chroma
from langchain_openai import ChatOpenAI, OpenAIEmbeddings
from langchain.document_loaders import PyPDFLoader
from langchain.chains import RetrievalQA

pdf_path = "path-to-pdf.pdf"
loader = PyPDFLoader(pdf_path) # LOAD
pdf_docs = loader.load_and_split() # SPLIT
embeddings = OpenAIEmbeddings() # EMBED
vectordb = Chroma.from_documents(pdf_docs, embedding=embeddings) # STORE
retriever = vectordb.as_retriever()
llm = ChatOpenAI(model="gpt-3.5-turbo-0125")
pdf_qa = RetrievalQA.from_llm(llm=llm, retriever=retriever) # RETRIEVE
pdf_qa.invoke("What is this paper about?") # ANSWER
```

Notebook Demo 4.1 - Q&A with LangChain

Q&A & Summary

- **RAG** = **R**etrieval **A**ugmented **G**eneration
- **RAG** is about connecting LLMs to documents like PDFs, Text files, HTML, etc.
- **Embeddings** are vectorized representations of text that capture content and meaning.
- **Vector DBs** are used to store and retrieve embeddings.
- **RAG** systems with LangChain are built using a pipeline that includes loading, splitting, embedding, and storing documents.
- **Optional Exercise During Q&A**

```
Create a simple RAG system with LangChain that can answer questions about pdfs or csvs.
```

Break

Practical Session

References

1. What are Transformers and How do They Work? By Serrano Academy
2. OpenAI Docs
3. OpenAI's Prompt Engineering Guide
4. OpenAI's Fine Tuning Docs
5. LangChain Docs
6. ReACT Paper