Getting Started with LangChain

Lucas Soares

21-02-2024

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'

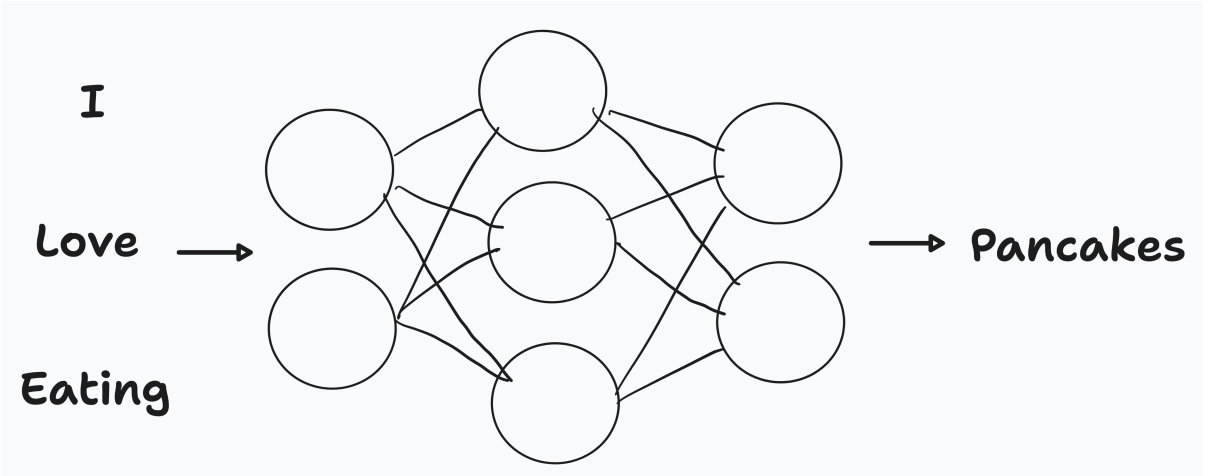
- The presentation will be organized into the following structure:
  - **Presentation Block**
  - **Notebook Demo**
  - **Quick Q&A + Summary**
  - **Optional Exercise** During Q&A (for those that don't have questions and want to try something out)

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

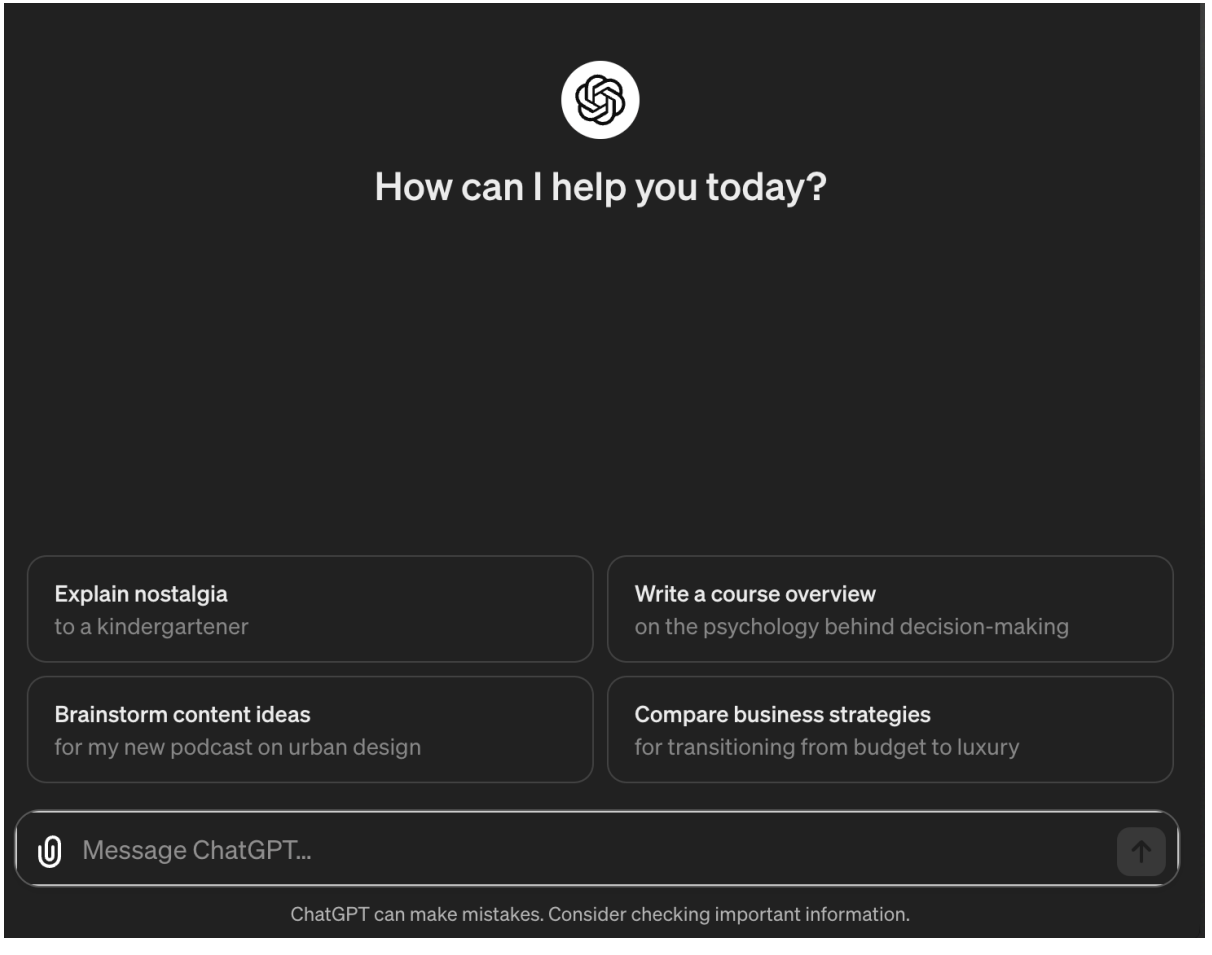
Large Language Models

Large Language Models Predict the Next Word



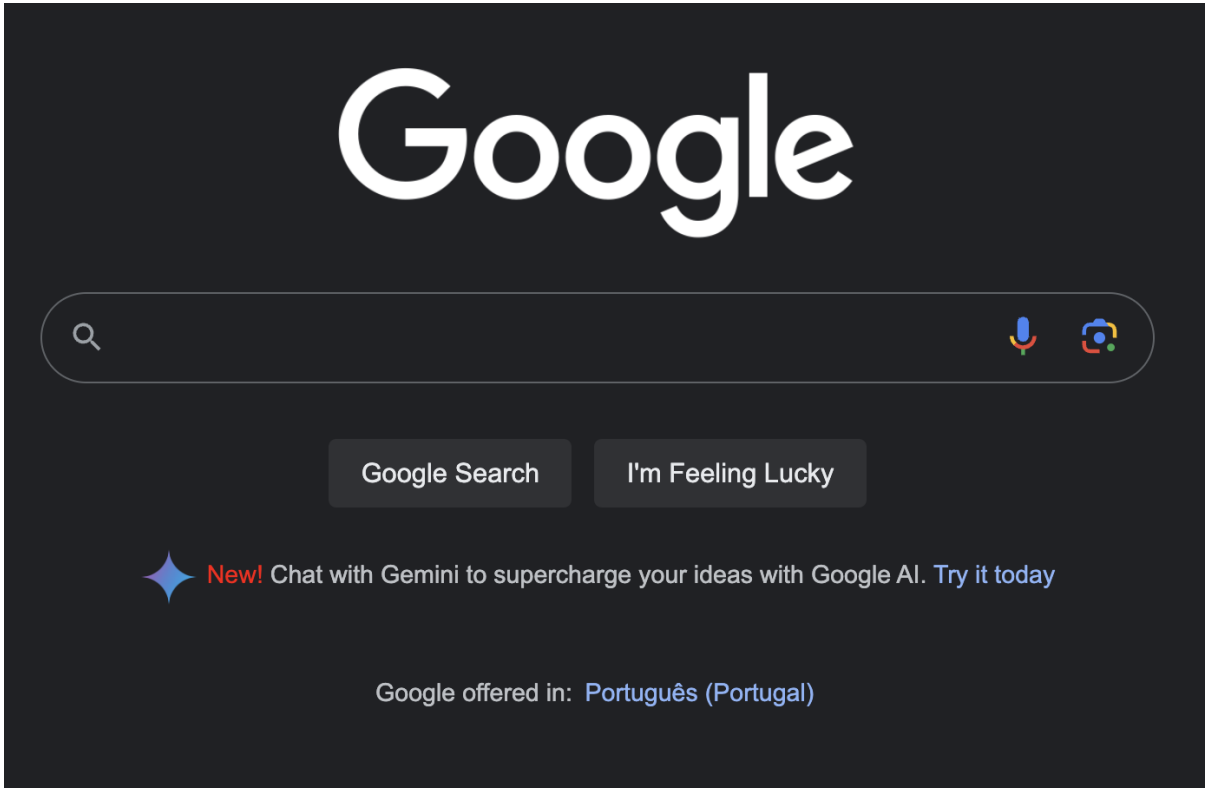
Applications of Large Language Models

- Conversational ChatBots



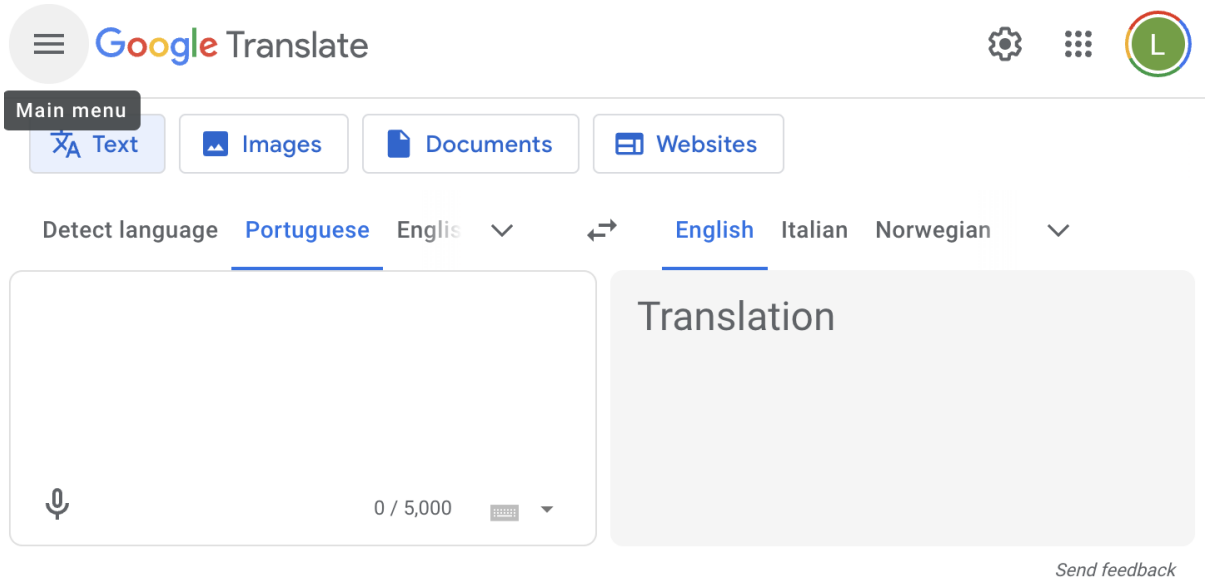
Applications of Large Language Models

- Search Engines



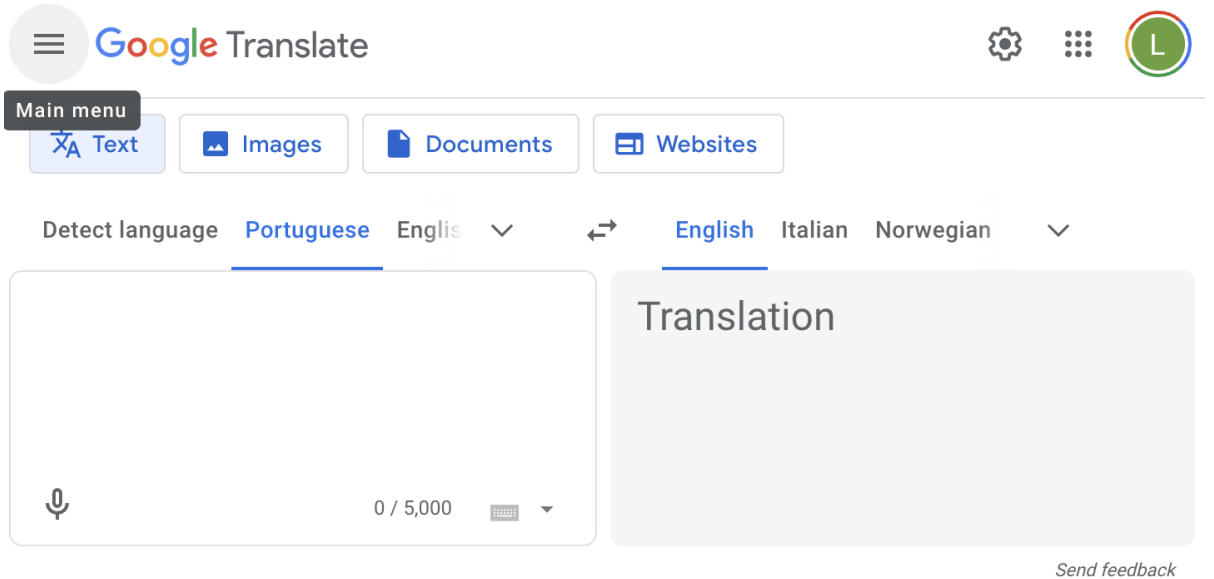
Applications of Large Language Models

- Translation



Applications of Large Language Models

- Translation



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

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
- Meaning: LangChain gives you building blocks for building interesting and powerful LLM applications

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)
```

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'
```

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()
```

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("""
Write 5 concepts that are fundamental to learn about {topic}.
                                          """)
chain = prompt | llm | output_parser
chain.invoke({"topic": "Artificial Neural Networks"})
```

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

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

Notebook Demo - Intro to LangChain

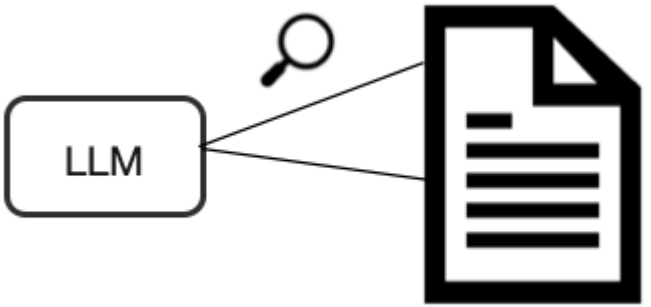
Q&A & Summary

- **LLMs can predict the next word in a sequence**. ("I Like eating...? ;P ")
- **LangChain framework:** eases the creation of LLM-based applications, featuring chains and the following basic components:
  - **Models**: Abstractions over LLM APIs (e.g ChatGPT).
  - **Prompt Templates**: Abstractions over prompts (makes them dynamic).
  - **Output Parsers**: Converts LLM outputs into usable formats (e.g string, json).
- **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**

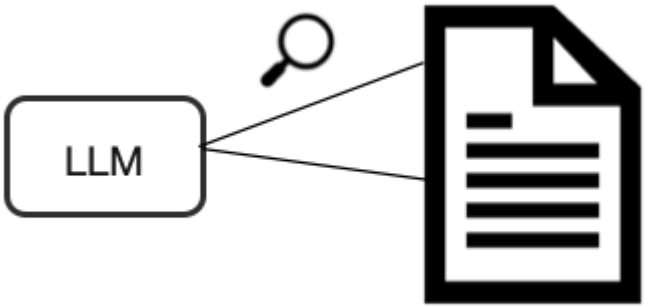
  ```
  Create a simple chain for summarization of content.
  ```

Break 5 minutes

LangChain for Chat Over Documents

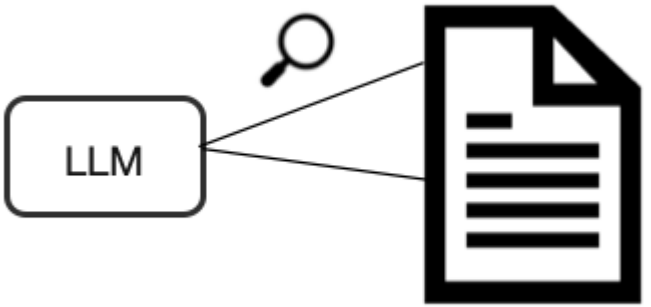


LangChain for Chat Over Documents



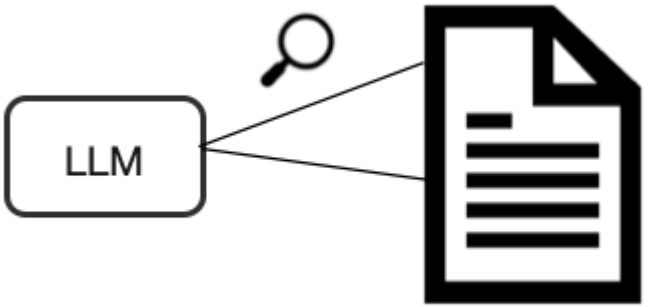
- RAG = **R**etrieval **A**ugmented **G**eneration

LangChain for Chat Over Documents



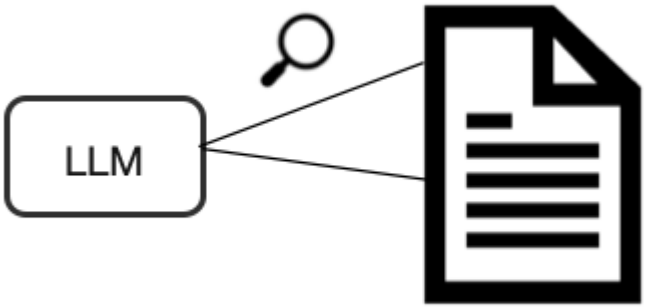
- RAG = **R**etrieval **A**ugmented **G**eneration
- It's about connecting LLMs to documents like PDFs, Text files, HTML, etc.

LangChain for Chat Over Documents



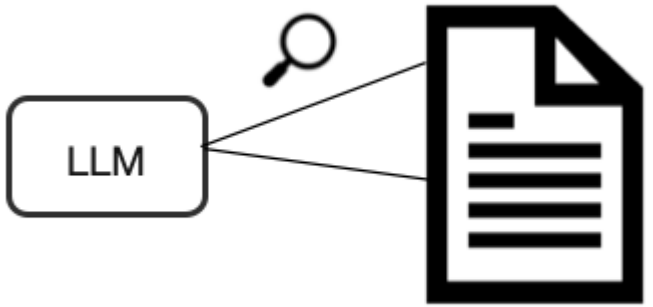
- RAG = **R**etrieval **A**ugmented **G**eneration
- It's about connecting LLMs to documents like PDFs, Text files, HTML, etc.
- How do we get around the context length limitations of LLMs?

LangChain for Chat Over Documents



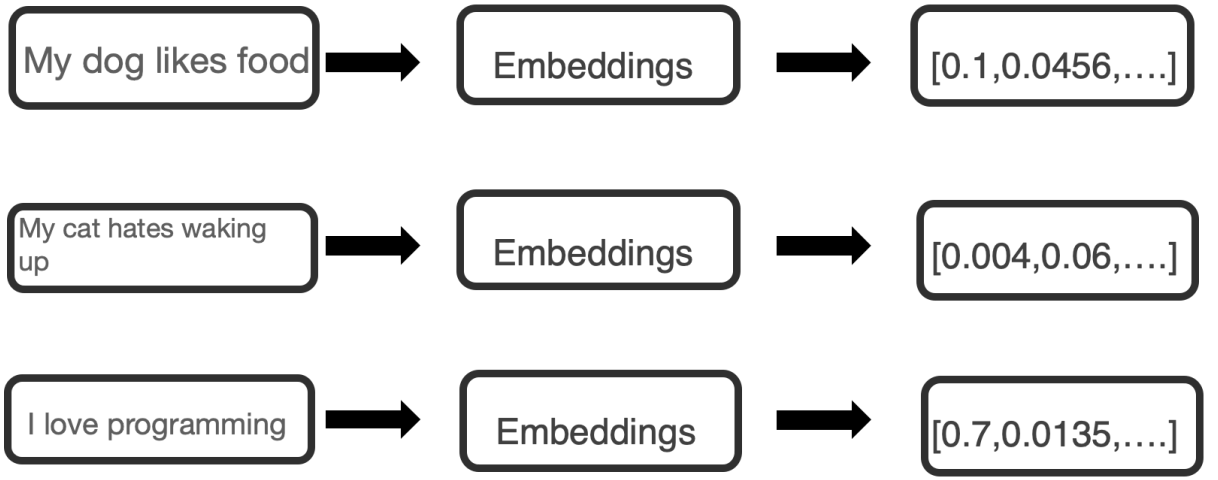
- RAG = **R**etrieval **A**ugmented **G**eneration
- It's about connecting LLMs to documents like PDFs, Text files, HTML, etc.
- How do we get around the context length limitations of LLMs?
- Quick Answer is **Embeddings**!

LangChain for Chat Over Documents

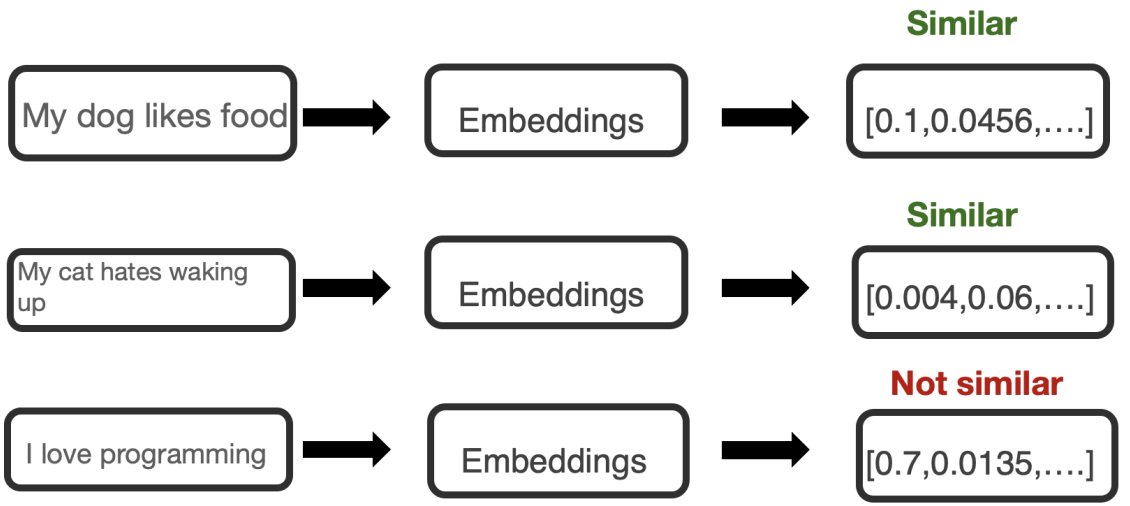


- Embeddings are vectorized representations of text

LangChain for Chat Over Documents

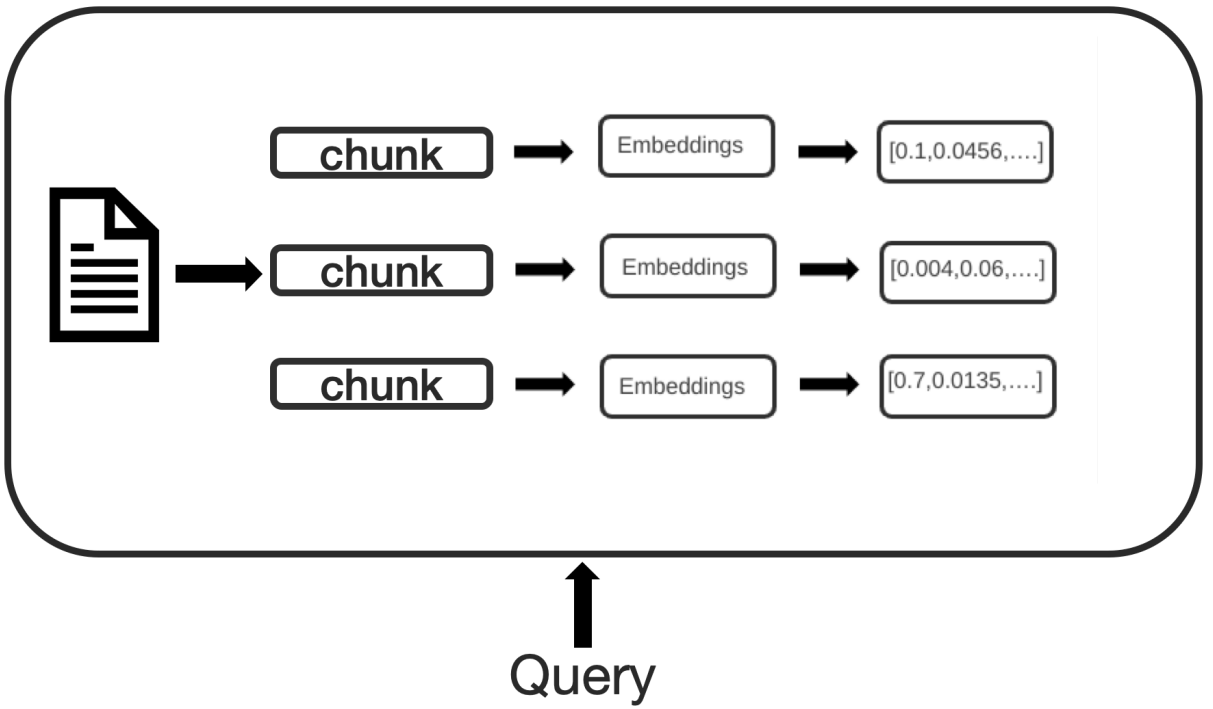


LangChain for Chat Over Documents



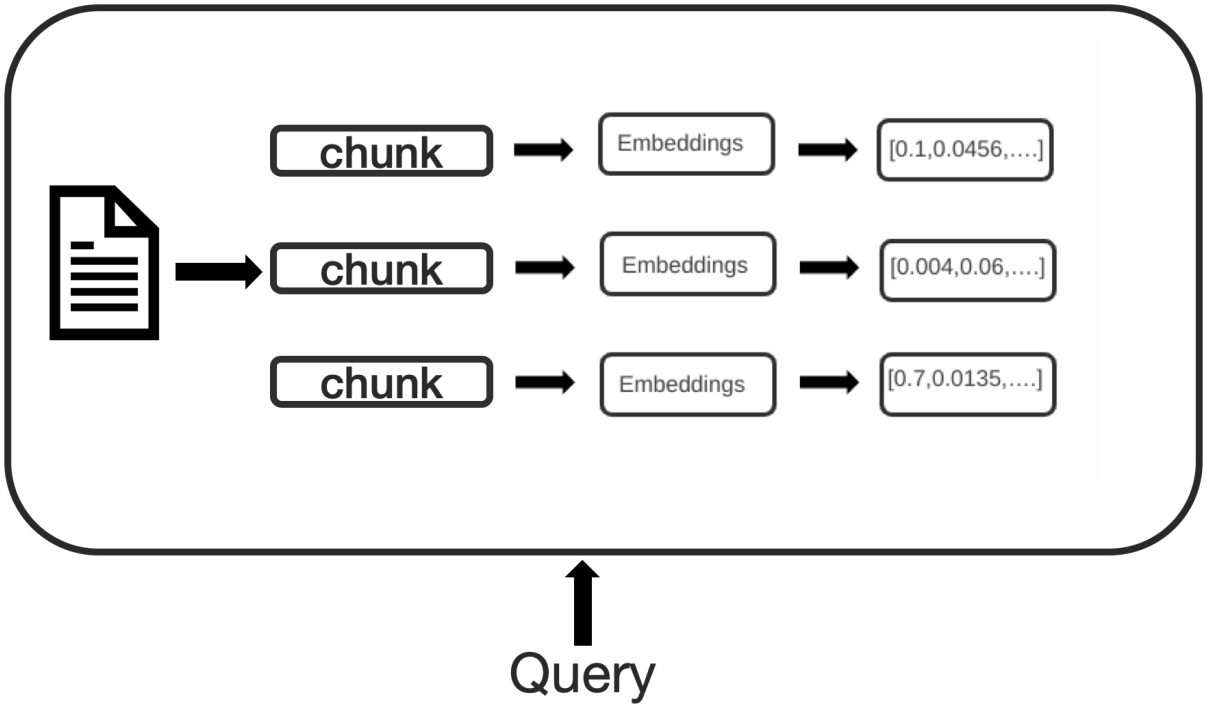
- Embeddings capture content and meaning

LangChain for Chat Over Documents



- Embeddings capture content and meaning
- Vector DBs

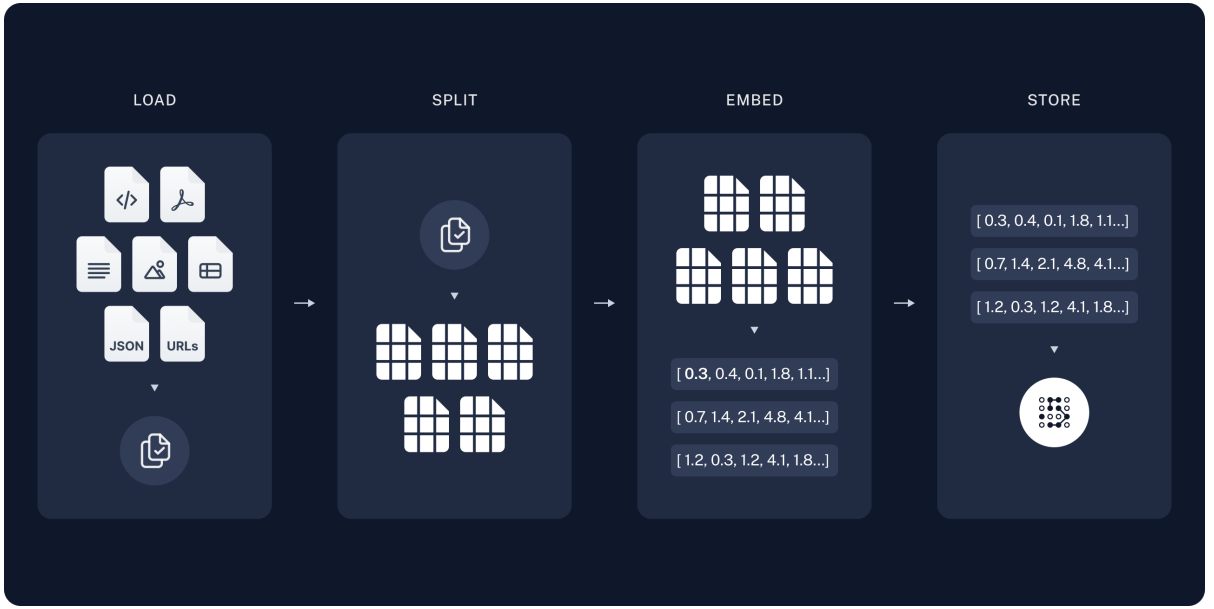
LangChain for Chat Over Documents



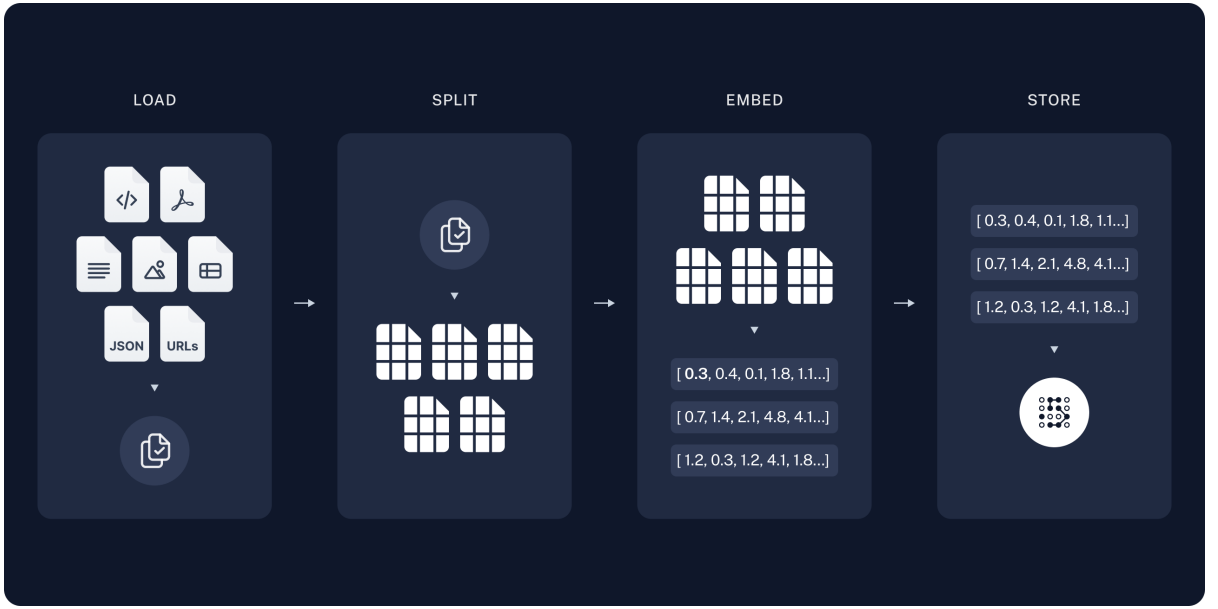
- Embeddings capture content and meaning
- Vector DBs
- How to build RAG systems with LangChain?

LangChain for Chat Over Documents

LangChain for Chat Over Documents

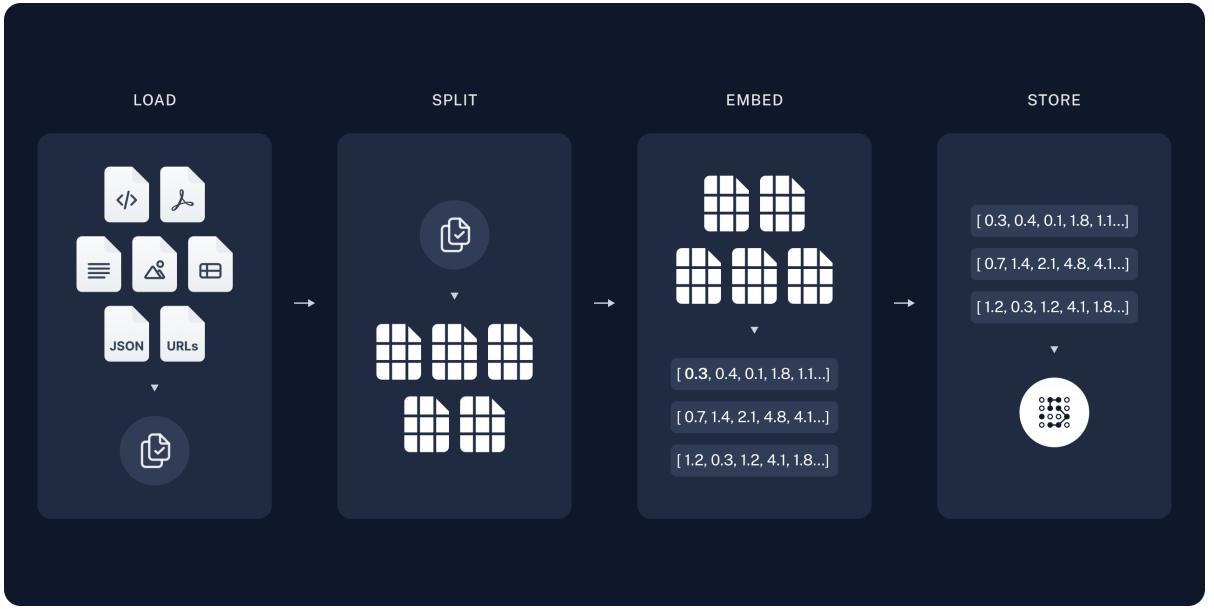


LangChain for Chat Over Documents



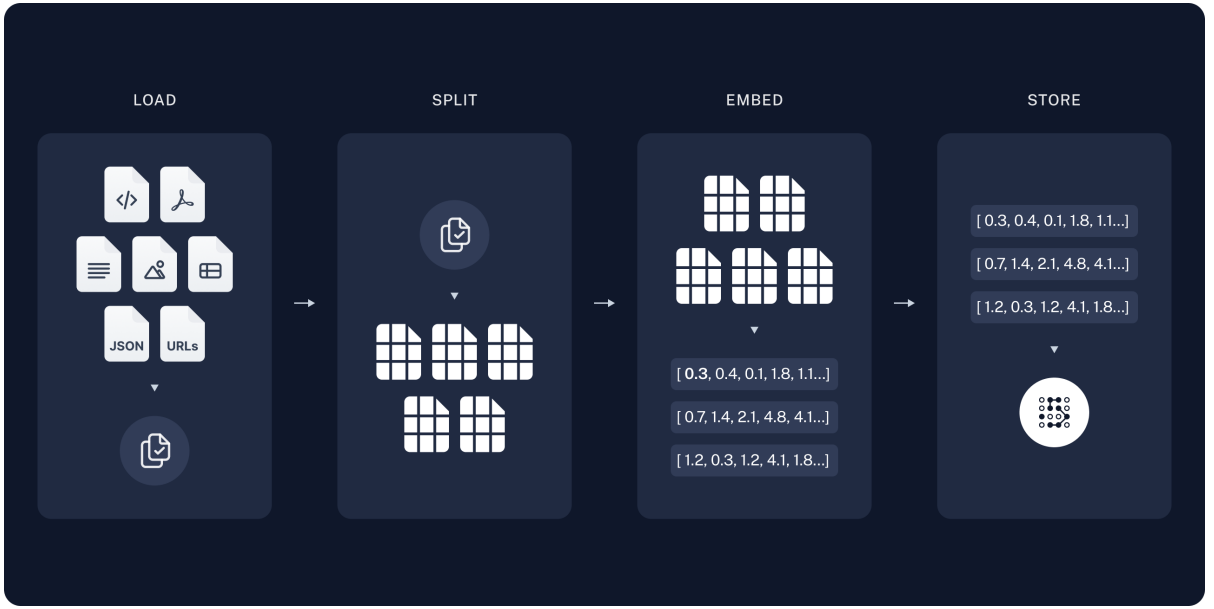
- Load

LangChain for Chat Over Documents



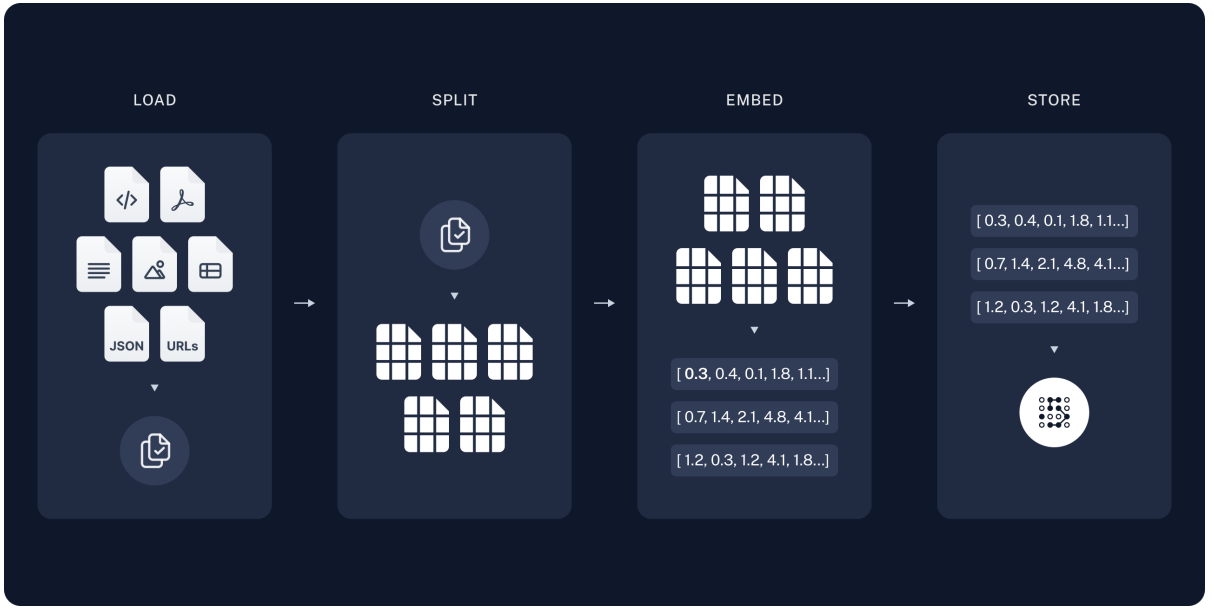
- Load
- Split

LangChain for Chat Over Documents



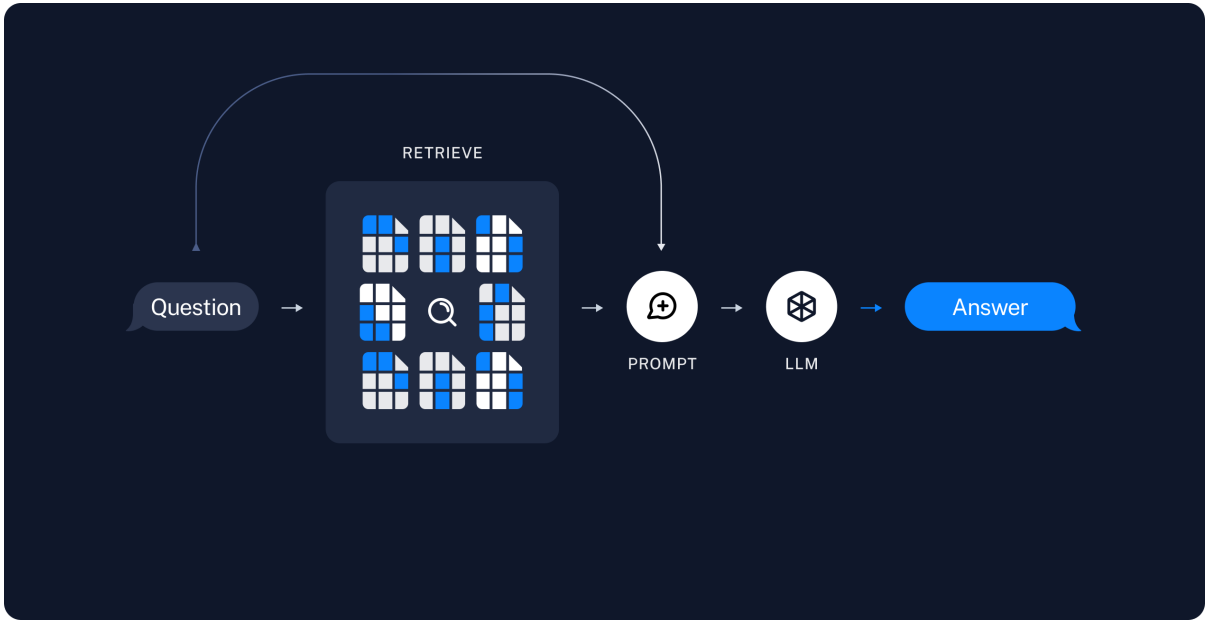
- Load
- Split
- Embed

LangChain for Chat Over Documents

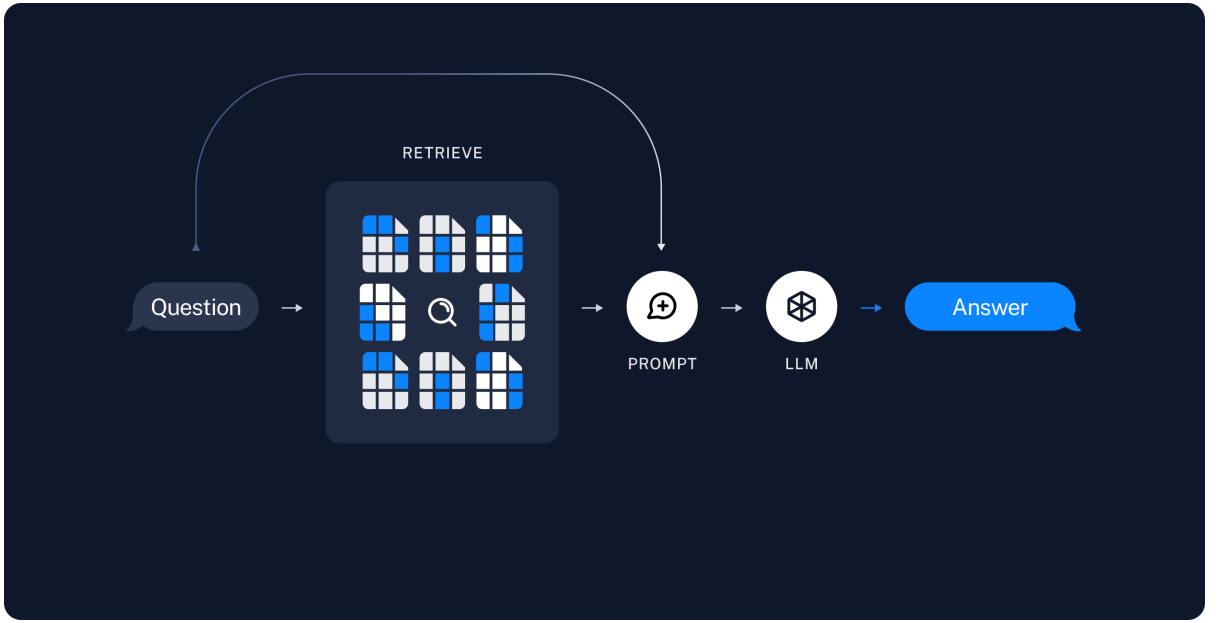


- Load
- Split
- Embed
- Store

LangChain for Chat Over Documents

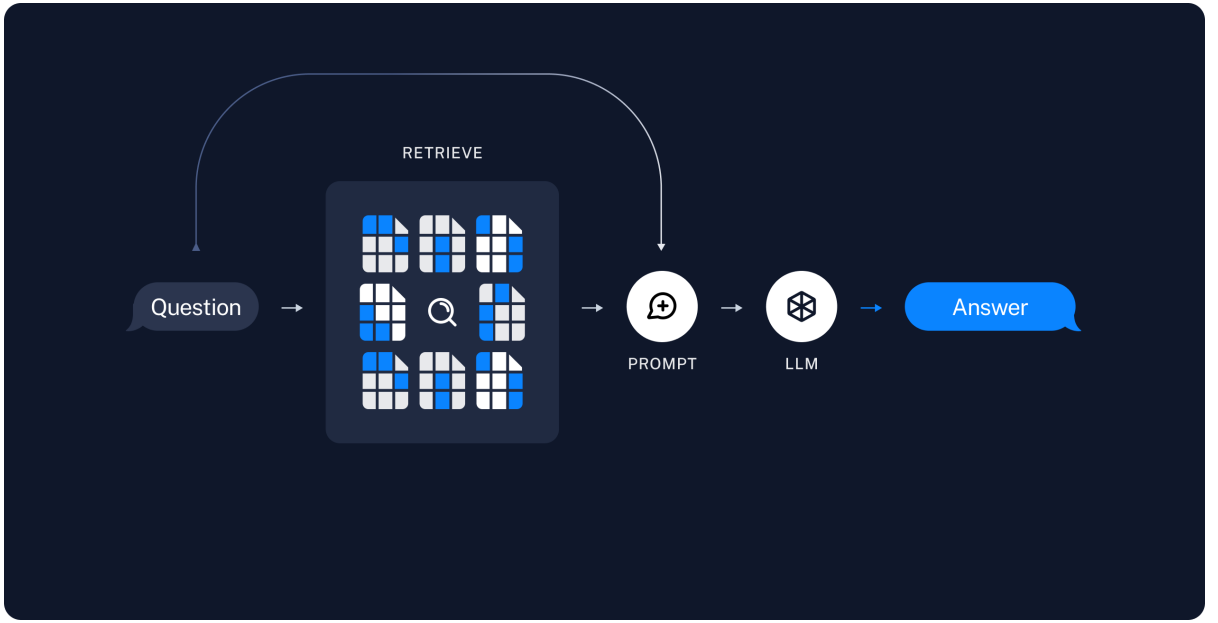


LangChain for Chat Over Documents



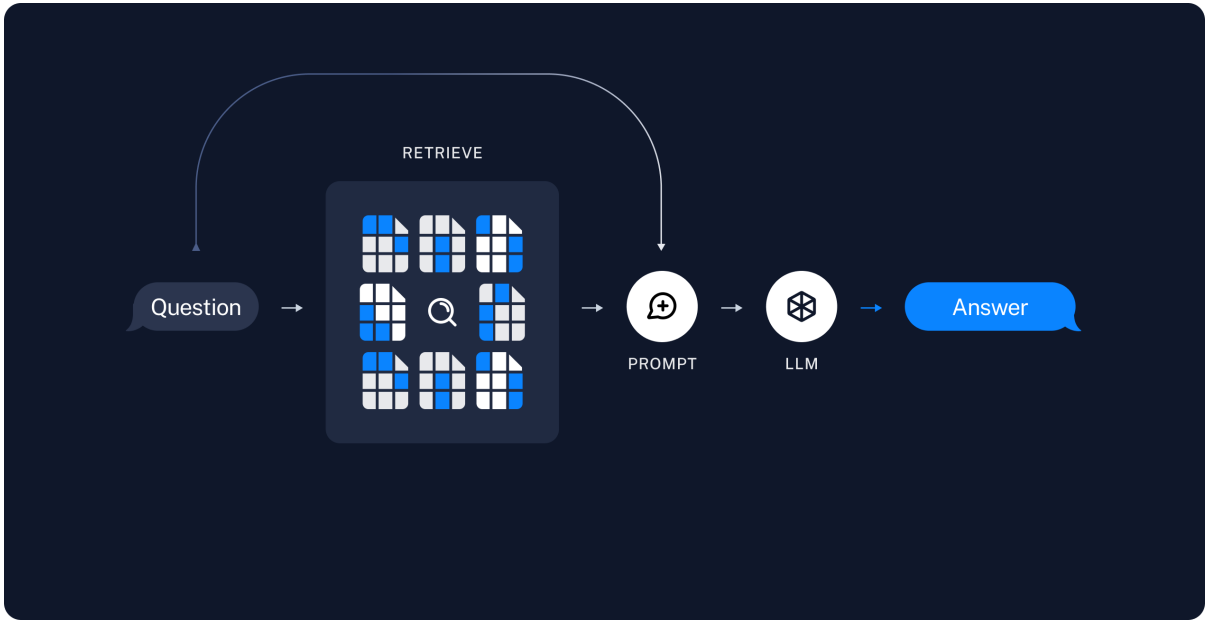
- Retrieval Piepeline

LangChain for Chat Over Documents



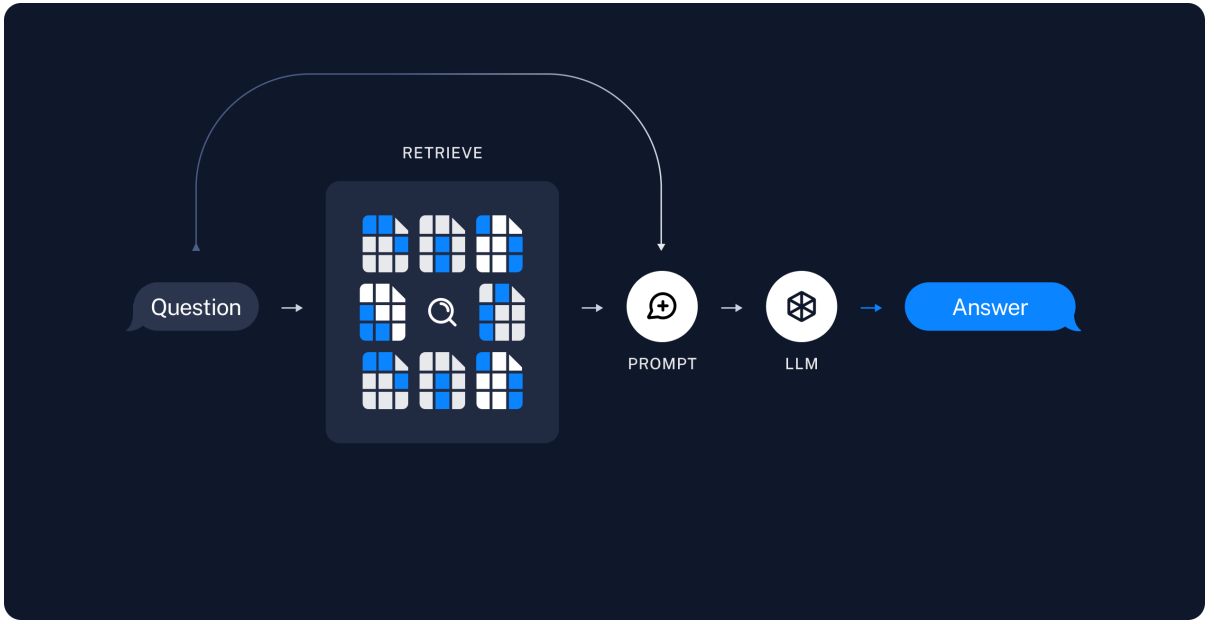
- Retrieval Piepeline
  - Input Question

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
```

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 - 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 5 minutes

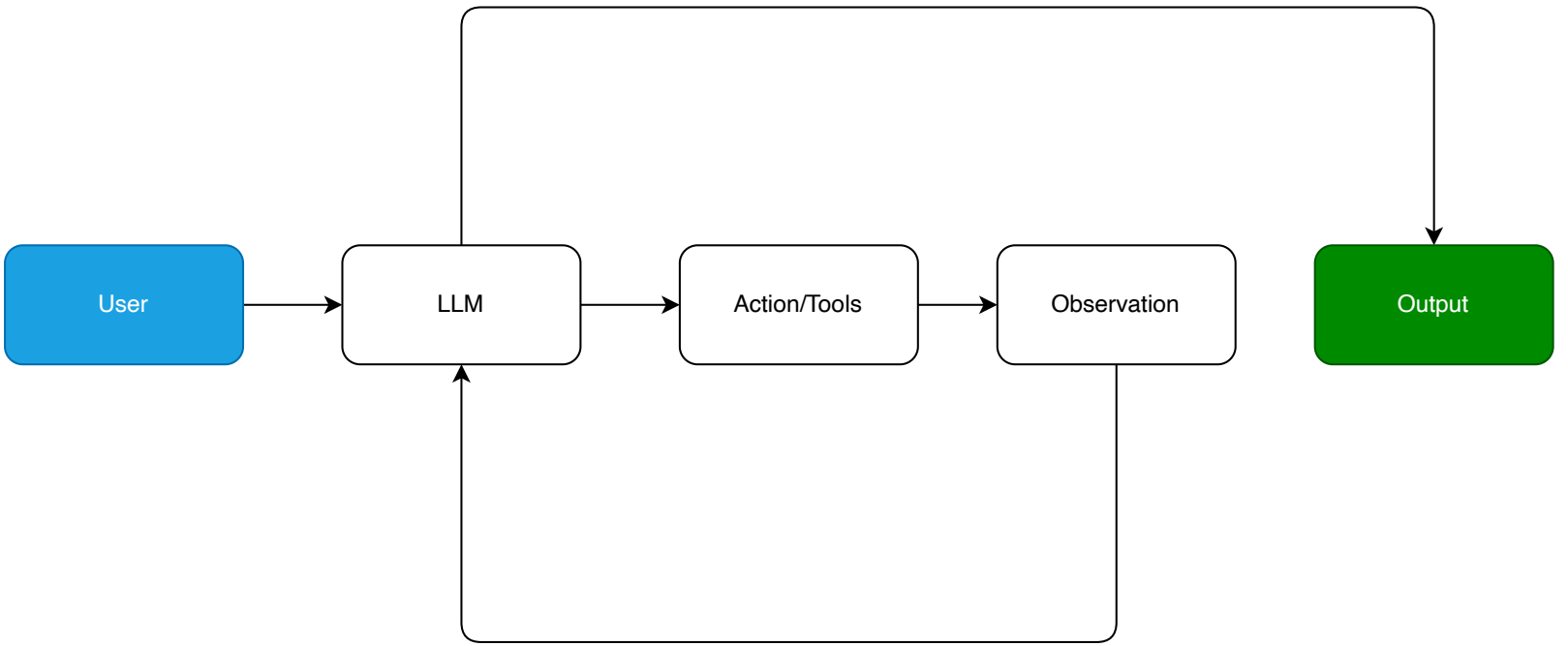
Building Agents with LangChain

Building Agents with LangChain

The Agent Loop

Building Agents with LangChain

The Agent Loop



Key LangChain Components for Agents

Schema

Key LangChain Components for Agents

Schema

- LangChain provides many abstractions for ease of use

Key LangChain Components for Agents

Schema

- LangChain provides many abstractions for ease of use
- **AgentAction**: Represents the action an agent should take.

Key LangChain Components for Agents

Schema

- LangChain provides many abstractions for ease of use
- **AgentAction**: Represents the action an agent should take.
- **AgentFinish**: Represents the final result to return to the user.

Key LangChain Components for Agents

Schema

- LangChain provides many abstractions for ease of use
- **AgentAction**: Represents the action an agent should take.
- **AgentFinish**: Represents the final result to return to the user.
- **Intermediate Steps**: Previous actions and outputs for the current agent run.

Key LangChain Components for Agents

Schema

- LangChain provides many abstractions for ease of use
- **AgentAction**: Represents the action an agent should take.
- **AgentFinish**: Represents the final result to return to the user.
- **Intermediate Steps**: Previous actions and outputs for the current agent run.
- **Agent**: Chain responsible for deciding the next step, powered by a language model.

Agent Inputs and Outputs

Agent Inputs

Agent Inputs and Outputs

Agent Inputs

- Key-value mapping.

Agent Inputs and Outputs

Agent Inputs

- Key-value mapping.
- Required key: `intermediate_steps`.

Agent Inputs and Outputs

Agent Inputs

- Key-value mapping.
- Required key: `intermediate_steps`.

Agent Outputs

Agent Inputs and Outputs

Agent Inputs

- Key-value mapping.
- Required key: `intermediate_steps`.

Agent Outputs

- Next actions or final response (AgentActions or AgentFinish).

Agent Inputs and Outputs

Agent Inputs

- Key-value mapping.
- Required key: `intermediate_steps`.

Agent Outputs

- Next actions or final response (AgentActions or AgentFinish).
- Handled by the output parser.

AgentExecutor

- The agent executor is the runtime for an agent.

AgentExecutor

- The agent executor is the runtime for an agent.
- It calls the agent, executes the actions it chooses, passes the action outputs back to the agent, and repeats.

AgentExecutor

- The agent executor is the runtime for an agent.
- It calls the agent, executes the actions it chooses, passes the action outputs back to the agent, and repeats.
- In pseudocode, this looks roughly like:

```
next_action = agent.get_action(...)
while next_action != AgentFinish:
    observation = run(next_action)
    next_action = agent.get_action(..., next_action, observation)
return next_action
```

AgentExecutor

- The agent executor is the runtime for an agent.
- It calls the agent, executes the actions it chooses, passes the action outputs back to the agent, and repeats.
- In pseudocode, this looks roughly like:

```
next_action = agent.get_action(...)
while next_action != AgentFinish:
    observation = run(next_action)
    next_action = agent.get_action(..., next_action, observation)
return next_action
```

- Runtime handles things like:
    - Handling cases where the agent selects a non-existent tool

AgentExecutor

- The agent executor is the runtime for an agent.
- It calls the agent, executes the actions it chooses, passes the action outputs back to the agent, and repeats.
- In pseudocode, this looks roughly like:

```
next_action = agent.get_action(...)
while next_action != AgentFinish:
    observation = run(next_action)
    next_action = agent.get_action(..., next_action, observation)
return next_action
```

- Runtime handles things like:
  - Handling cases where the agent selects a non-existent tool
  - Handling cases where the tool errors

AgentExecutor

- The agent executor is the runtime for an agent.
- It calls the agent, executes the actions it chooses, passes the action outputs back to the agent, and repeats.
- In pseudocode, this looks roughly like:

```
next_action = agent.get_action(...)
while next_action != AgentFinish:
    observation = run(next_action)
    next_action = agent.get_action(..., next_action, observation)
return next_action
```

- Runtime handles things like:
  - Handling cases where the agent selects a non-existent tool
  - Handling cases where the tool errors
  - Handling cases where the agent produces output that cannot be parsed into a tool invocation

AgentExecutor

- The agent executor is the runtime for an agent.
- It calls the agent, executes the actions it chooses, passes the action outputs back to the agent, and repeats.
- In pseudocode, this looks roughly like:

```
next_action = agent.get_action(...)
while next_action != AgentFinish:
    observation = run(next_action)
    next_action = agent.get_action(..., next_action, observation)
return next_action
```

- Runtime handles things like:
  - Handling cases where the agent selects a non-existent tool
  - Handling cases where the tool errors
  - Handling cases where the agent produces output that cannot be parsed into a tool invocation
  - Logging and observability at all levels (agent decisions, tool calls) to stdout and/or to LangSmith.

Tools in LangChain

Tools in LangChain

- Functions that an agent can call.

Tools in LangChain

- Functions that an agent can call.
- Consists of:

Tools in LangChain

- Functions that an agent can call.
- Consists of:
    - Input schema for the tool.

Tools in LangChain

- Functions that an agent can call.
- Consists of:
  - Input schema for the tool.
  - Function to run.

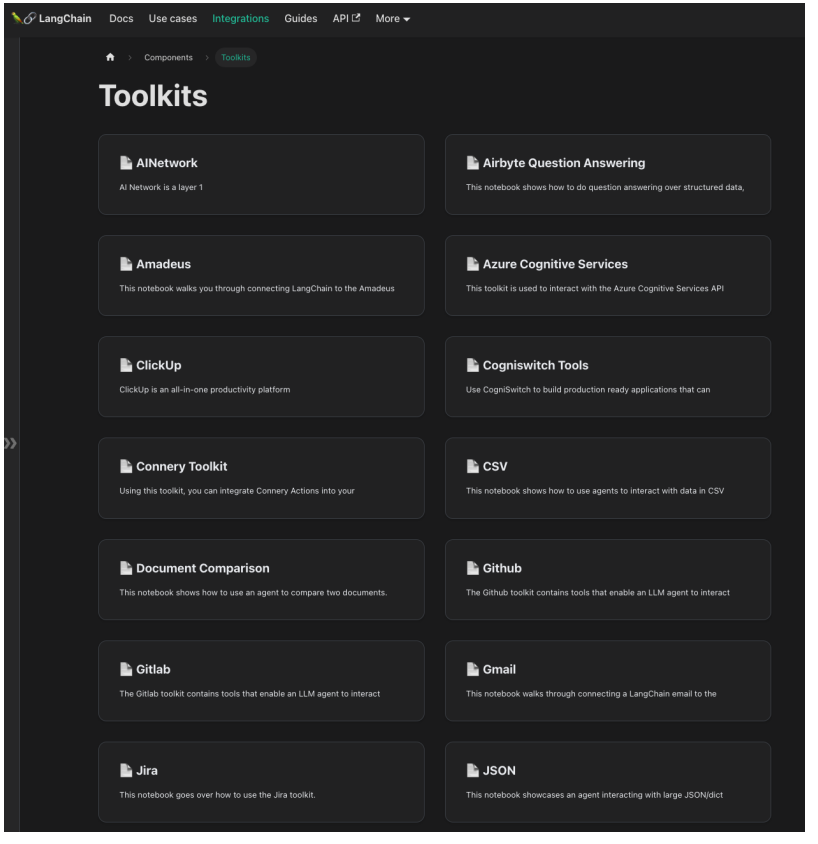
Tools in LangChain

- Functions that an agent can call.
- Consists of:
  - Input schema for the tool.
  - Function to run.
- Important for building a working agent.

Langchain Toolkits

Langchain Toolkits

- LangChain provides a wide set of toolkits.
- Groups of 3-5 tools for specific objectives.
- Example: GitHub toolkit for interacting with GitHub.



Let's Build Agents!

Notebook Demo - Building LLM Agents with LangChain; Github Agent; Research Assistant

Q&A & Summary

- **LangChain** provides abstractions for building agents, including AgentAction, AgentFinish, and Agent.
- **AgentExecutor** is the runtime for an agent, handling agent decisions, tool calls, and observability.
- **Tools** in LangChain are functions that an agent can call, consisting of an input schema and a function to run.
- **LangChain** provides a wide set of toolkits, groups of 3-5 tools for specific objectives, an example is GitHub toolkit for interacting with GitHub.
- **Optional Exercise During Q&A**

```
Create a simple agent that can create a schedule for you given a table of tasks and deac
table format = task | date
```

Break 5 minutes

LangServe Super Quick Intro

- LangServe helps developers deploy LangChain runnables and chains as a REST API.

LangServe Super Quick Intro

- LangServe helps developers deploy LangChain runnables and chains as a REST API.
- It is integrated with FastAPI and uses pydantic for data validation.

LangServe Super Quick Intro

- LangServe helps developers deploy LangChain runnables and chains as a REST API.
- It is integrated with FastAPI and uses pydantic for data validation.
- It provides a client that can be used to call into runnables deployed on a server.

LangServe Super Quick Intro

- LangServe helps developers deploy LangChain runnables and chains as a REST API.
- It is integrated with FastAPI and uses pydantic for data validation.
- It provides a client that can be used to call into runnables deployed on a server.
- Input and Output schemas automatically inferred from your LangChain object, and enforced on every API call, with rich error messages

LangServe Super Quick Intro

- LangServe helps developers deploy LangChain runnables and chains as a REST API.
- It is integrated with FastAPI and uses pydantic for data validation.
- It provides a client that can be used to call into runnables deployed on a server.
- Input and Output schemas automatically inferred from your LangChain object, and enforced on every API call, with rich error messages
- Efficient /invoke/, /batch/ and /stream/ endpoints with support for many concurrent requests on a single server

LangServe Super Quick Intro

- LangServe helps developers deploy LangChain runnables and chains as a REST API.
- It is integrated with FastAPI and uses pydantic for data validation.
- It provides a client that can be used to call into runnables deployed on a server.
- Input and Output schemas automatically inferred from your LangChain object, and enforced on every API call, with rich error messages
- Efficient /invoke/, /batch/ and /stream/ endpoints with support for many concurrent requests on a single server
- Playground page at /playground/ with streaming output and intermediate steps

LangServe Super Quick Intro

- LangServe helps developers deploy LangChain runnables and chains as a REST API.
- It is integrated with FastAPI and uses pydantic for data validation.
- It provides a client that can be used to call into runnables deployed on a server.
- Input and Output schemas automatically inferred from your LangChain object, and enforced on every API call, with rich error messages
- Efficient /invoke/, /batch/ and /stream/ endpoints with support for many concurrent requests on a single server
- Playground page at /playground/ with streaming output and intermediate steps
- Built-in (optional) tracing to LangSmith, just add your API key (see Instructions)

LangServe Super Quick Intro

- LangServe helps developers deploy LangChain runnables and chains as a REST API.
- It is integrated with FastAPI and uses pydantic for data validation.
- It provides a client that can be used to call into runnables deployed on a server.
- Input and Output schemas automatically inferred from your LangChain object, and enforced on every API call, with rich error messages
- Efficient /invoke/, /batch/ and /stream/ endpoints with support for many concurrent requests on a single server
- Playground page at /playground/ with streaming output and intermediate steps
- Built-in (optional) tracing to LangSmith, just add your API key (see Instructions)
- Use the client SDK to call a LangServe server as if it was a Runnable running locally (or call the HTTP API directly)

LangServe Super Quick Intro

- LangServe helps developers deploy LangChain runnables and chains as a REST API.
- It is integrated with FastAPI and uses pydantic for data validation.
- It provides a client that can be used to call into runnables deployed on a server.
- Input and Output schemas automatically inferred from your LangChain object, and enforced on every API call, with rich error messages
- Efficient /invoke/, /batch/ and /stream/ endpoints with support for many concurrent requests on a single server
- Playground page at /playground/ with streaming output and intermediate steps
- Built-in (optional) tracing to LangSmith, just add your API key (see Instructions)
- Use the client SDK to call a LangServe server as if it was a Runnable running locally (or call the HTTP API directly)

Final Q&A & Summary

- **LangServe** helps developers deploy LangChain runnables and chains as a REST API.
- It is integrated with FastAPI and uses pydantic for data validation.

References

- LangChain Intro Docs
- LangChain Documentation
- Gen Agents
- WebGPT
- OpenAI
- OpenAI Function Calling
- AutoGPT
- GPT-Engineer
- BabyAGI
- Karpathy on Agents
- ReACT Paper