Getting Started with LangChain

Lucas Soares

27-02-2025

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'

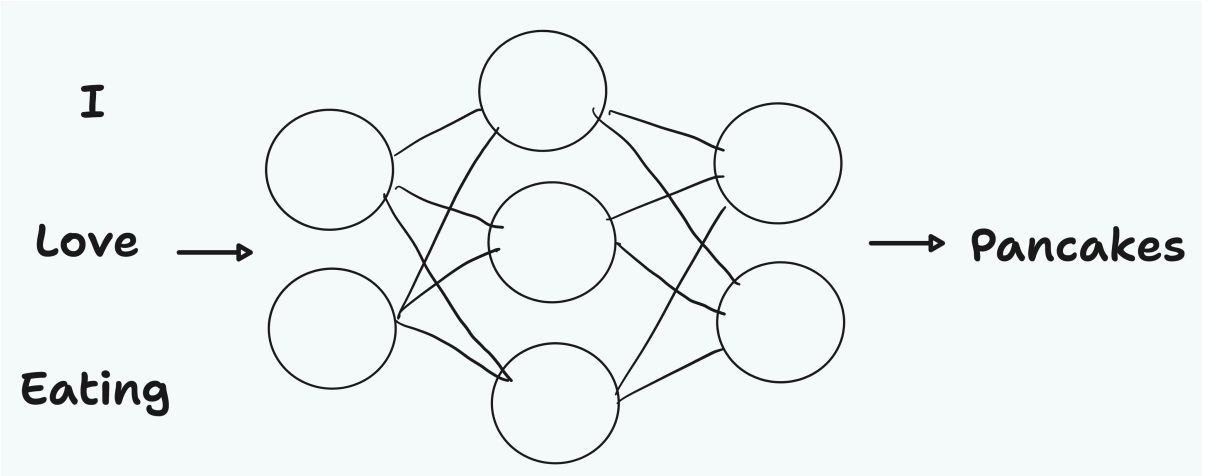
- The presentation will be organized into the following structure:
  - **Presentation Block**
  - **Notebook Demo**
  - **Quick Q&A + Summary**
  - **Optional Exercise** During Q&A (for those that don't have questions and want to try something out)

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

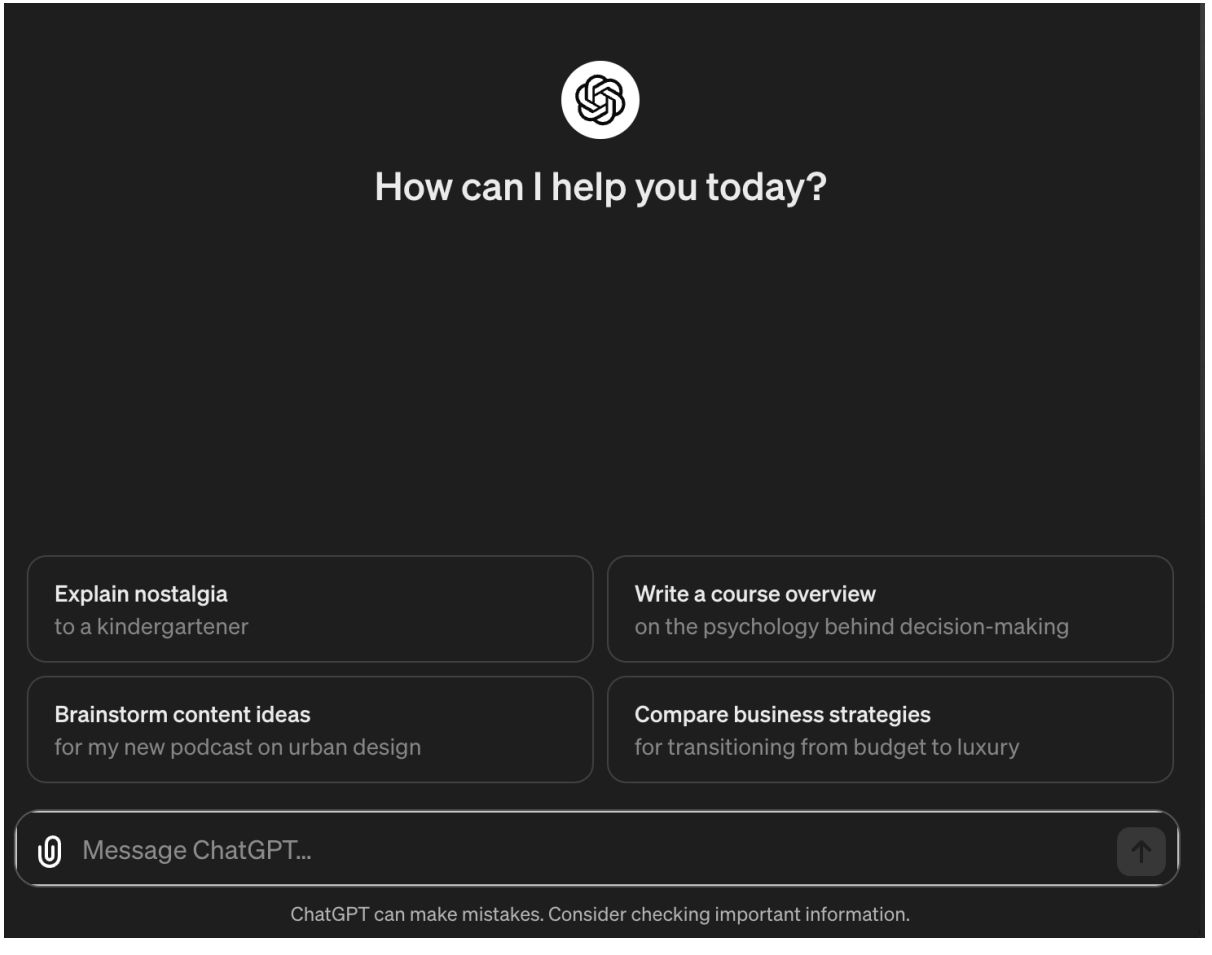
Large Language Models

Large Language Models Predict the Next Word



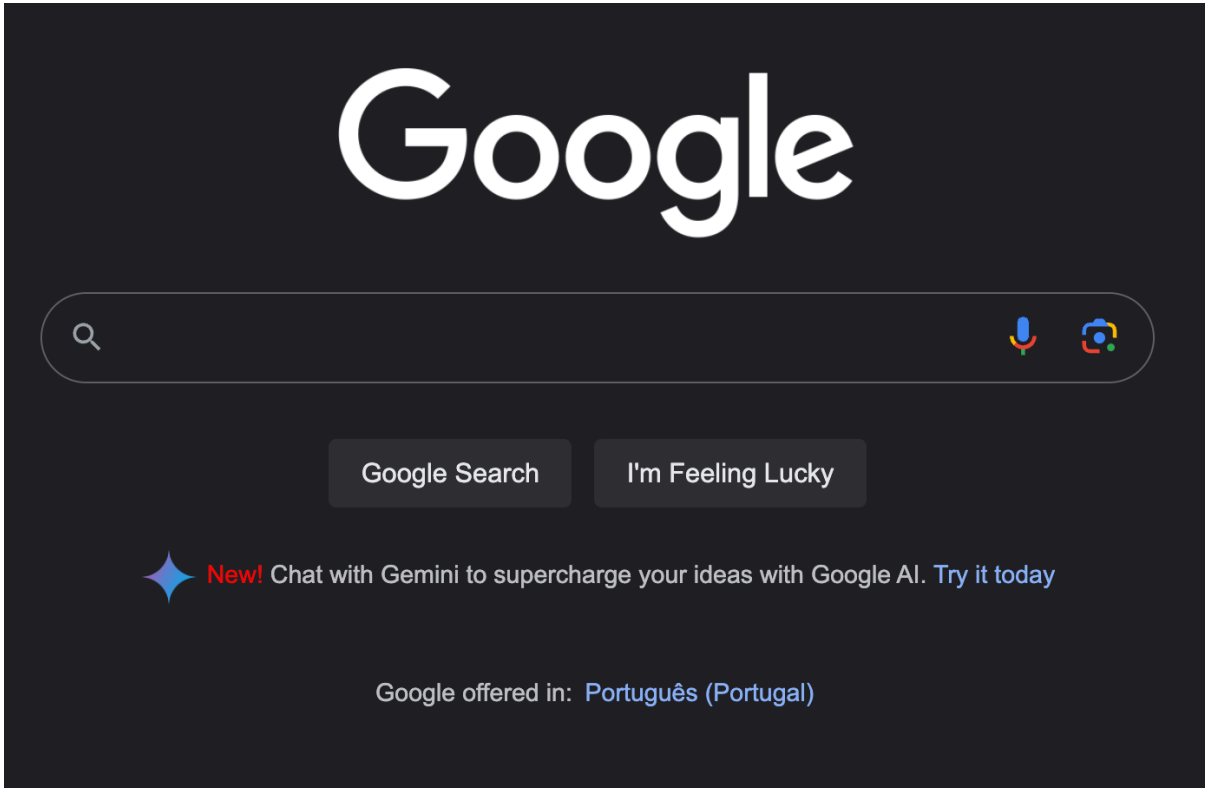
Applications of Large Language Models

- Conversational ChatBots



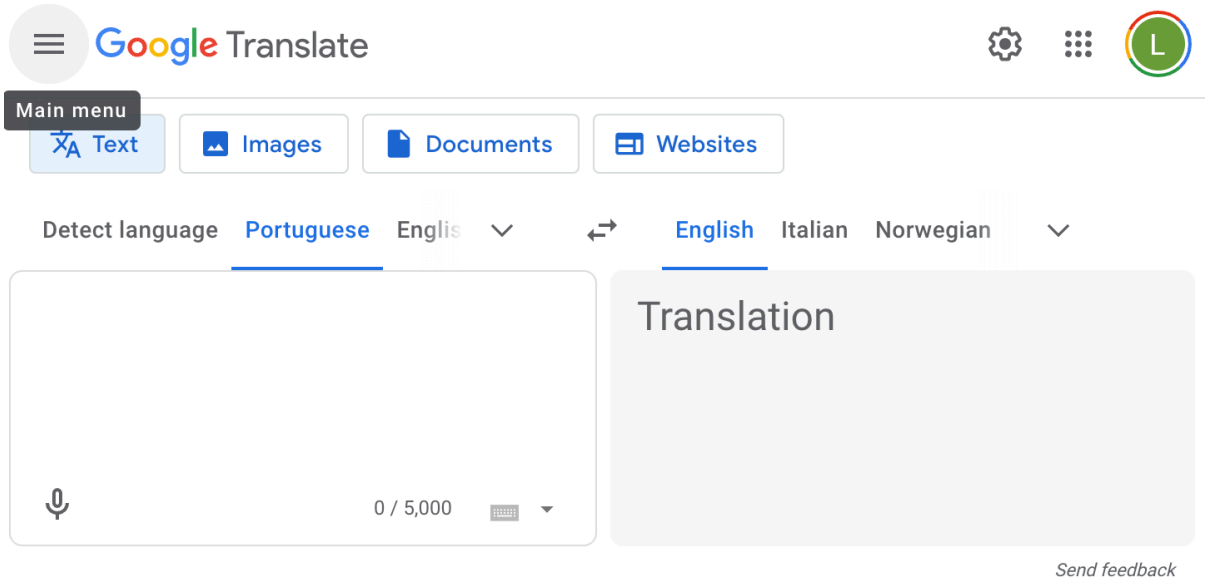
Applications of Large Language Models

- Search Engines



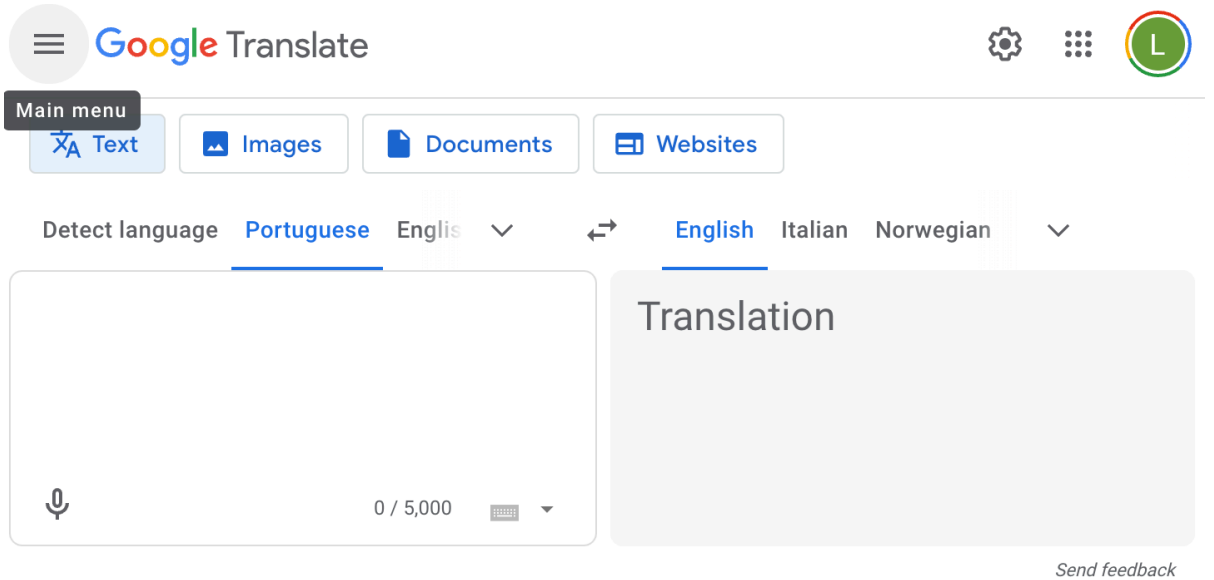
Applications of Large Language Models

- 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

Whiteboard Explain - Building Blocks of Working with LLMs

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

Poll Question

Which statement best captures the primary advantage of using a framework like LangChain for

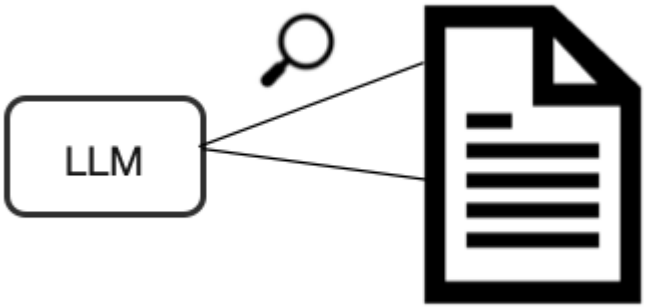
A. It automatically trains your model without any configuration
B. It offers unified abstractions (chains, prompts, parsers) that streamline development
C. It focuses solely on deployment, not on application logic
D. It restricts user inputs to enhance security

Poll Question

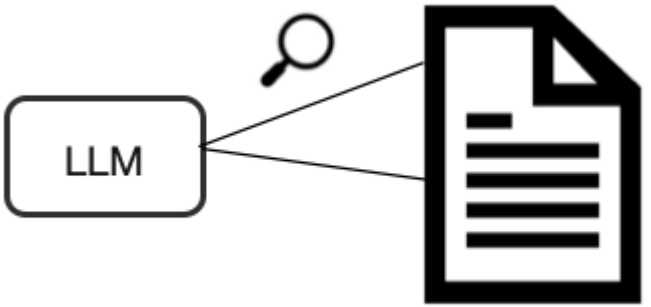
Which best describes the purpose of a "Prompt Template" in LangChain?

A. A fixed script that cannot be modified once generated
B. A dynamic format that reuses prompt structures with variable placeholders
C. An external file that stores long sequences of textual data
D. A built-in mechanism for parsing JSON responses

LangChain for Chat Over Documents

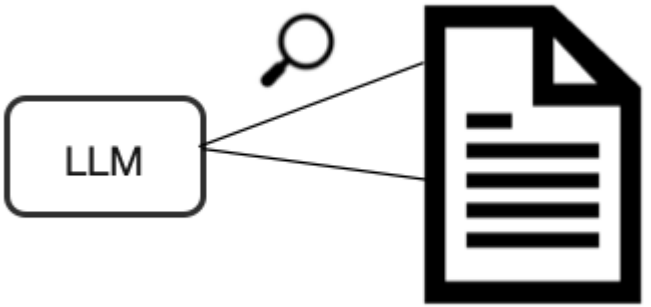


LangChain for Chat Over Documents



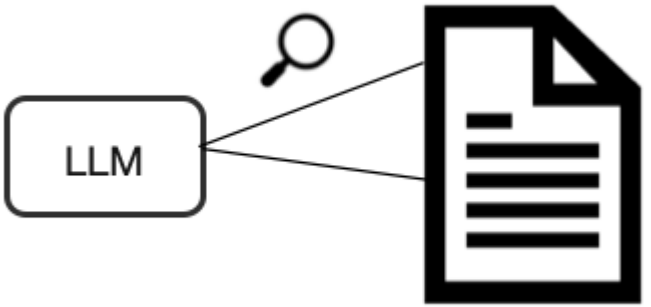
- RAG = **R**etrieval **A**ugmented **G**eneration

LangChain for Chat Over Documents



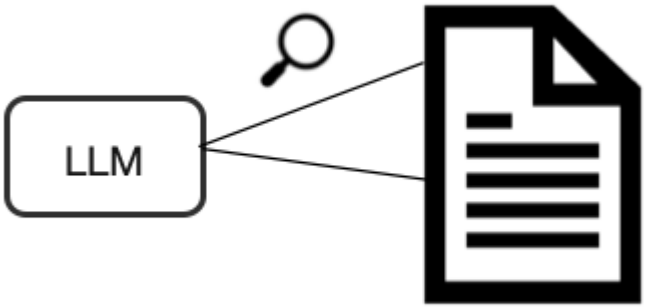
- RAG = **R**etrieval **A**ugmented **G**eneration
- It's about connecting LLMs to documents like PDFs, Text files, HTML, etc.

LangChain for Chat Over Documents



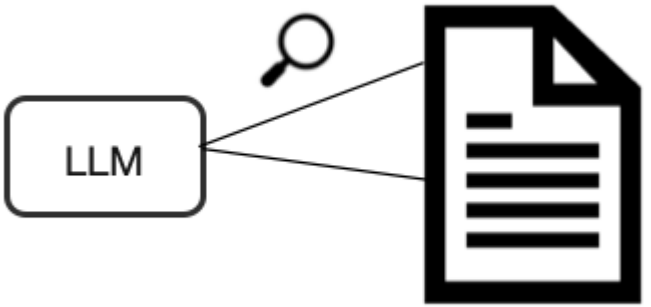
- RAG = **R**etrieval **A**ugmented **G**eneration
- It's about connecting LLMs to documents like PDFs, Text files, HTML, etc.
- How do we get around the context length limitations of LLMs?

LangChain for Chat Over Documents



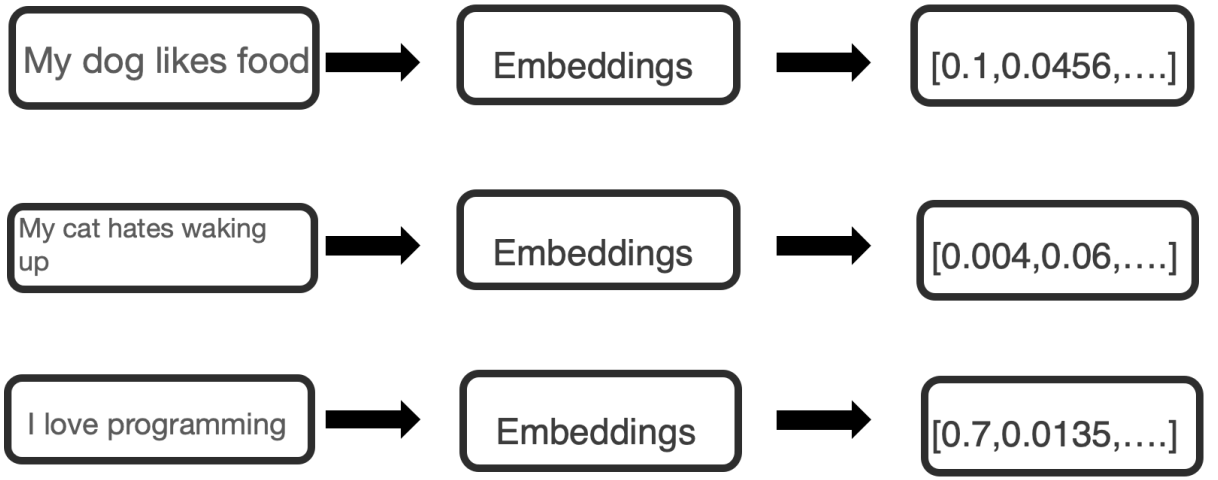
- RAG = **R**etrieval **A**ugmented **G**eneration
- It's about connecting LLMs to documents like PDFs, Text files, HTML, etc.
- How do we get around the context length limitations of LLMs?
- Quick Answer is **Embeddings**!

LangChain for Chat Over Documents

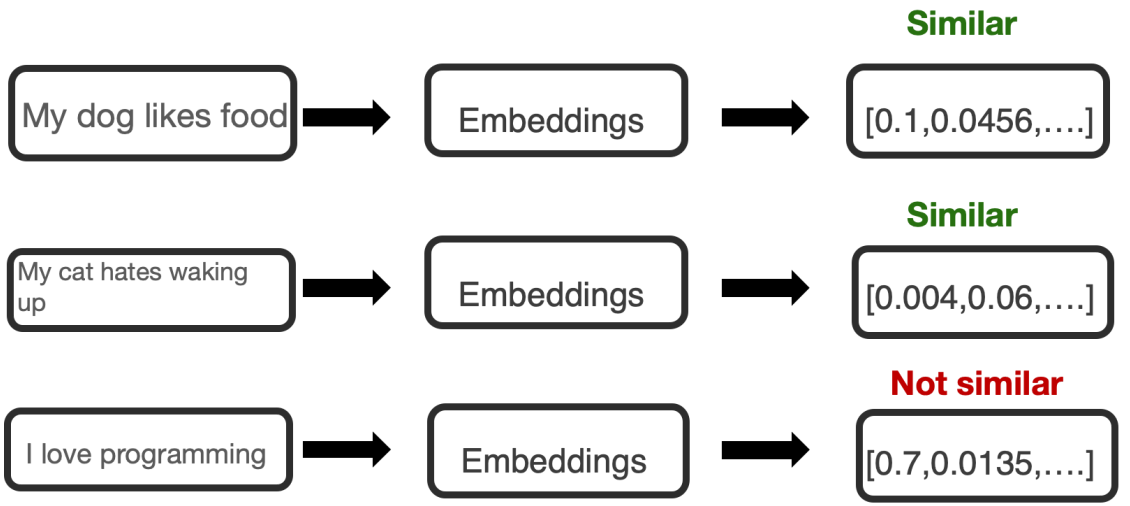


- Embeddings are vectorized representations of text

LangChain for Chat Over Documents

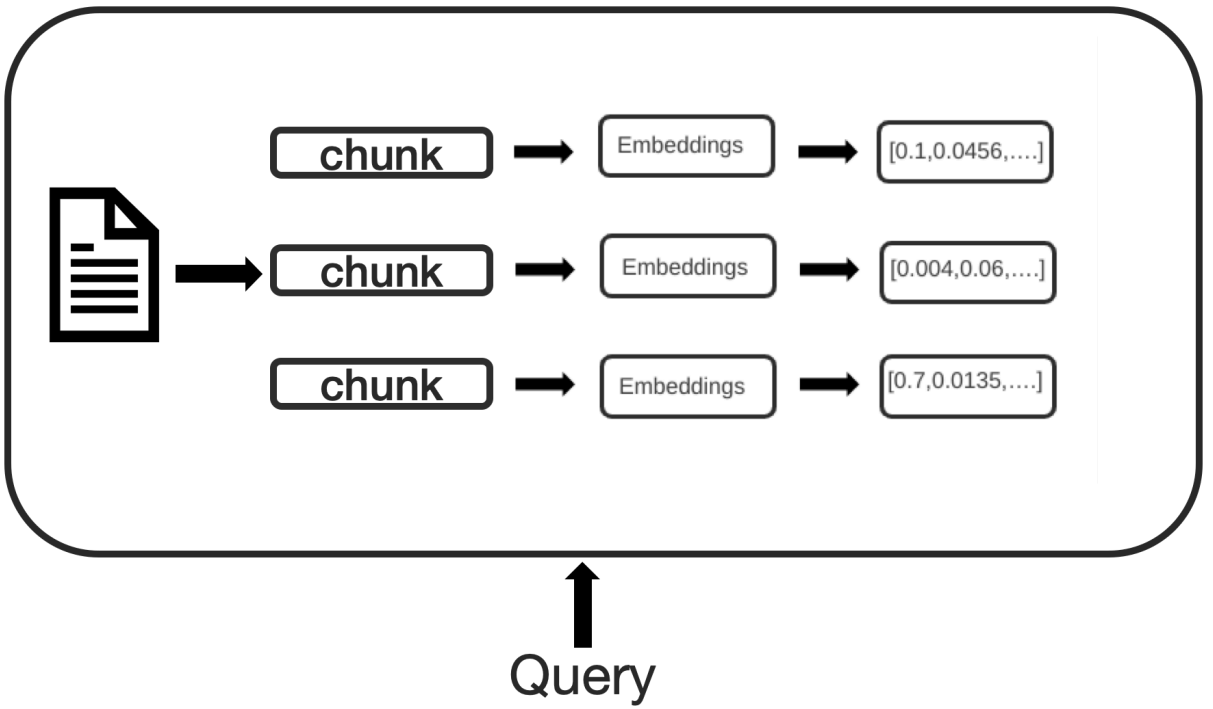


LangChain for Chat Over Documents



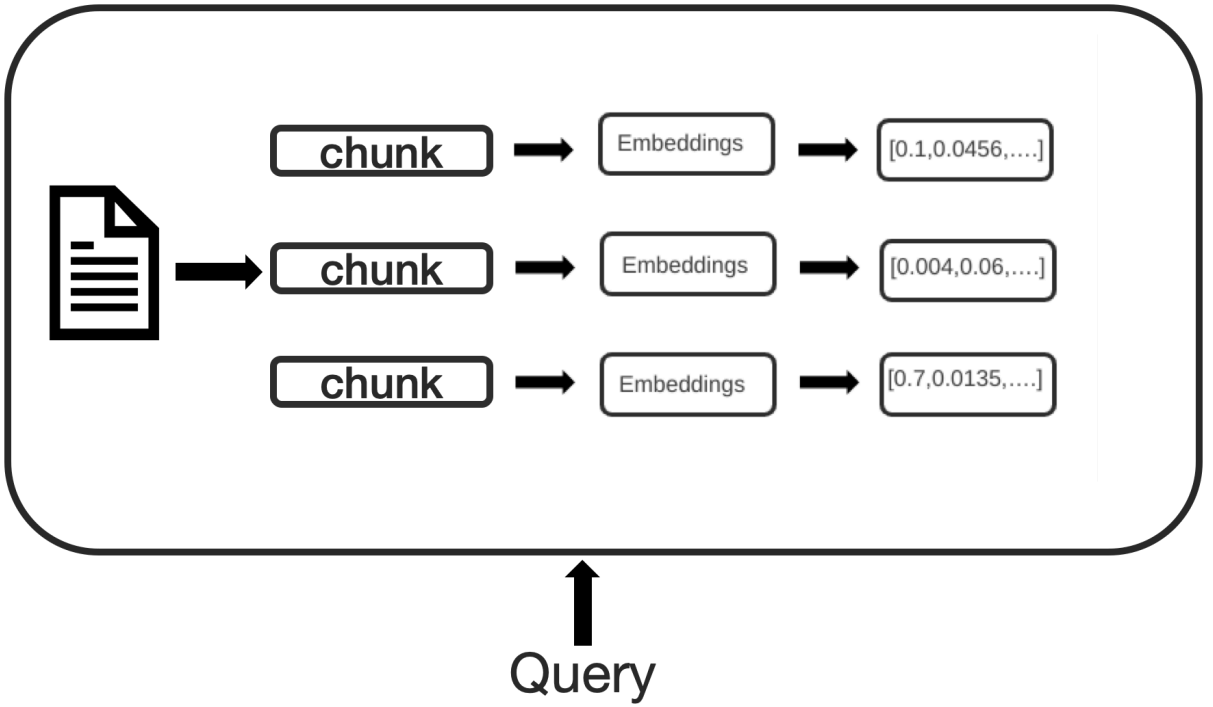
- Embeddings capture content and meaning

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?

Whiteboard Explain - RAG Systems

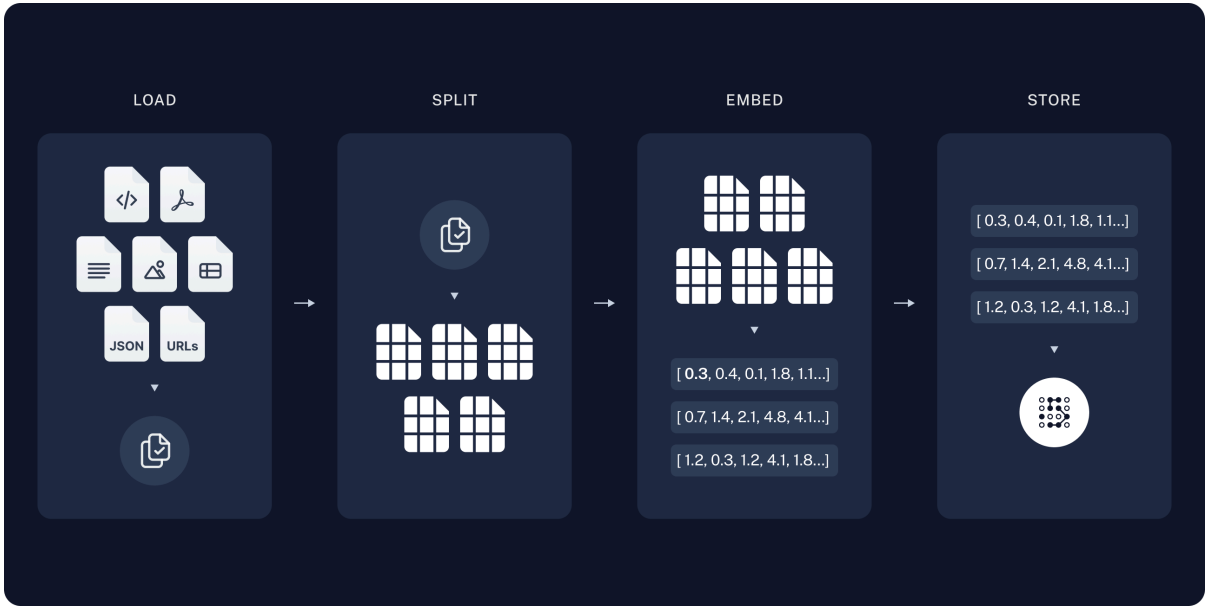
Poll Question

What is the primary purpose of embeddings when building LLM-powered document chat systems?

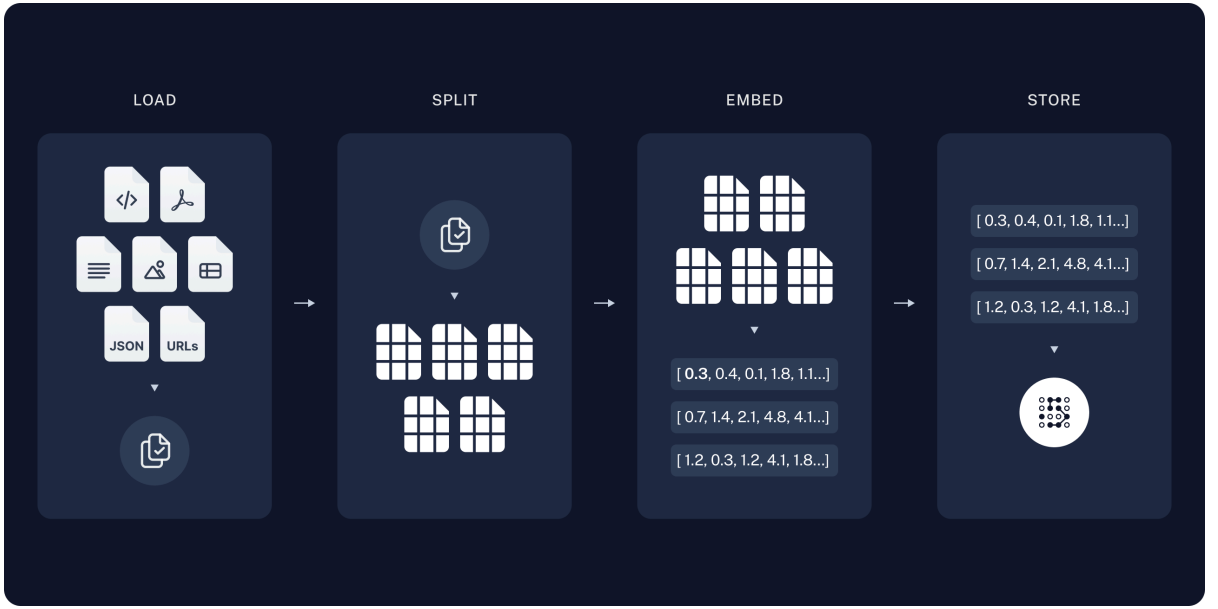
A. Managing API requests and rate limits
B. Turning text into numerical vectors for similarity comparison
C. Automatically generating user interface elements
D. Serving as a secure authentication method

LangChain for Chat Over Documents

LangChain for Chat Over Documents

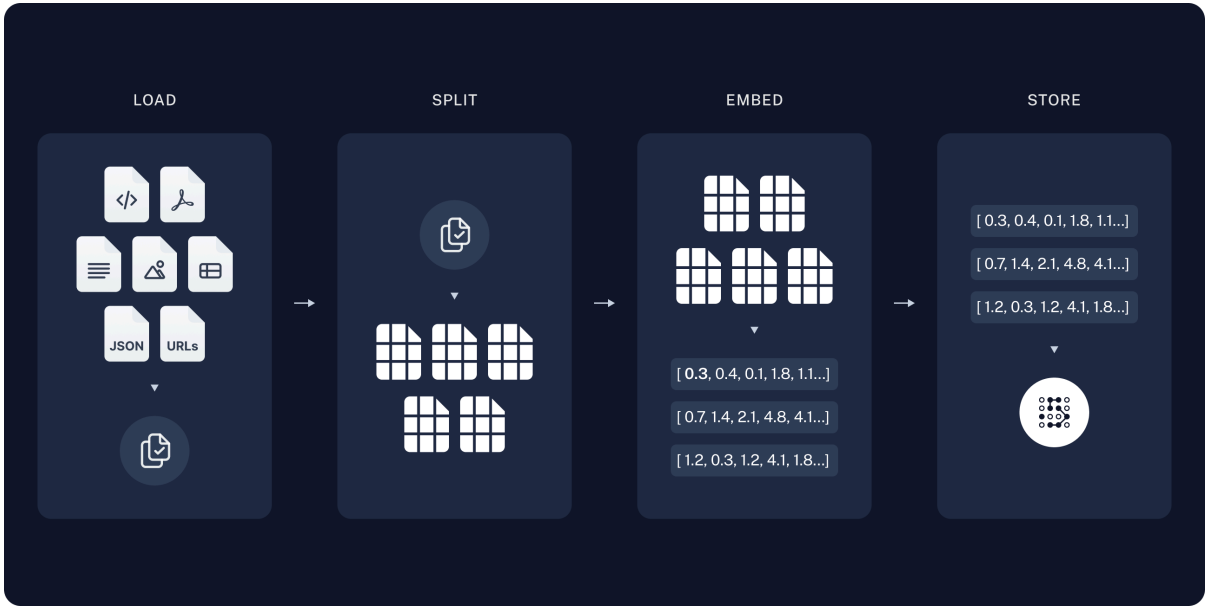


LangChain for Chat Over Documents



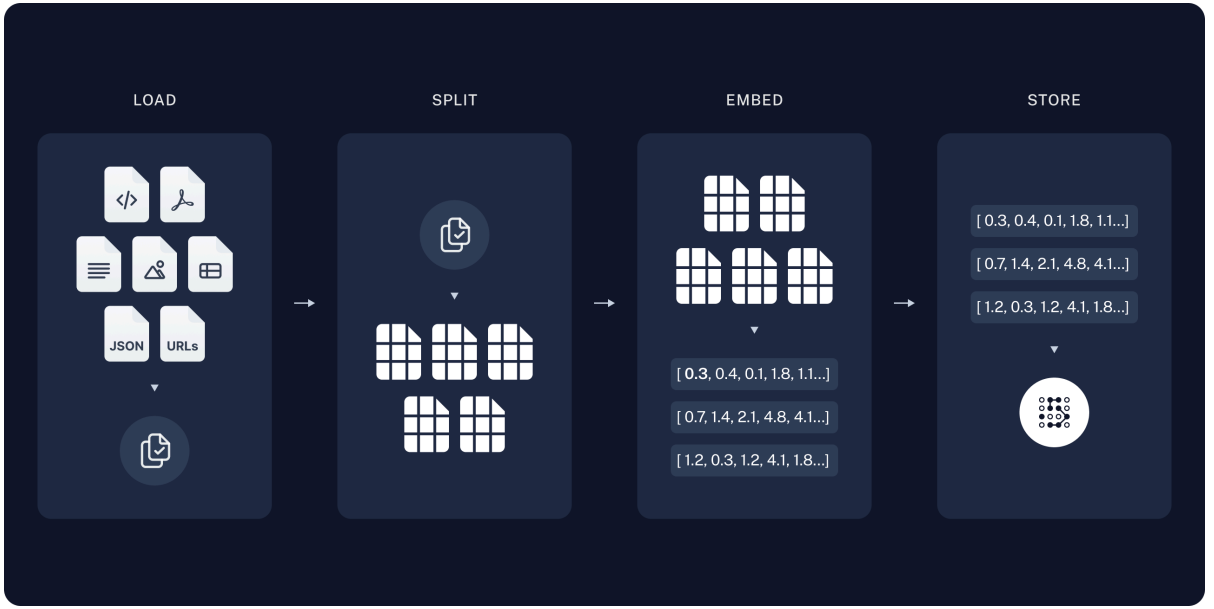
- Load

LangChain for Chat Over Documents



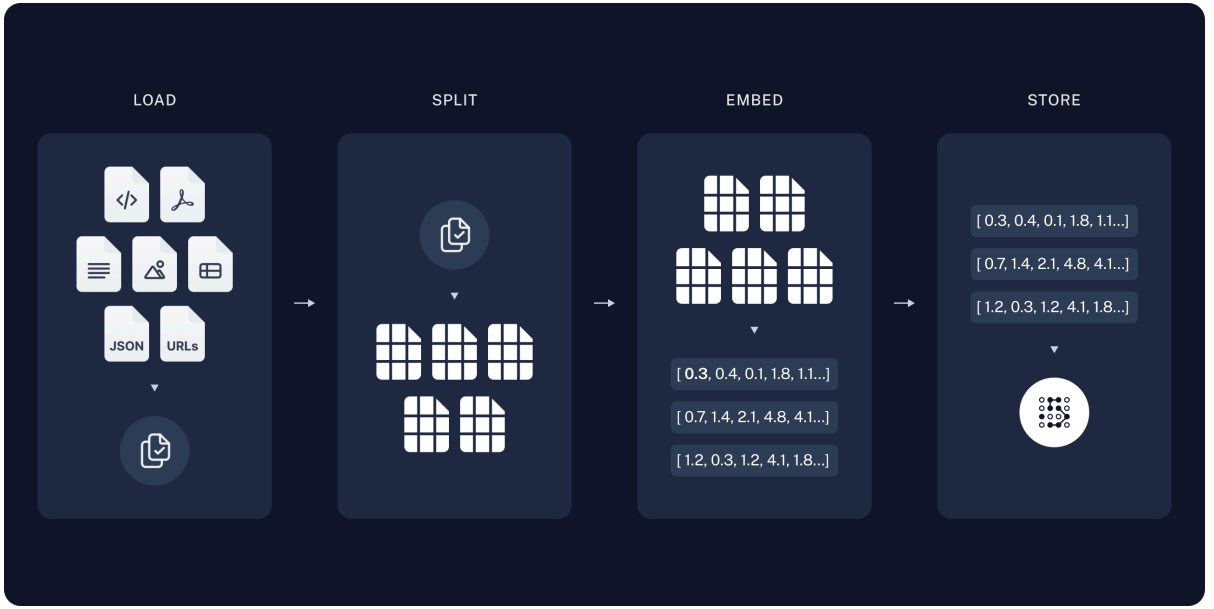
- Load
- Split

LangChain for Chat Over Documents



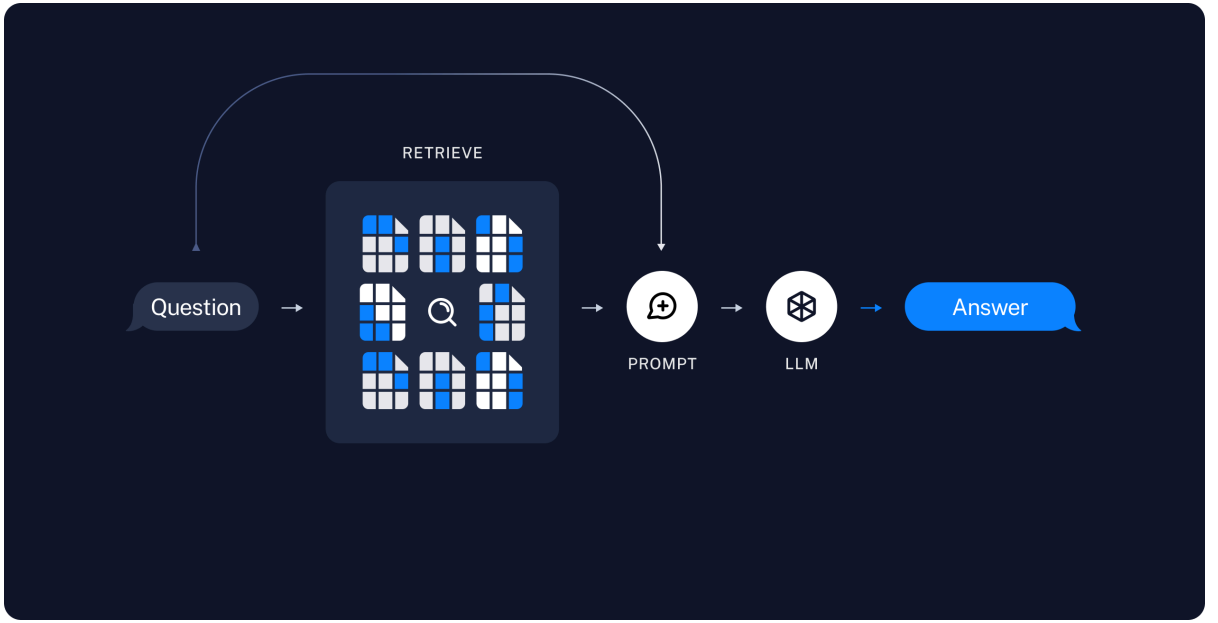
- Load
- Split
- Embed

LangChain for Chat Over Documents

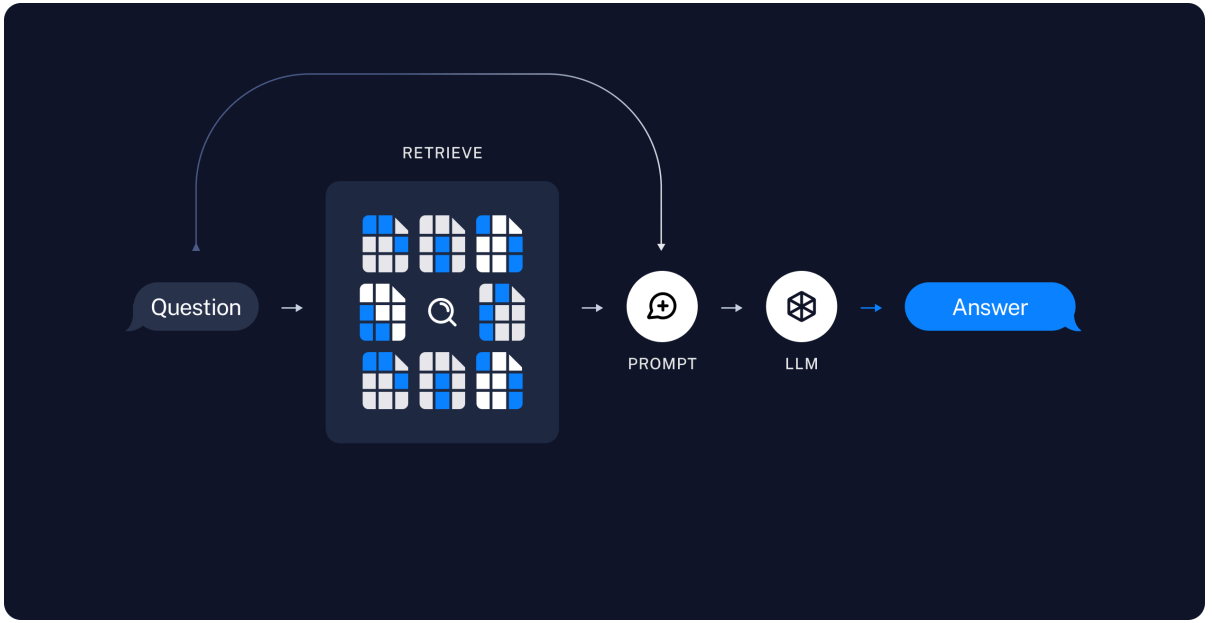


- Load
- Split
- Embed
- Store

LangChain for Chat Over Documents

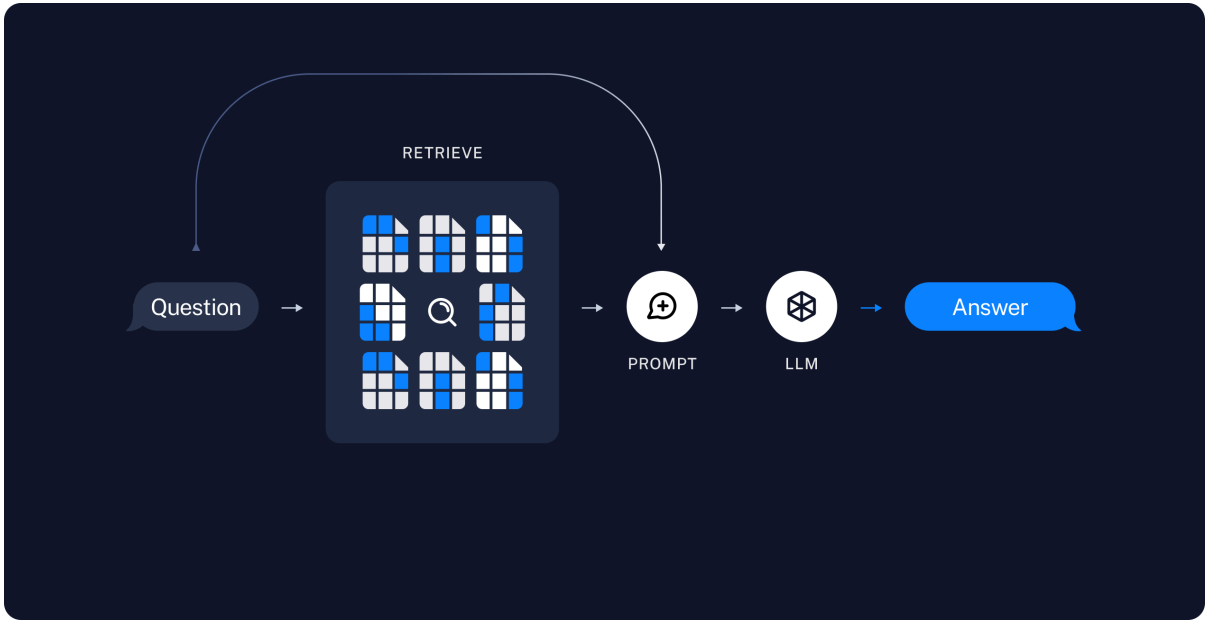


LangChain for Chat Over Documents



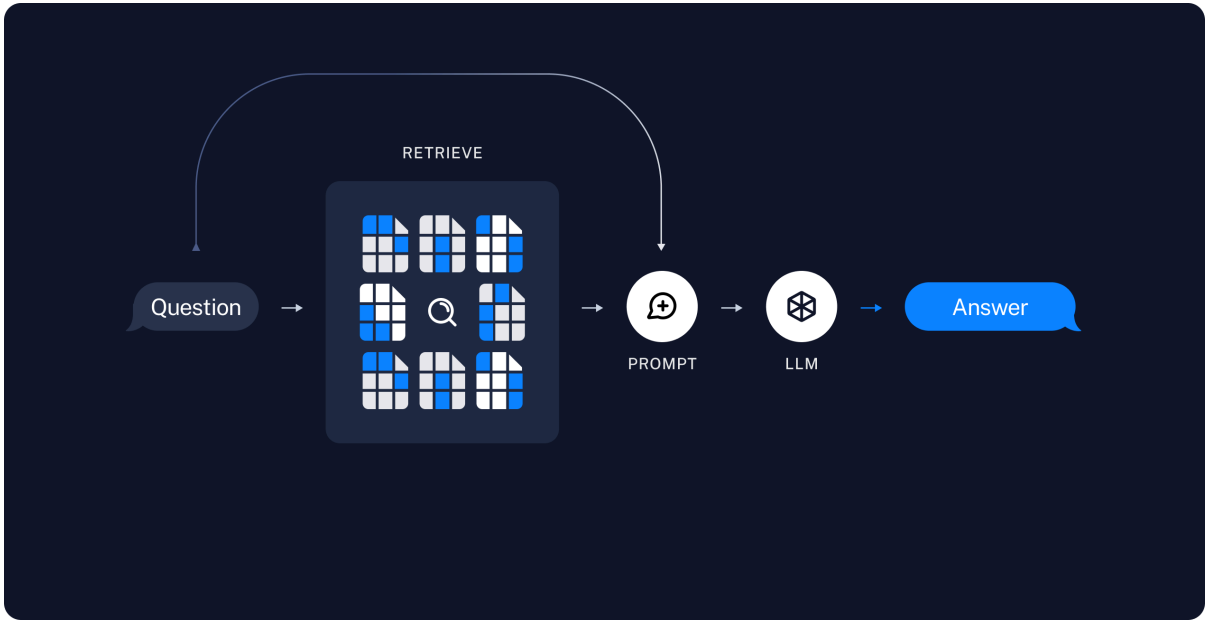
- Retrieval Piepeline

LangChain for Chat Over Documents



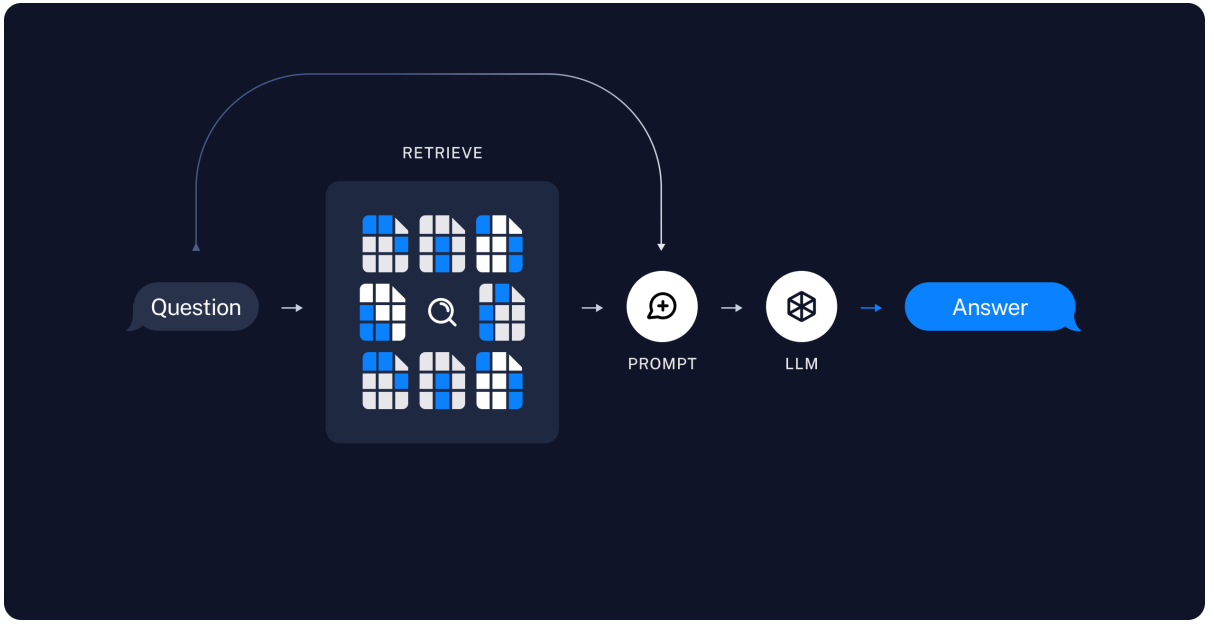
- Retrieval Piepeline
  - Input Question

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

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

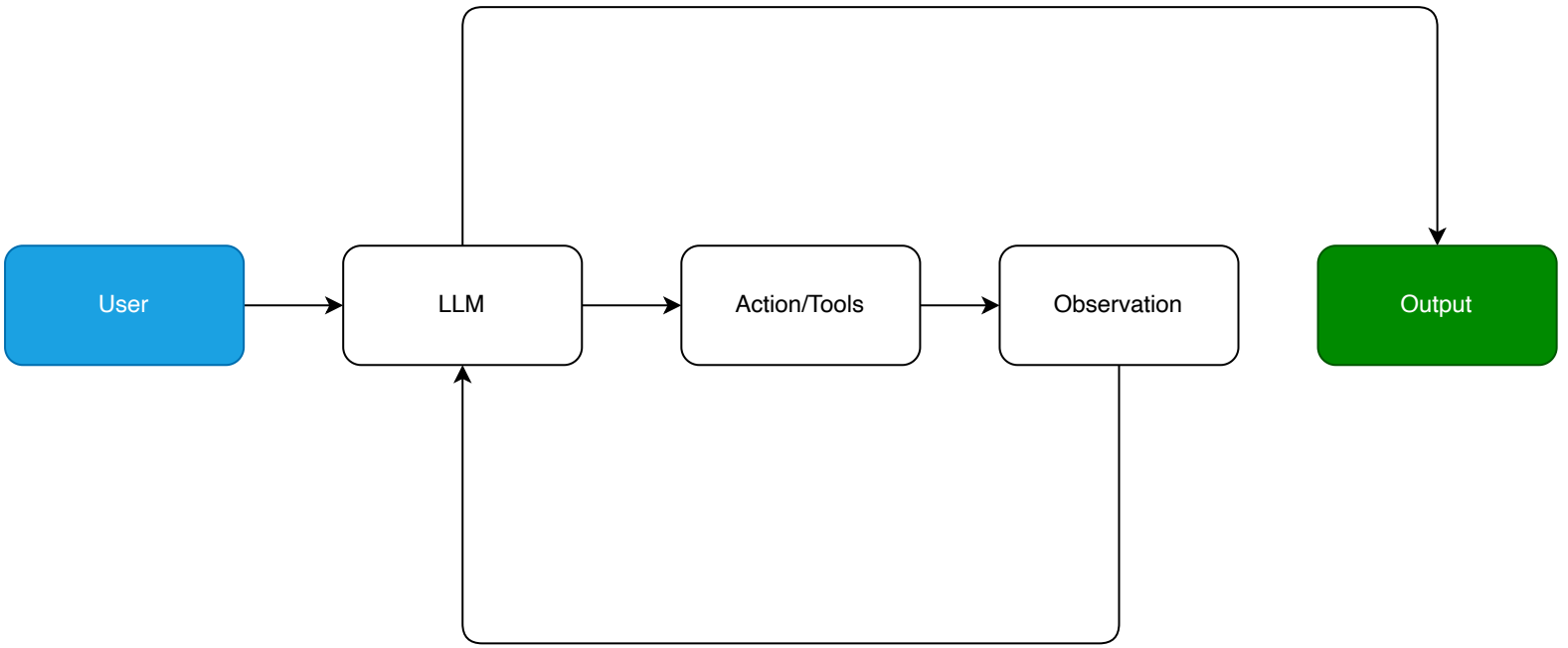
Building Agents with LangChain

Building Agents with LangChain

The Agent Loop

Building Agents with LangChain

The Agent Loop



Components of an Agent

Poll Question

Which best describes the “Agent Loop” illustrated in the presentation?

A. A direct data-loading pipeline for vector databases
B. A continuous decision-making cycle where an LLM decides actions and checks results
C. A script for parallel execution of tasks in a single pass
D. A framework for chunking documents into smaller sections

Notebook Demo - Simple React Agent Walthrough in LangChain

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 deadlines.

```
table format = task | date
```

Understanding Agentic Systems

- **Agentic Systems**: Systems that utilize large language models (LLMs) to manage the control flow of applications.

Understanding Agentic Systems

- **Agentic Systems**: Systems that utilize large language models (LLMs) to manage the control flow of applications.
- **Key Functions**: Routing decisions, tool selection, and evaluating output sufficiency.

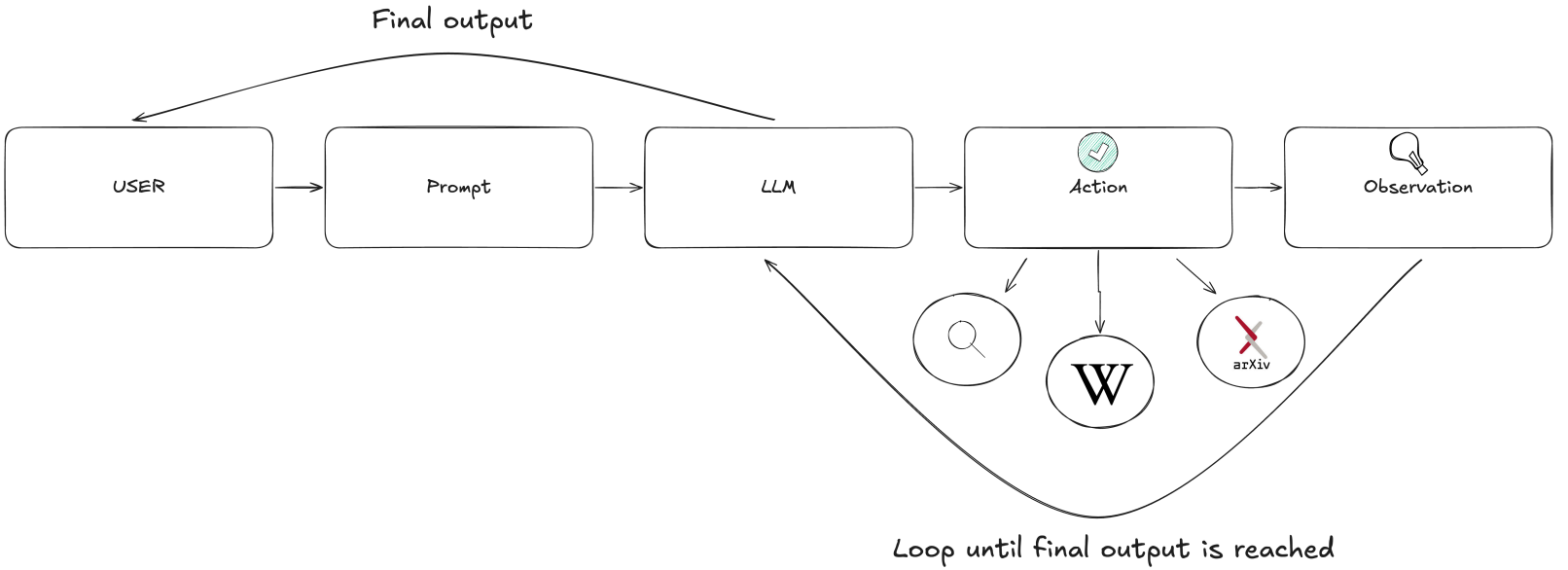
Understanding Agentic Systems

- **Agentic Systems**: Systems that utilize large language models (LLMs) to manage the control flow of applications.
- **Key Functions**: Routing decisions, tool selection, and evaluating output sufficiency.
- **Agent Loop**: Continuous decision-making process that enables agents to solve complex tasks.

Understanding agentic systems in LangGraph.

The Agent Loop

The Agent Loop



Explanation of the agent loop in cognitive architectures.

Practical Use Case: Customer Support Agent

Practical Use Case: Customer Support Agent

- **Scenario**: An LLM-powered customer support agent.

Practical Use Case: Customer Support Agent

- **Scenario**: An LLM-powered customer support agent.
- **User Input**: Customer asks about order status.

Practical Use Case: Customer Support Agent

- **Scenario**: An LLM-powered customer support agent.
- **User Input**: Customer asks about order status.
- **LLM Decision**: Determines if it can provide the status directly or if it needs to fetch data from the database.

Practical Use Case: Customer Support Agent

- **Scenario**: An LLM-powered customer support agent.
- **User Input**: Customer asks about order status.
- **LLM Decision**: Determines if it can provide the status directly or if it needs to fetch data from the database.
- **Action Taken**: If data fetch is needed, the agent queries the database and updates the user with the order status.

Practical use case of LLM agents in customer support.

Advantages of LLM Agents

Advantages of LLM Agents

- **Flexibility**: Agents can adapt to various tasks by determining the best action to take.

Advantages of LLM Agents

- **Flexibility**: Agents can adapt to various tasks by determining the best action to take.
- **Specialization**: LLM agents can be specialized with a set of tools to perform niche tasks.

Advantages of LLM Agents

- **Flexibility**: Agents can adapt to various tasks by determining the best action to take.
- **Specialization**: LLM agents can be specialized with a set of tools to perform niche tasks.
- **Multi-Agent Collaboration**: Specialized LLM agents can collaborate to perform complex tasks.

LangGraph: Multi-Agent Workflows

Key Components of Agentic Systems

Key Components of Agentic Systems

1. **Tool Calling**: Utilizing external tools to perform tasks.

Key Components of Agentic Systems

1. **Tool Calling**: Utilizing external tools to perform tasks.
2. **Action Taking**: Executing actions based on LLM outputs.

Key Components of Agentic Systems

1. **Tool Calling**: Utilizing external tools to perform tasks.
2. **Action Taking**: Executing actions based on LLM outputs.
3. **Memory**: Keeping track of interactions for context-aware responses.

Key Components of Agentic Systems

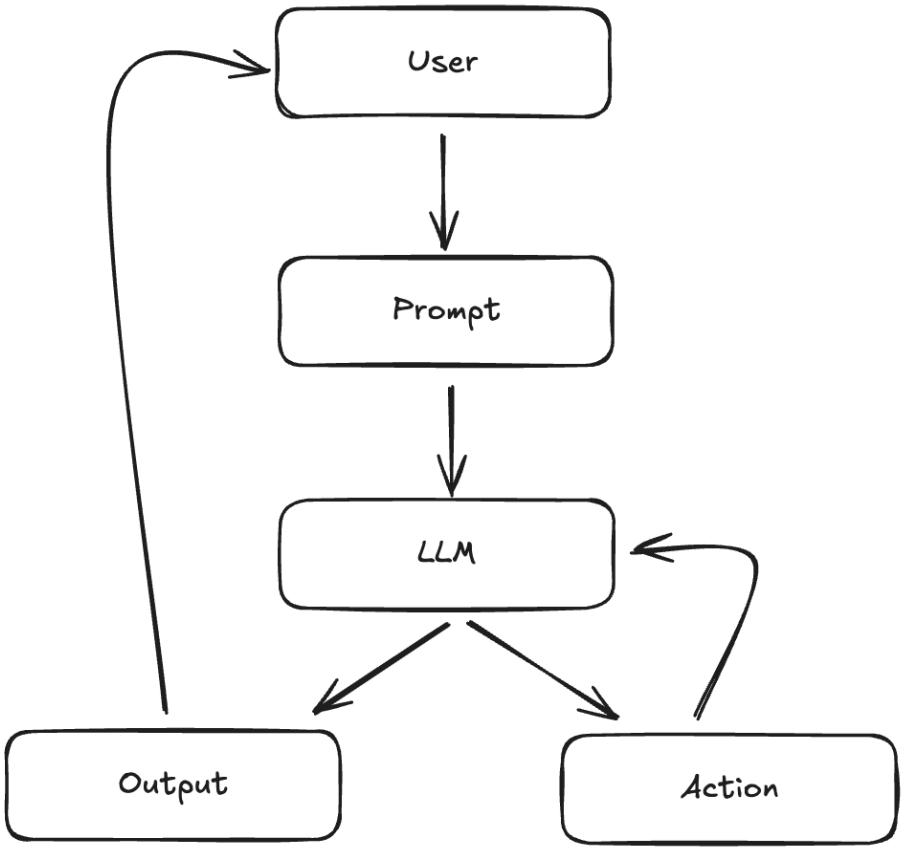
1. **Tool Calling**: Utilizing external tools to perform tasks.
2. **Action Taking**: Executing actions based on LLM outputs.
3. **Memory**: Keeping track of interactions for context-aware responses.
4. **Planning**: Structuring steps to ensure optimal decision-making.

Key components of agentic systems.

Agents as Graphs

Agents as Graphs

- Workflows built with agents are usually structured as graphs!!



Why LangGraph?

LangGraph is designed for building agentic applications with some core principles:

Why LangGraph?

LangGraph is designed for building agentic applications with some core principles:

- **Controllability**: Offers low-level control which increases reliability in agentic systems.

Why LangGraph?

LangGraph is designed for building agentic applications with some core principles:

- **Controllability**: Offers low-level control which increases reliability in agentic systems.
- **Human-in-the-Loop**: Built-in persistence layer enhances human-agent interaction patterns.

Why LangGraph?

LangGraph is designed for building agentic applications with some core principles:

- **Controllability**: Offers low-level control which increases reliability in agentic systems.
- **Human-in-the-Loop**: Built-in persistence layer enhances human-agent interaction patterns.
- **Streaming First**: Supports streaming of events and tokens, providing real-time feedback to users.

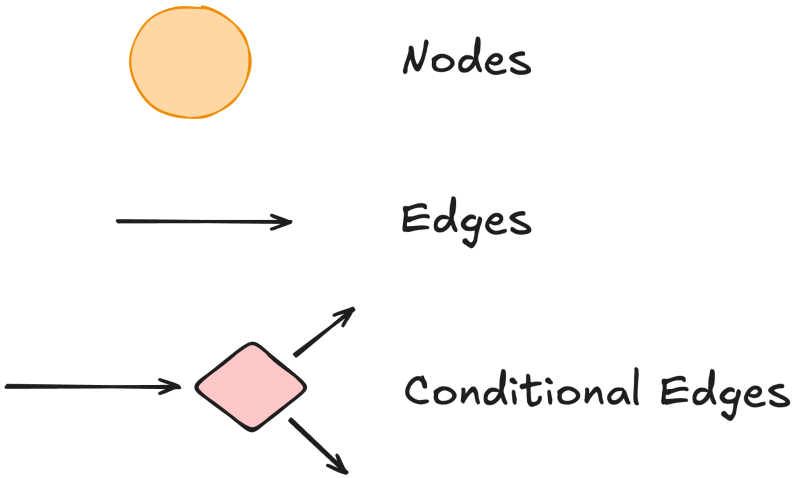
Overview of LangGraph's purpose and principles.

The Basic Components of LangGraph

LangGraph models agent workflows as graphs:

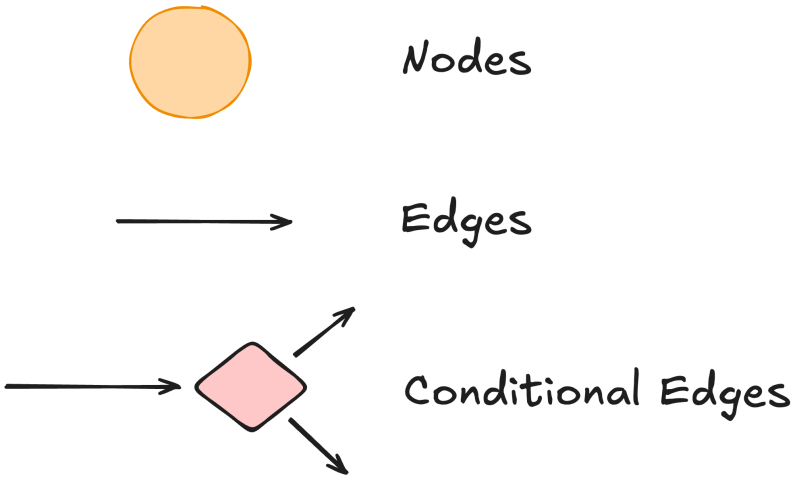
The Basic Components of LangGraph

LangGraph models agent workflows as graphs:



The Basic Components of LangGraph

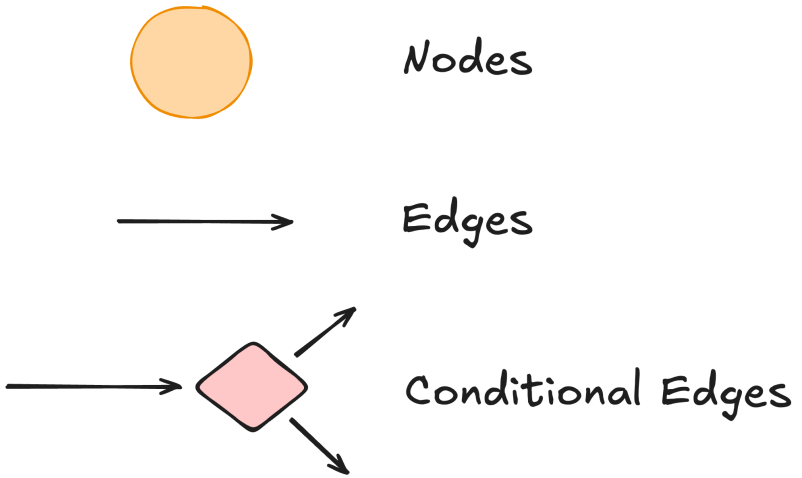
LangGraph models agent workflows as graphs:



- **Nodes**: Python functions that implement the logic of agents, taking the current State as input and returning an updated State.

The Basic Components of LangGraph

LangGraph models agent workflows as graphs:



- **Nodes**: Python functions that implement the logic of agents, taking the current State as input and returning an updated State.
- **Edges/Conditional Edges**: Functions that implement fixed/conditional transitions to determine which Node to execute next based on the current State.

Explanation of LangGraph's components.

States in LangGraph

These graphs in LangGraph are driven by:

States in LangGraph

These graphs in LangGraph are driven by:

- **States**: Shared data structures that evolve over time as Nodes execute and pass messages along Edges.

States in LangGraph

These graphs in LangGraph are driven by:

- **States**: Shared data structures that evolve over time as Nodes execute and pass messages along Edges.
- **Message Passing**: Nodes send messages to activate other Nodes, facilitating the execution of workflows in discrete iterations or "super-steps".

Overview of how graphs and states interact in LangGraph.

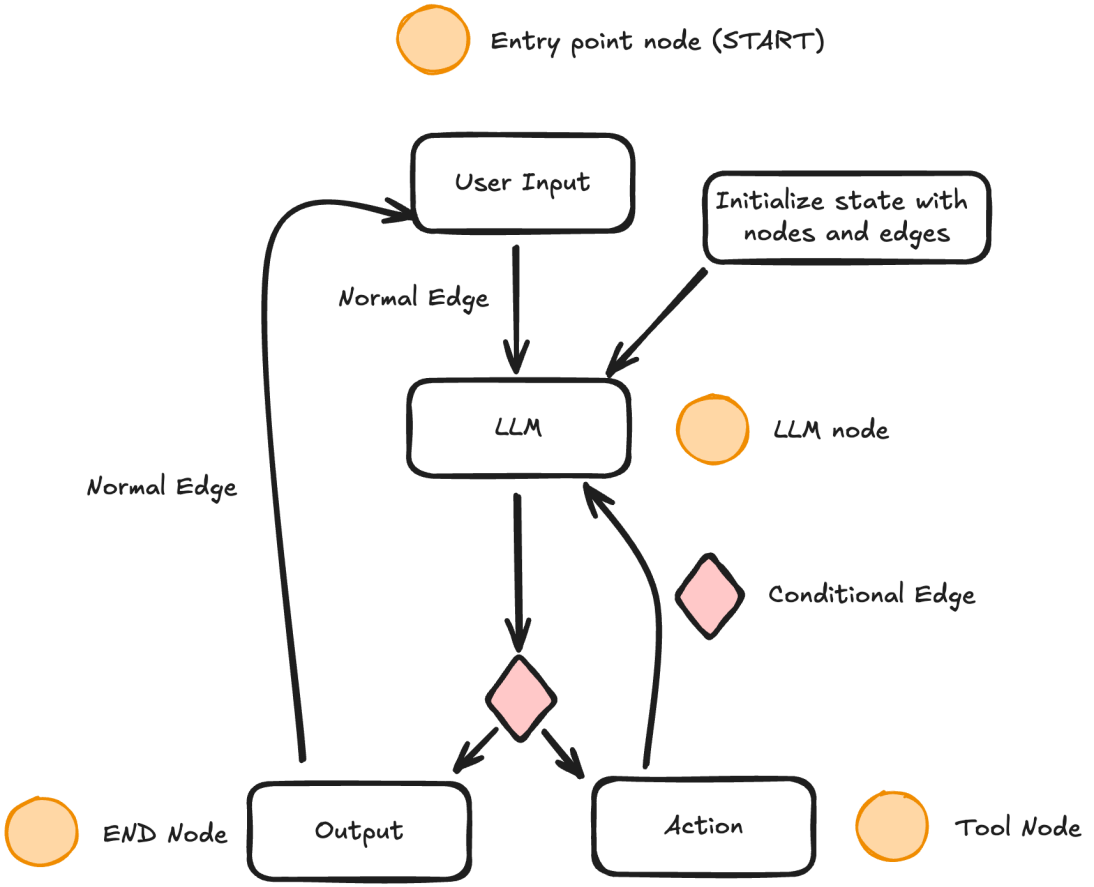
Notebook Demo: Introduction to LangGraph

Agent Loop in LangGraph

- Outline of a basic agent loop in langgraph:

Agent Loop in LangGraph

- Outline of a basic agent loop in langgraph:



Notebook Demo: A Basic Research Agent in LangGraph

Q&A & Summary

- **LangGraph** is a framework for building agentic applications.
- **Nodes** are the core functional units in LangGraph.
- **Edges** define the routing logic in LangGraph.
- **States** are shared data structures that evolve over time as Nodes execute and pass messages along Edges.
- **Agent Loop** is the continuous decision-making process that enables agents to solve complex tasks.

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)

Notebook Demo - Deployment with LangServe

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