

Integrating document loaders

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

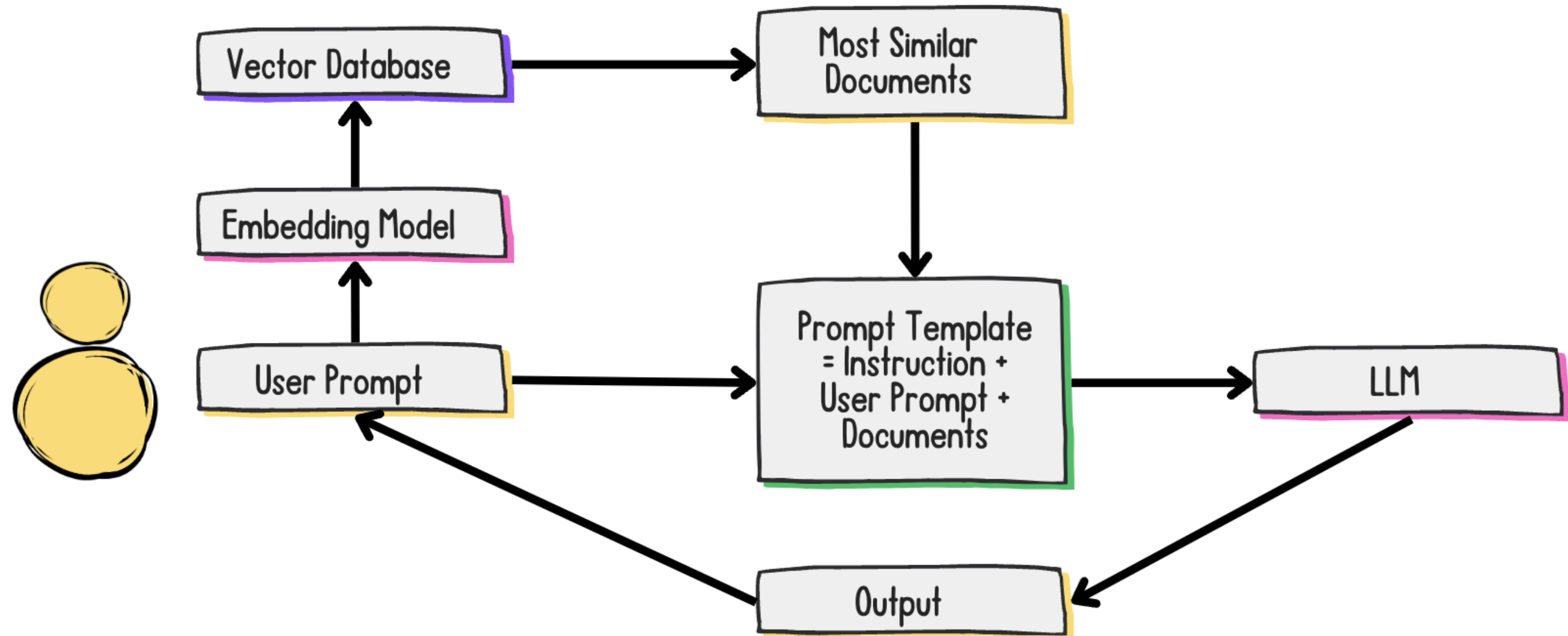


Jonathan Bennion

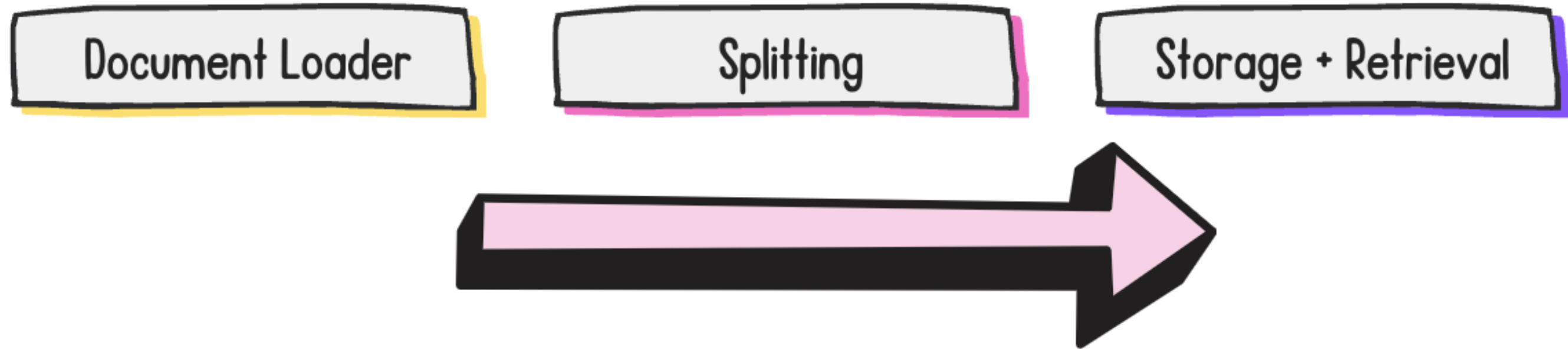
AI Engineer & LangChain Contributor

Retrieval Augmented Generation (RAG)

- Use embeddings to *retrieve* relevant information to integrate into the *prompt*



RAG development steps



LangChain document loaders

- Classes designed to *load* and *configure* documents for system integration
- Document loaders for common file types:
`.pdf` , `.csv`
- 3rd party loaders: S3, `.ipynb` , `.wav`



¹ https://python.langchain.com/docs/integrations/document_loaders

PDF document loader

- Requires installation of the `pypdf` package: `pip install pypdf`

```
from langchain_community.document_loaders import PyPDFLoader
loader = PyPDFLoader("path/to/file/attention_is_all_you_need.pdf")

data = loader.load()
print(data[0])
```

```
Document(page_content='Provided proper attribution is provided, Google hereby grants  
permission to\nreproduce the tables and figures in this paper solely for use in [...]
```

CSV document loader

```
from langchain_community.document_loaders.csv_loader import CSVLoader

loader = CSVLoader('fifa_countries_audience.csv')

data = loader.load()
print(data[0])
```

```
Document(page_content='country: United States\nconfederation: CONCACAF\npopulation_share: [...]
```

HTML document loader

```
from langchain_community.document_loaders import UnstructuredHTMLLoader

loader = UnstructuredHTMLLoader("white_house_executive_order_nov_2023.html")
data = loader.load()

print(data[0])
print(data[0].metadata)
```

```
page_content="To search this site, enter a search term\n\nSearch\n\nExecutive Order on the Safe, Secure,  
and Trustworthy Development and Use of Artificial Intelligence\n\nHome\n\nBriefing Room\n\nPresidential  
Actions\n\nBy the authority vested in me as President by the Constitution and the laws of the United  
States of America, it is hereby ordered as follows: ..."
```

```
{'source': 'white_house_executive_order_nov_2023.html'}
```

Let's practice!

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Splitting external data for retrieval

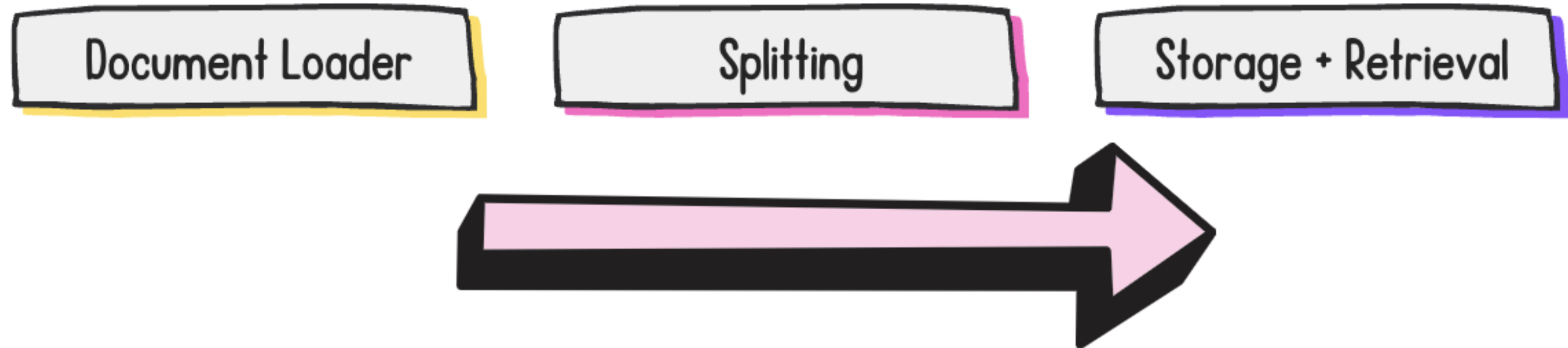
DEVELOPING LLM APPLICATIONS WITH LANGCHAIN



Jonathan Bennion

AI Engineer & LangChain Contributor

RAG development steps



- **Document splitting:** split document into *chunks*
- Break documents up to fit within an LLM's *context window*

Thinking about splitting...

1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [35, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [38, 24, 15].

Line 1:

```
Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks
```

Line 2:

```
in particular, have been firmly established as state of the art approaches in sequence modeling and
```

¹ <https://arxiv.org/abs/1706.03762>

Chunk overlap

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [35, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [38, 24, 15].

What is the best document splitting strategy?



1. `CharacterTextSplitter`
2. `RecursiveCharacterTextSplitter`
3. Many others

¹ Wikipedia Commons

```
quote = '''One machine can do the work of fifty ordinary humans.\nNo machine can do  
the work of one extraordinary human.'''
```

```
len(quote)
```

```
103
```

```
chunk_size = 24  
chunk_overlap = 3
```

¹ Elbert Hubbard

```
from langchain.text_splitter import CharacterTextSplitter

ct_splitter = CharacterTextSplitter(
    separator='.',
    chunk_size=chunk_size,
    chunk_overlap=chunk_overlap)

docs = ct_splitter.split_text(quote)
print(docs)
print([len(doc) for doc in docs])
```

```
['One machine can do the work of fifty ordinary humans',
 'No machine can do the work of one extraordinary human']
[52, 53]
```

- Split on separator so < `chunk_size` , but may not always succeed!

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

rc_splitter = RecursiveCharacterTextSplitter(
    separators=["\n\n", "\n", " ", ""]
    chunk_size=chunk_size,
    chunk_overlap=chunk_overlap)

docs = rc_splitter.split_text(quote)
print(docs)
```


RecursiveCharacterTextSplitter

- `separators=["\n\n", "\n", " ", ""]`

```
['One machine can do the',  
 'work of fifty ordinary',  
 'humans.',  
 'No machine can do the',  
 'work of one',  
 'extraordinary human.']
```

1. Try splitting by paragraph: `"\n\n"`
2. Try splitting by sentence: `"\n"`
3. Try splitting by words: `" "`

RecursiveCharacterTextSplitter with HTML

```
from langchain_community.document_loaders import UnstructuredHTMLLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

loader = UnstructuredHTMLLoader("white_house_executive_order_nov_2023.html")
data = loader.load()

rc_splitter = RecursiveCharacterTextSplitter(
    chunk_size=chunk_size,
    chunk_overlap=chunk_overlap,
    separators=['.'])

docs = rc_splitter.split_documents(data)
print(docs[0])
```

```
Document(page_content="To search this site, enter a search term [...]
```

Let's practice!

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

RAG storage and retrieval using vector databases

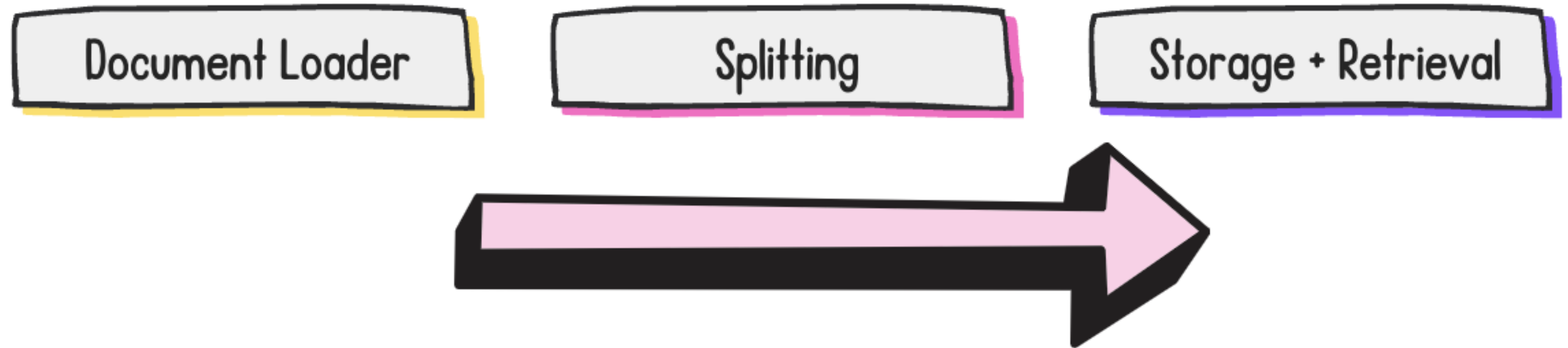
DEVELOPING LLM APPLICATIONS WITH LANGCHAIN



Jonathan Bennion

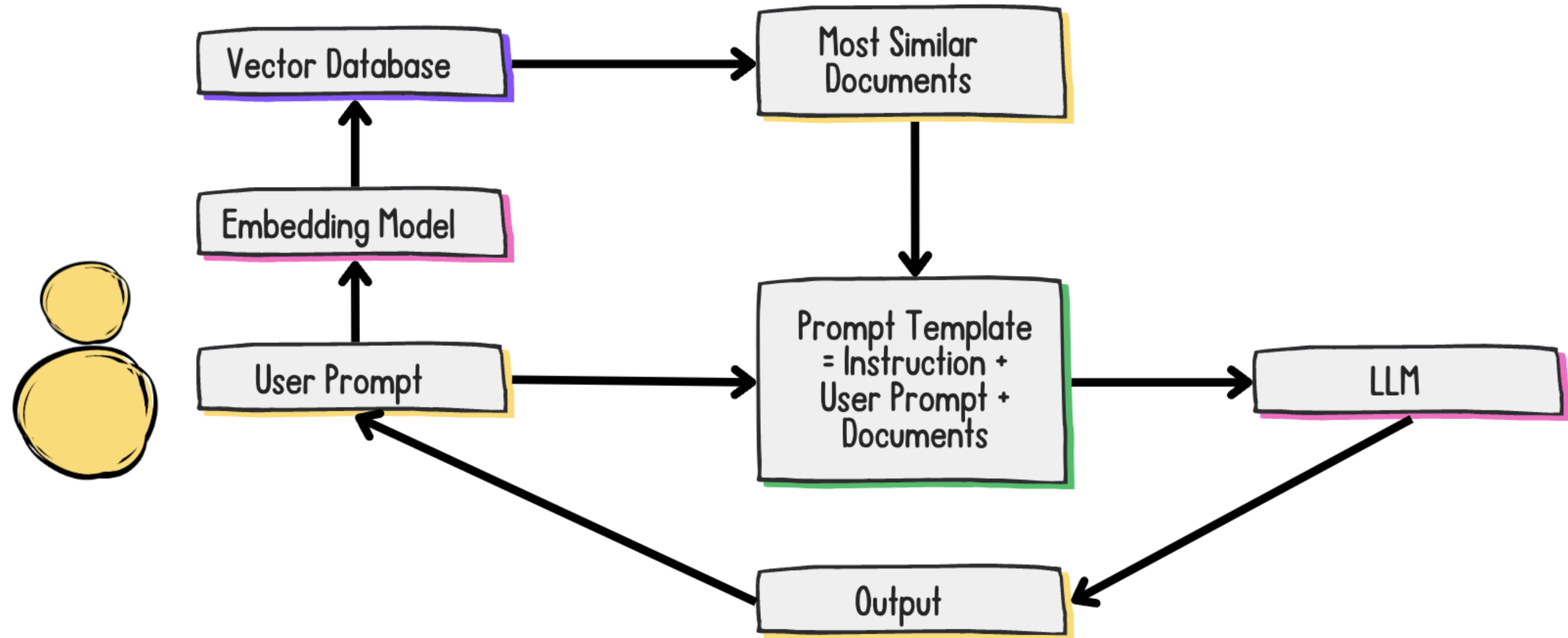
AI Engineer & LangChain Contributor

RAG development steps

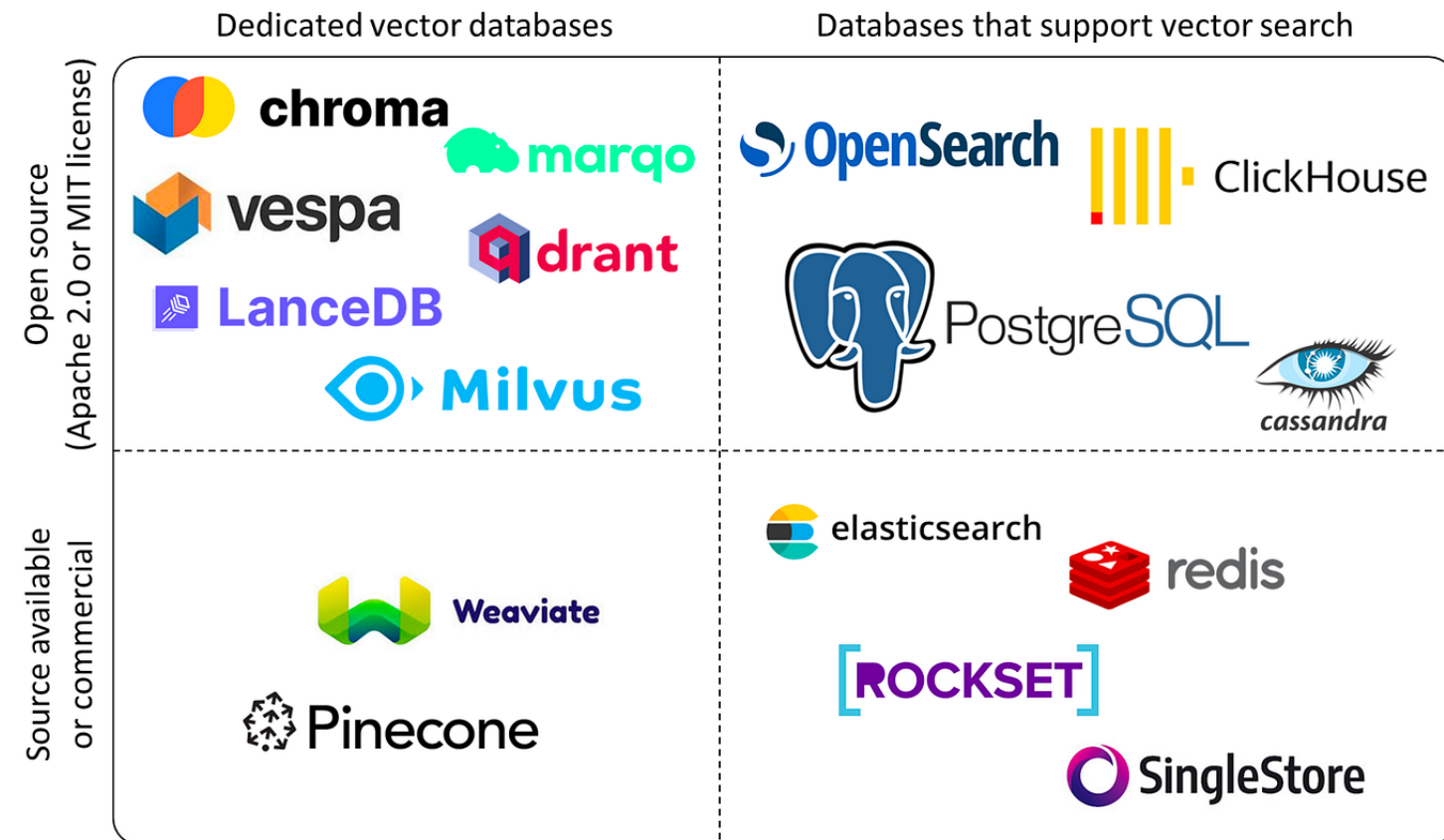


- Focus of this video: *storage and retrieval*

What is a vector database and why do I need it?



Which vector database should I use?



Need to consider:

- Open source vs. closed source (license)
- Cloud vs. on-premises
- Lightweight vs. powerful

¹ Image Credit: Yingjun Wu

Meet the documents...

docs

```
[
  Document(
    page_content="In all marketing copy, TechStack should always be written with the T and S capitalized. Incorrect: techstack, Techstack, etc.",
    metadata={"guideline": "brand-capitalization"}
  ),
  Document(
    page_content="Our users should be referred to as techies in both internal and external communications.",
    metadata={"guideline": "referring-to-users"}
  )
]
```


Setting up a Chroma vector database

```
from langchain_openai import OpenAIEmbeddings
from langchain_chroma import Chroma

embedding_function = OpenAIEmbeddings(openai_api_key=openai_api_key)

vectorstore = Chroma.from_documents(
    docs,
    embedding=embedding_function,
    persist_directory="path/to/directory"
)

retriever = vectorstore.as_retriever(
    search_type="similarity",
    search_kwargs={"k": 2}
)
```

Building a prompt template

```
message = """
Review and fix the following TechStack marketing copy with the following guidelines in consideration:

Guidelines:
{guidelines}

Copy:
{copy}

Fixed Copy:
"""

prompt_template = ChatPromptTemplate.from_messages([("human", message)])
```

Chaining it all together!

```
from langchain_core.runnables import RunnablePassthrough

rag_chain = ({ "guidelines": retriever, "copy": RunnablePassthrough() }
             | prompt_template
             | llm)

response = rag_chain.invoke("Here at techstack, our users are the best in the world!")
print(response.content)
```

```
Here at TechStack, our techies are the best in the world!
```

Let's practice!

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Wrap-up!

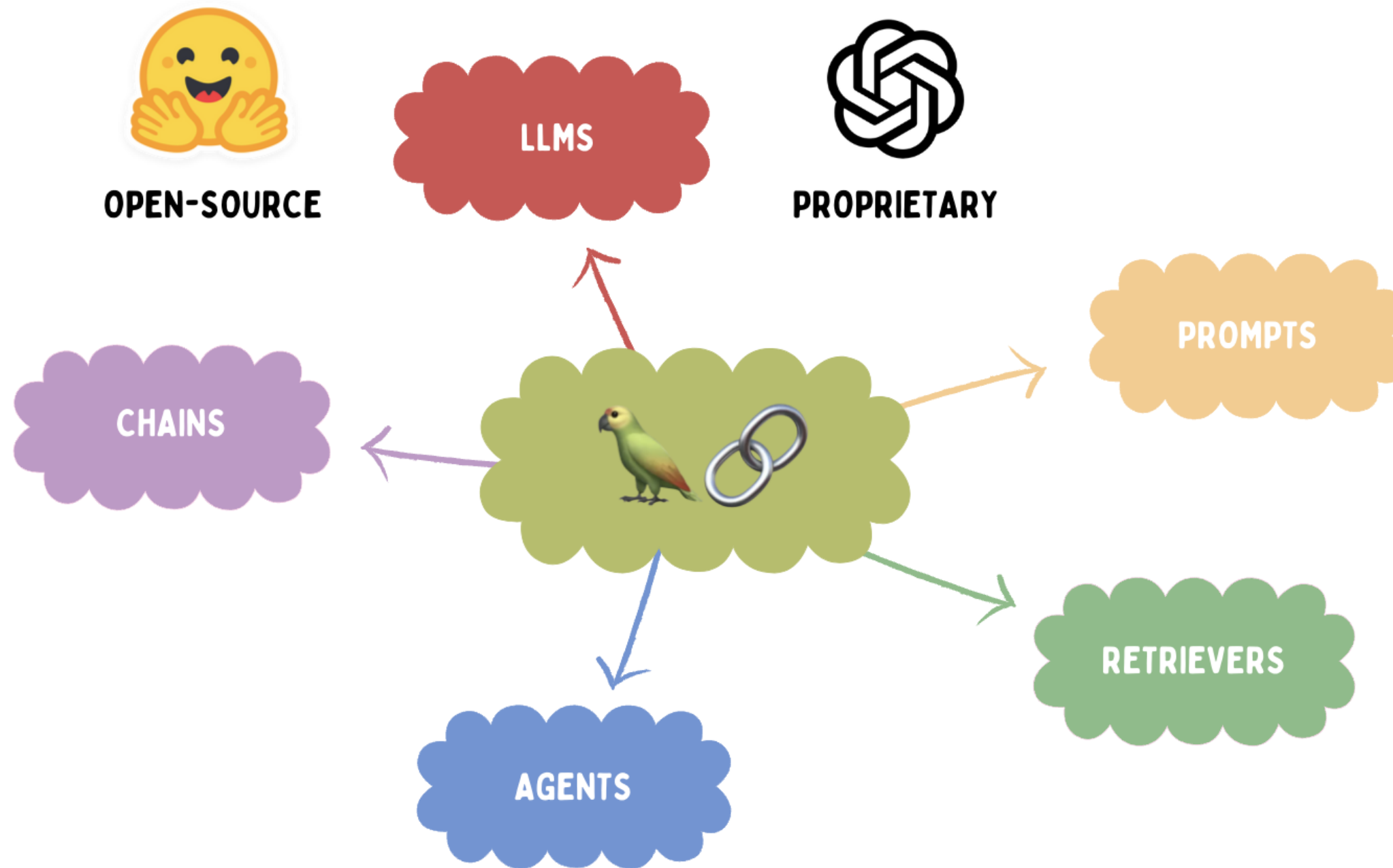
DEVELOPING LLM APPLICATIONS WITH LANGCHAIN



Jonathan Bennion

AI Engineer & LangChain Contributor

LangChain's core components



Chains and agents

User Input: Why isn't my code working? Here it is...

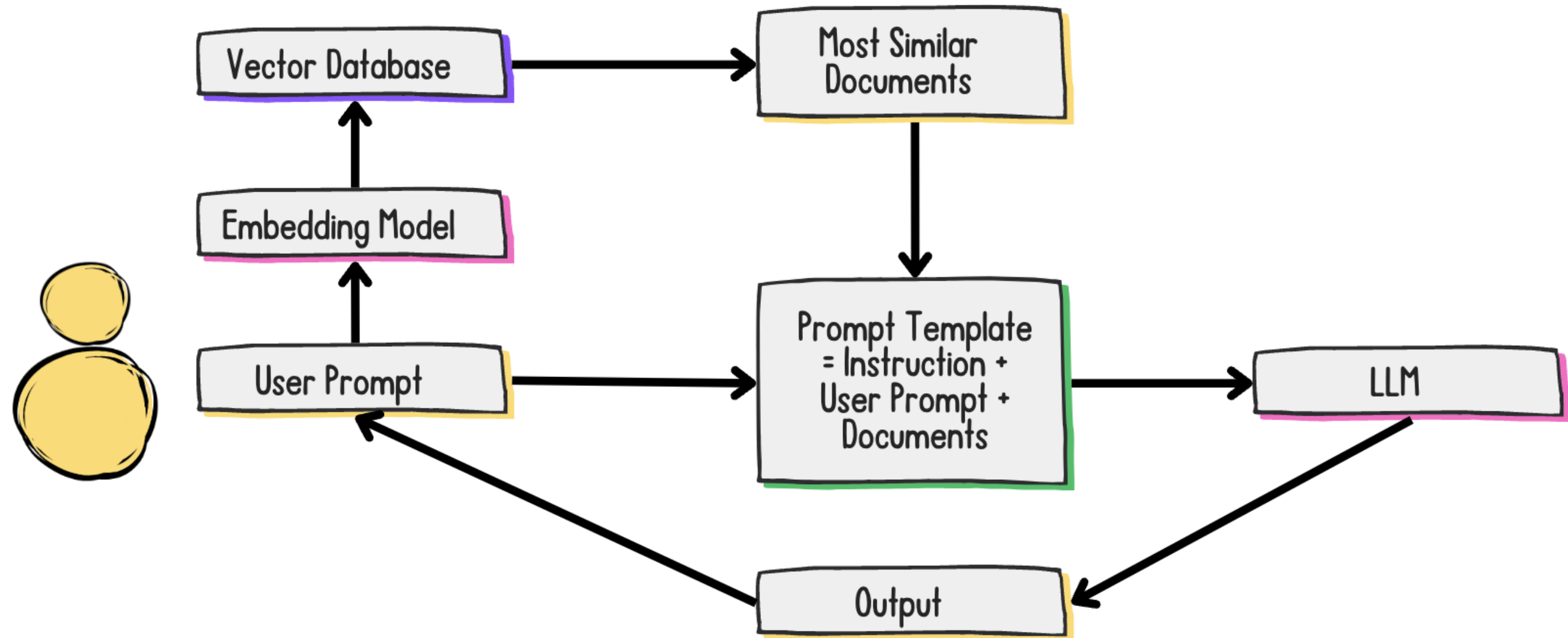


Agent



Tools

Retrieval Augmented Generation (RAG)



LangChain Hub

The screenshot displays the LangChain Hub interface. On the left is a sidebar with filters for Use Cases, Type, and Language. The main content area features a search bar, sorting options (Top Favorited, Top Viewed, Top Downloaded, Recently Updated), and a list of prompts with details like author, description, and statistics.

Use Cases

- ☐ Agent simulations 4
- ☐ Agents 53
- ☐ Autonomous agents 11
- ☐ Chatbots 73
- ☐ Classification 5
- ☐ Code understanding 17
- ☐ Code writing 19
- ☐ Evaluation 21
- ☐ Extraction 38
- ☐ Interacting with APIs 17
- ☐ Multi-modal 3
- ☐ QA over documents 59
- ☐ Self-checking 8
- ☐ SQL 5
- ☐ Summarization 59
- ☐ Tagging 9

Type

- ☐ ChatPromptTemp... 240
- ☐ StringPromptTem... 183

Language

Search

Search for prompts, use cases, models...

Sorting

- Top Favorited
- Top Viewed
- Top Downloaded
- Recently Updated

Prompts

Agents **Interacting with APIs** **ChatPromptTemplate** **meta:llama-2-70b-chat** **Try it**

homanp/superagent

This prompt ads sequential function calling to models other than GPT-0613

{x} Prompt • Updated 3 months ago • ❤️ 62 • 👁 29.2k • ⬇ 1.91k • 🔗 11

ChatPromptTemplate **Summarization** **English** **openai:gpt-3.5-turbo** **Try it**

hardkothari/prompt-maker

Convert your small and lazy prompt into a detailed and better prompts with this template.

{x} Prompt • Updated 3 months ago • ❤️ 52 • 👁 14k • ⬇ 1.47k • 🔗 1

ChatPromptTemplate **Chatbots** **Agents** **QA over documents** **Self-checking** **English** **openai:gpt-3.5-turbo** **openai:gpt-4** **Try it**

smithing-gold/assumption-checker

Assert whether assumptions are made in a user's query and provide follow up questions to debunk their claims.

Access the LangChain Hub at: <https://smith.langchain.com/hub>



LangChain Templates

[+ Request a template](#)

Featured

rag

OpenAI

Pinecone

rag-conversation

by Elastic

Conversational RAG using Pinecone

 Github

 11

extraction

OpenAI

Function Calling

extraction-openai-functions

by LangChain

Use OpenAI function calling for tasks like...

 Github

 12

agent

Anthropic

xml-agent

by LangChain

Agent that uses XML syntax to communicat...

 Github

 6

rag

OpenAI

Chroma

Gpt4all

rag-chroma-private

by LangChain

Private RAG using local LLM, embeddings,...

 Github

 14

research

OpenAI

Tavily

openai-functions-agent

by LangChain

Agent using OpenAI function calling to...

 Github

 11

The LangChain ecosystem



LangSmith: troubleshooting and evaluating applications

LangServe: deploying applications

LangGraph: multi-agent knowledge graphs

Let's practice!

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN