Integrating document loaders

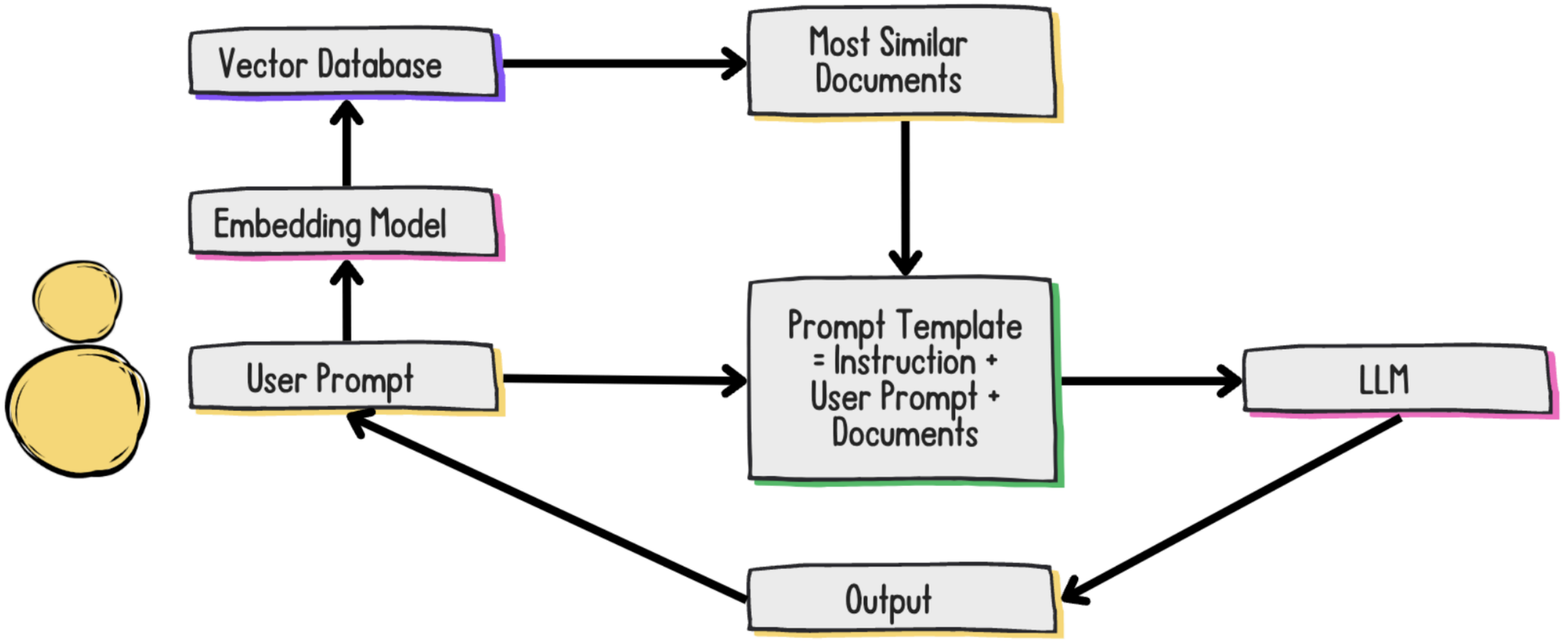
DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Jonathan Bennion
AI Engineer & LangChain Contributor

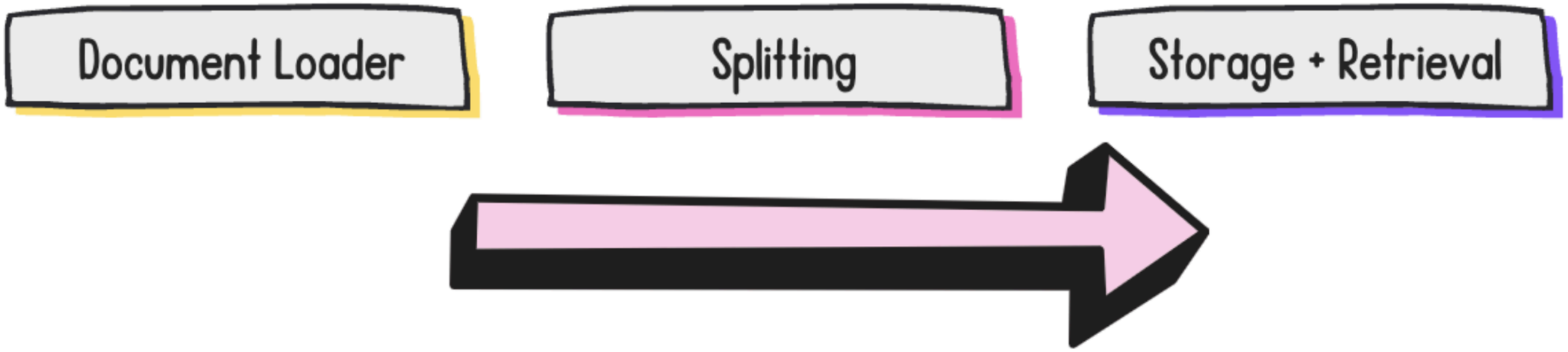
datacamp

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

- *Requires installation of the `unstructured` package: `pip install unstructured`*

```
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

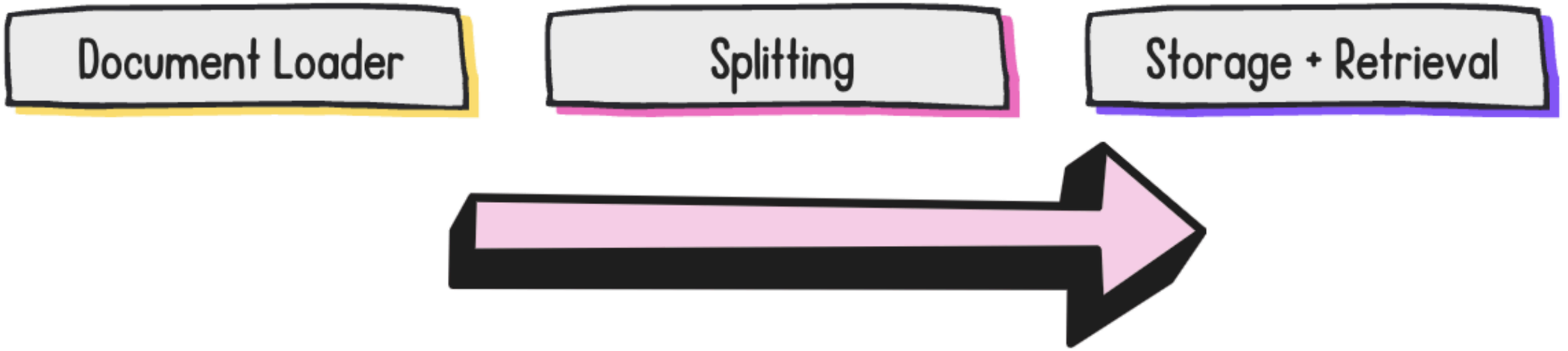
Splitting external data for retrieval

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Jonathan Bennion
AI Engineer & LangChain Contributor

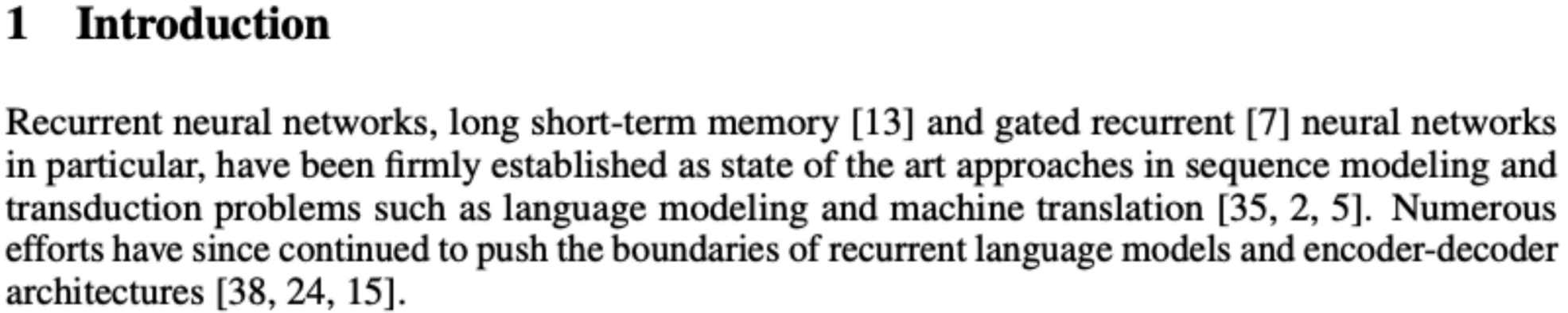
datacamp

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:

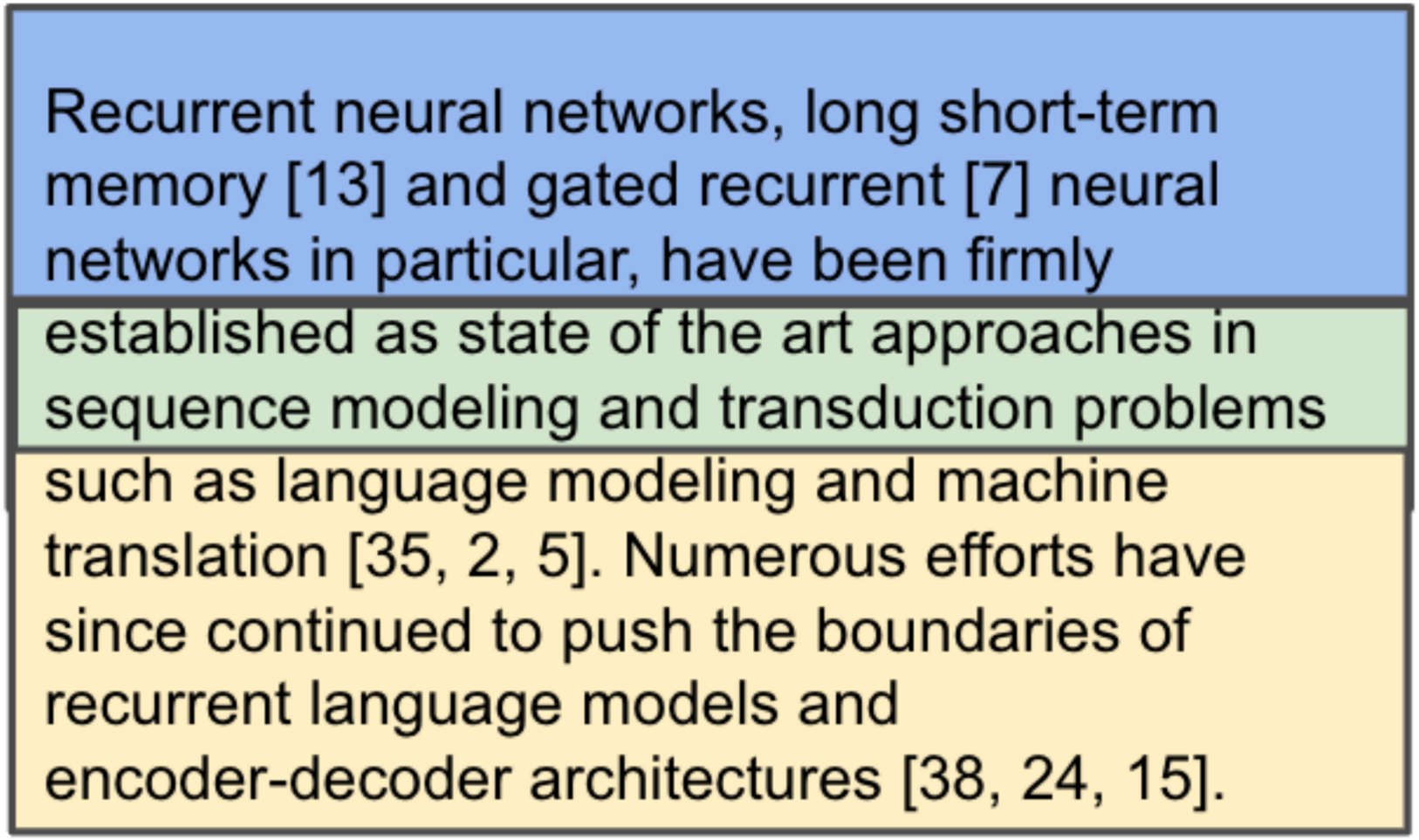
```
Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks
```

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



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_splitters 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_splitters 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_splitters 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

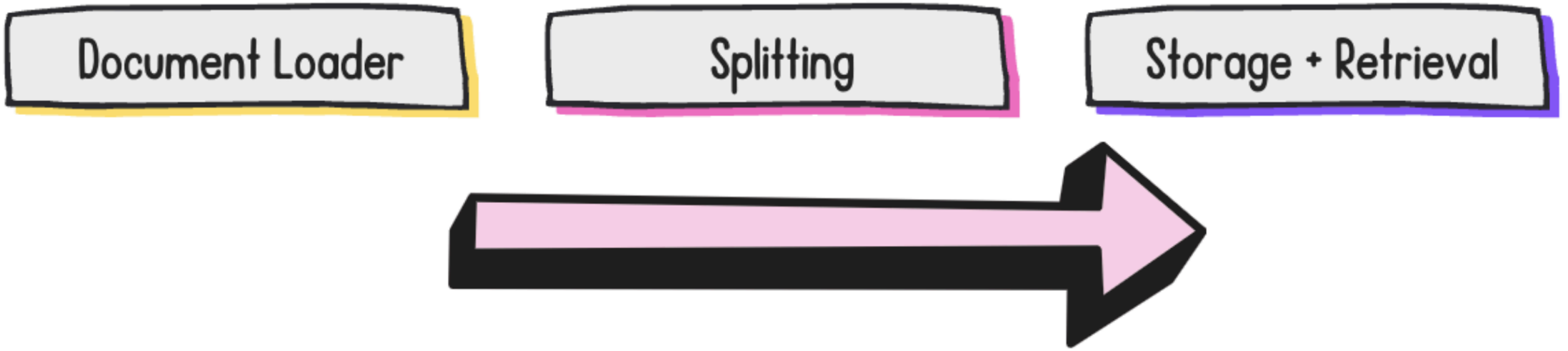
RAG storage and retrieval using vector databases

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Jonathan Bennion
AI Engineer & LangChain Contributor

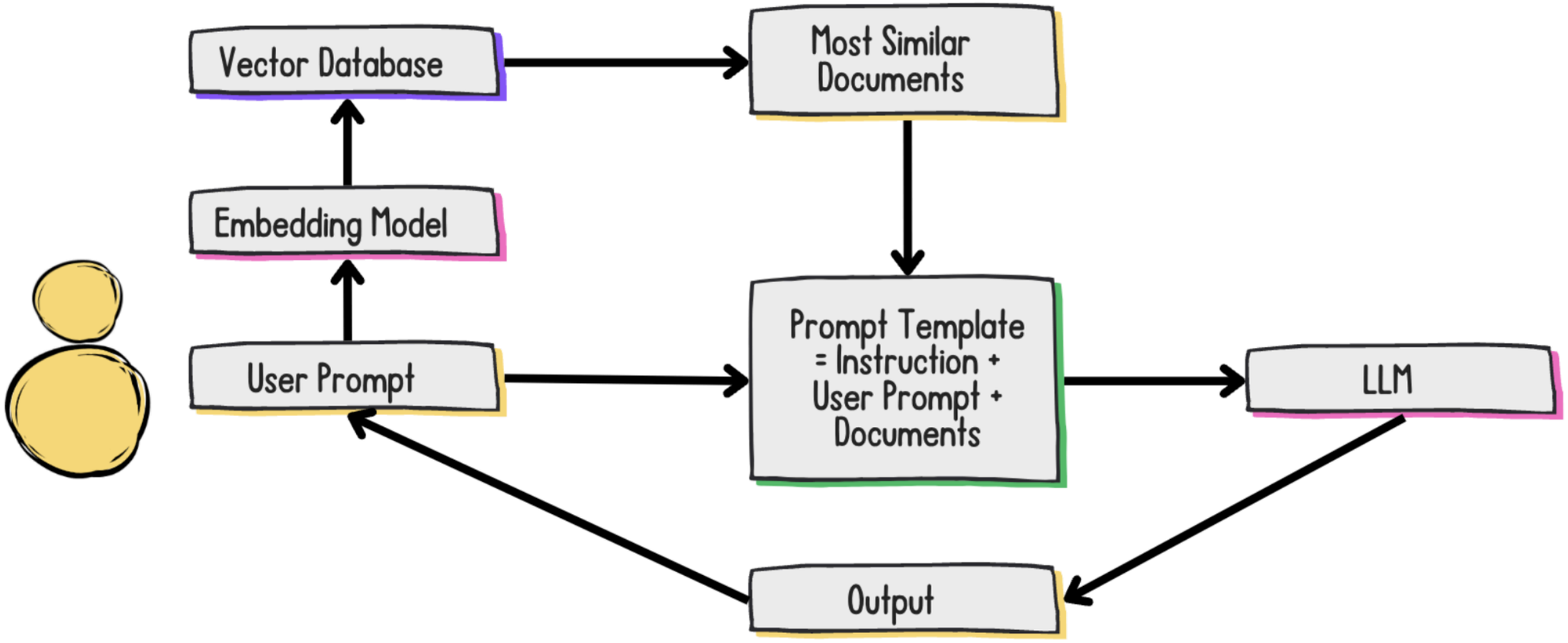
datacamp

RAG development steps

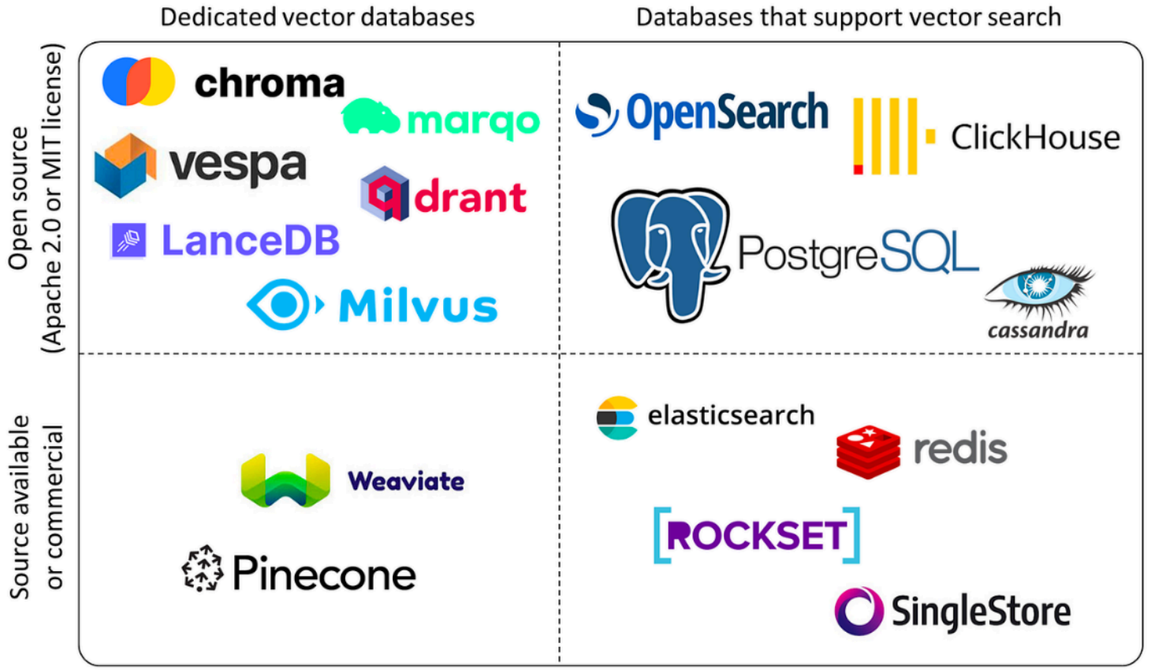


- **Focus of this video:** *storage* and *retrieval*

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(api_key=openai_api_key, model='text-embedding-3-small')

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

```
from langchain_core.prompts import ChatPromptTemplate

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!

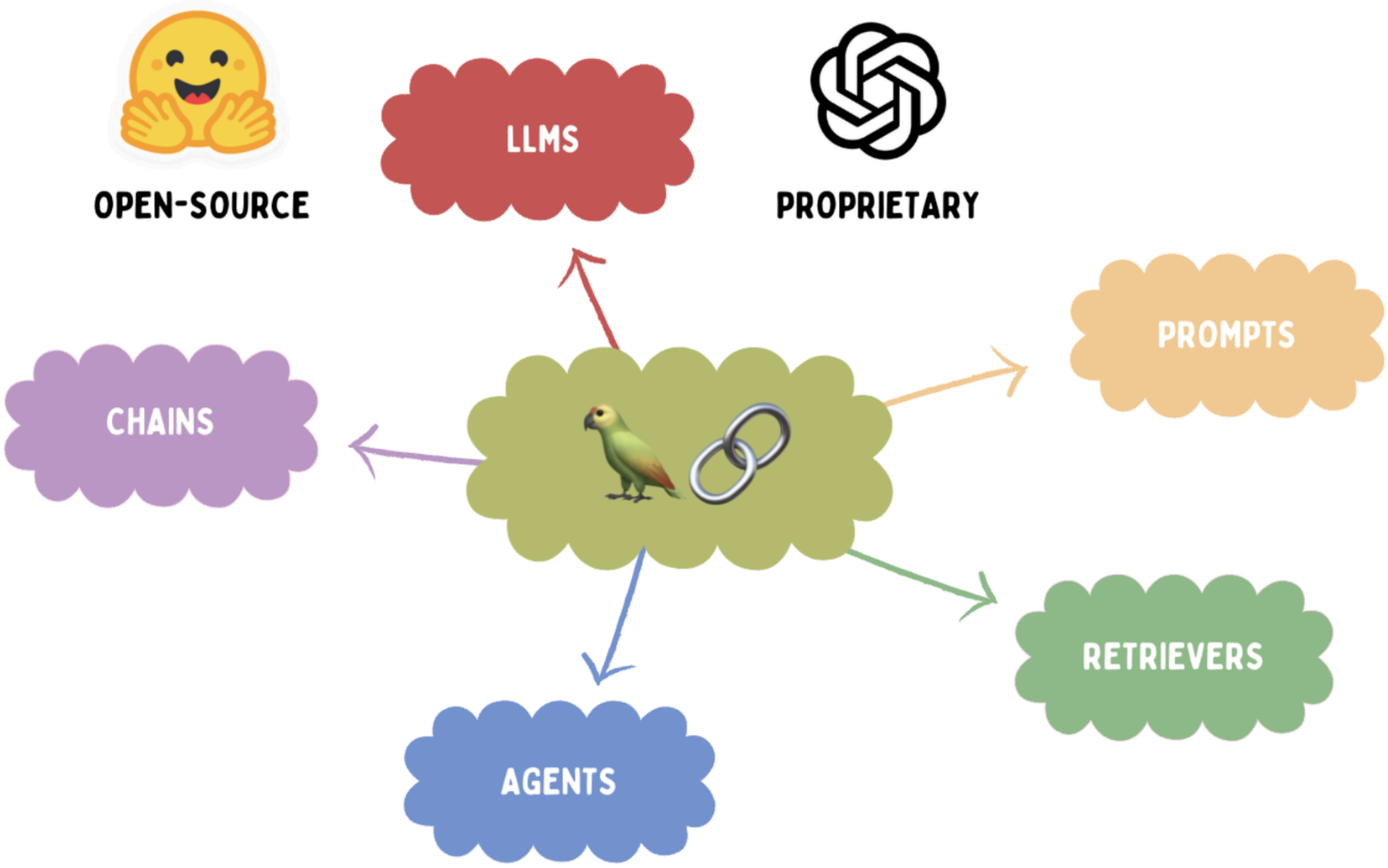
DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Wrap-up!

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

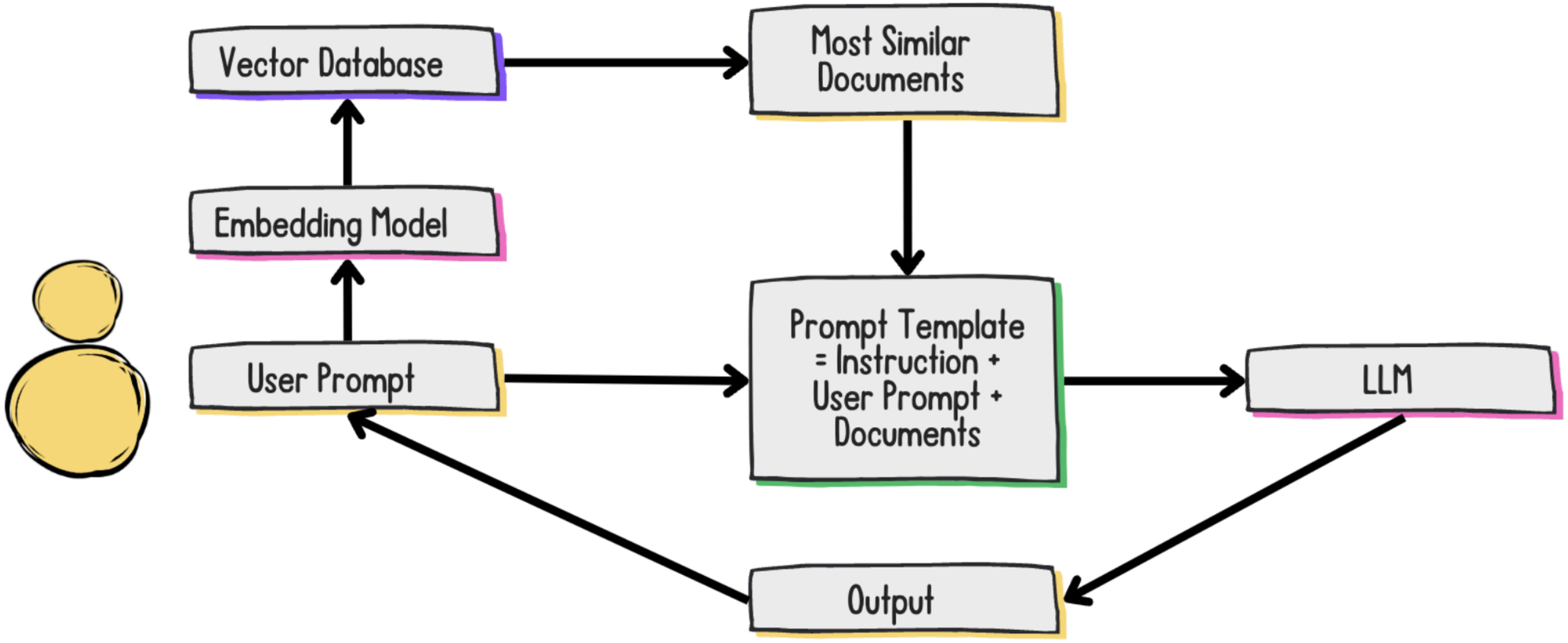
Jonathan Bennion
AI Engineer & LangChain Contributor

LangChain's core components

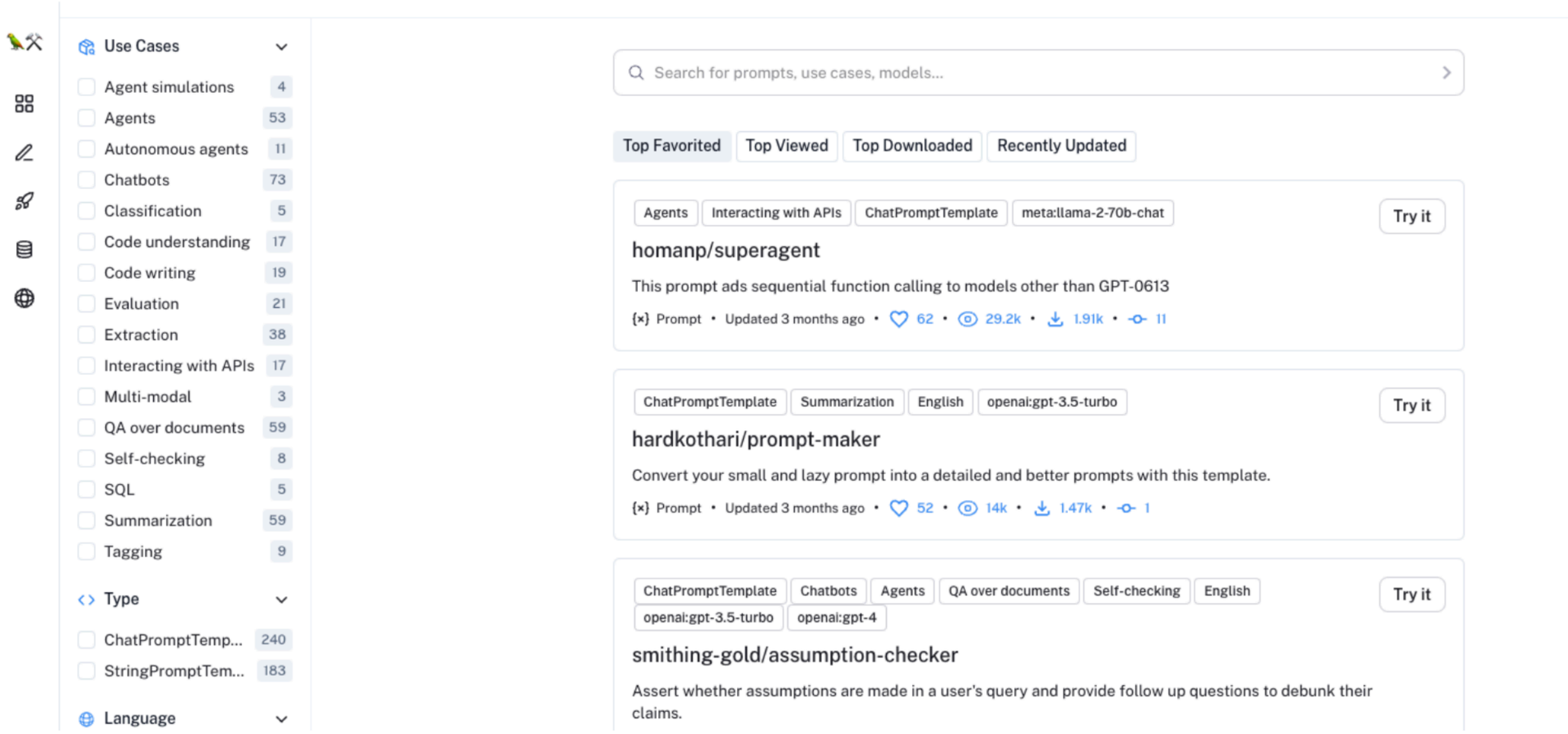


Chains and agents

Retrieval Augmented Generation (RAG)



LangChain Hub



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