The LangChain ecosystem

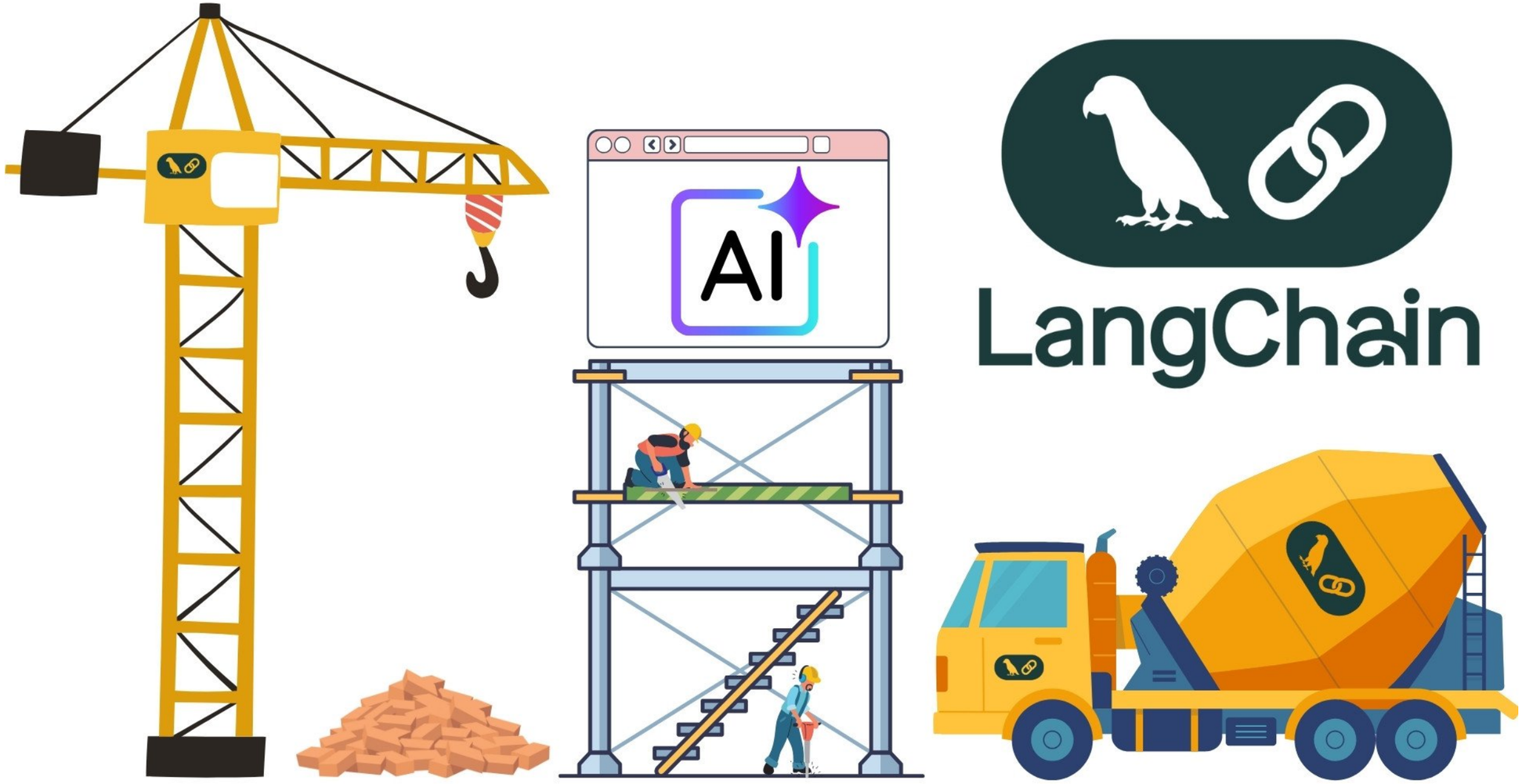
DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

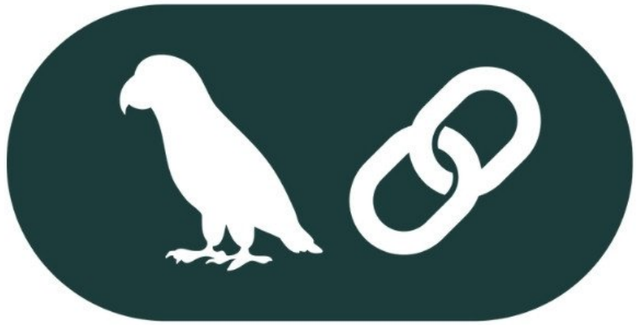
Jonathan Bennion
AI Engineer & LangChain Contributor

Meet your instructor...

- Jonathan Bennion, **AI Engineer**
- ML & AI at Facebook, Google, Amazon, Disney, EA
- Created *Logical Fallacy chain* in LangChain
- Contributor to **DeepEval**

Build LLM Apps with LangChain



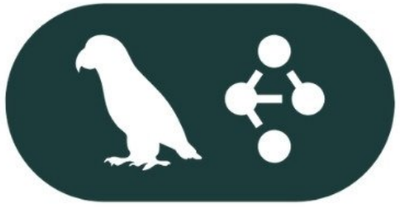


LangChain

LangSmith

=

Deploying Apps



LangGraph

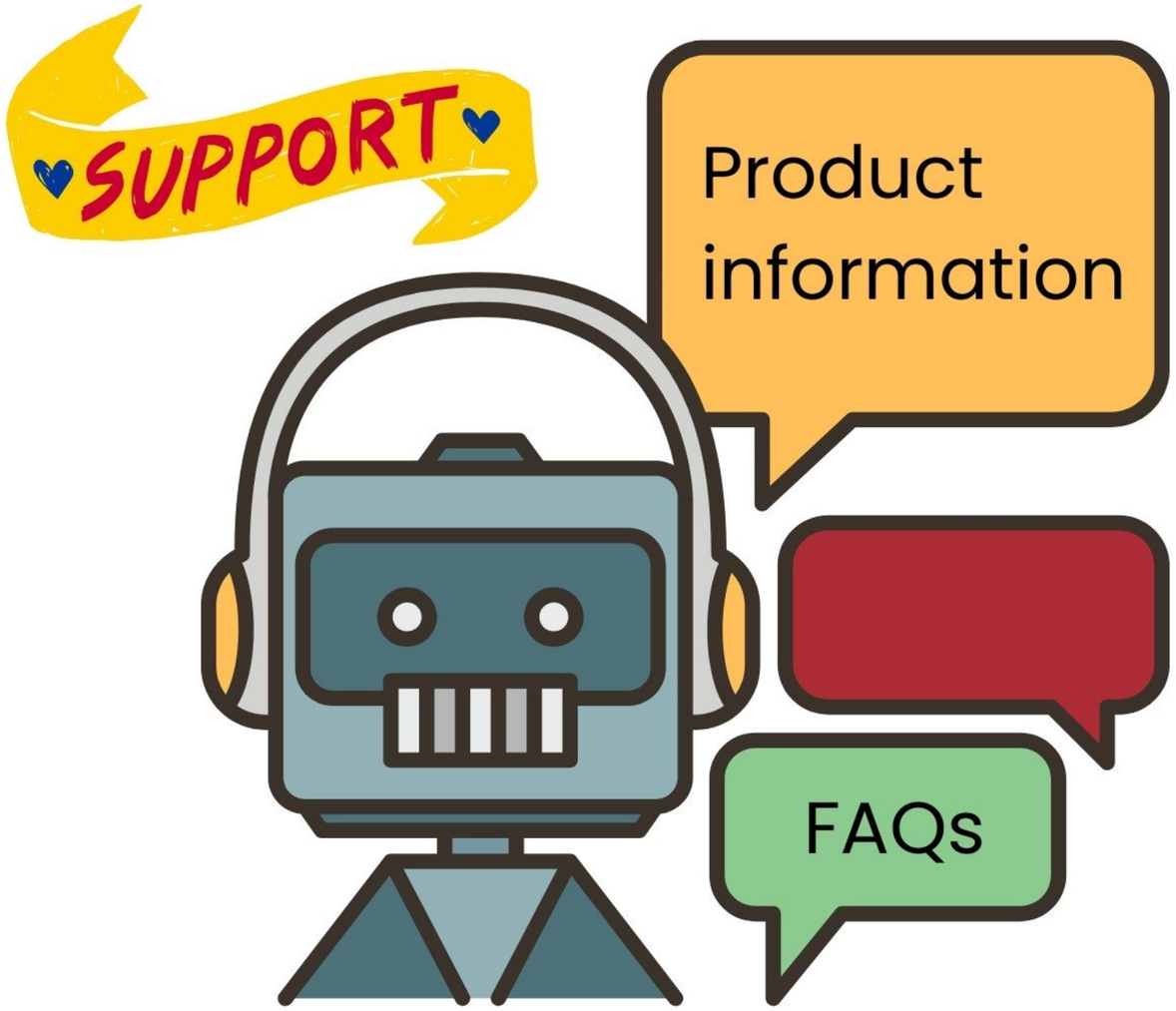
=

AI Agents

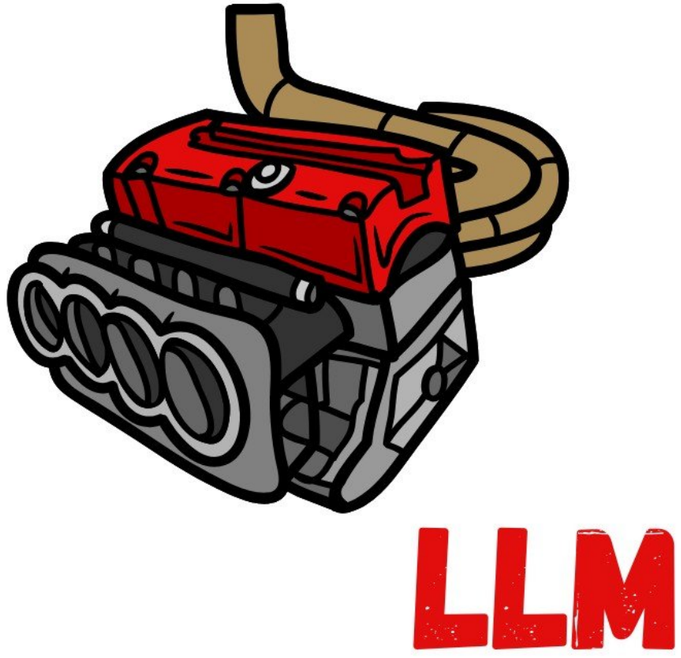
LangChain integrations

[1] https://python.langchain.com/docs/integrations/providers/

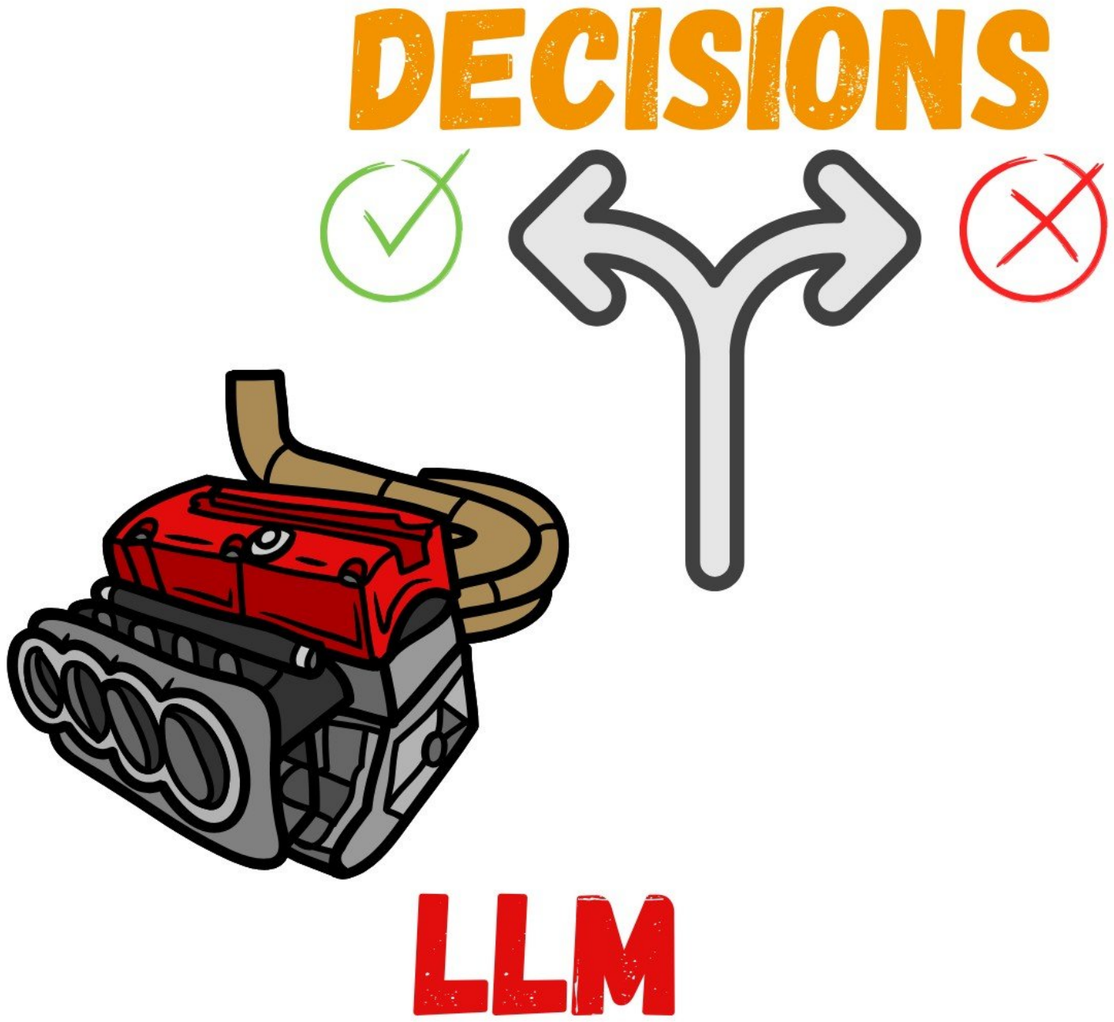
Building LLM apps the LangChain way...



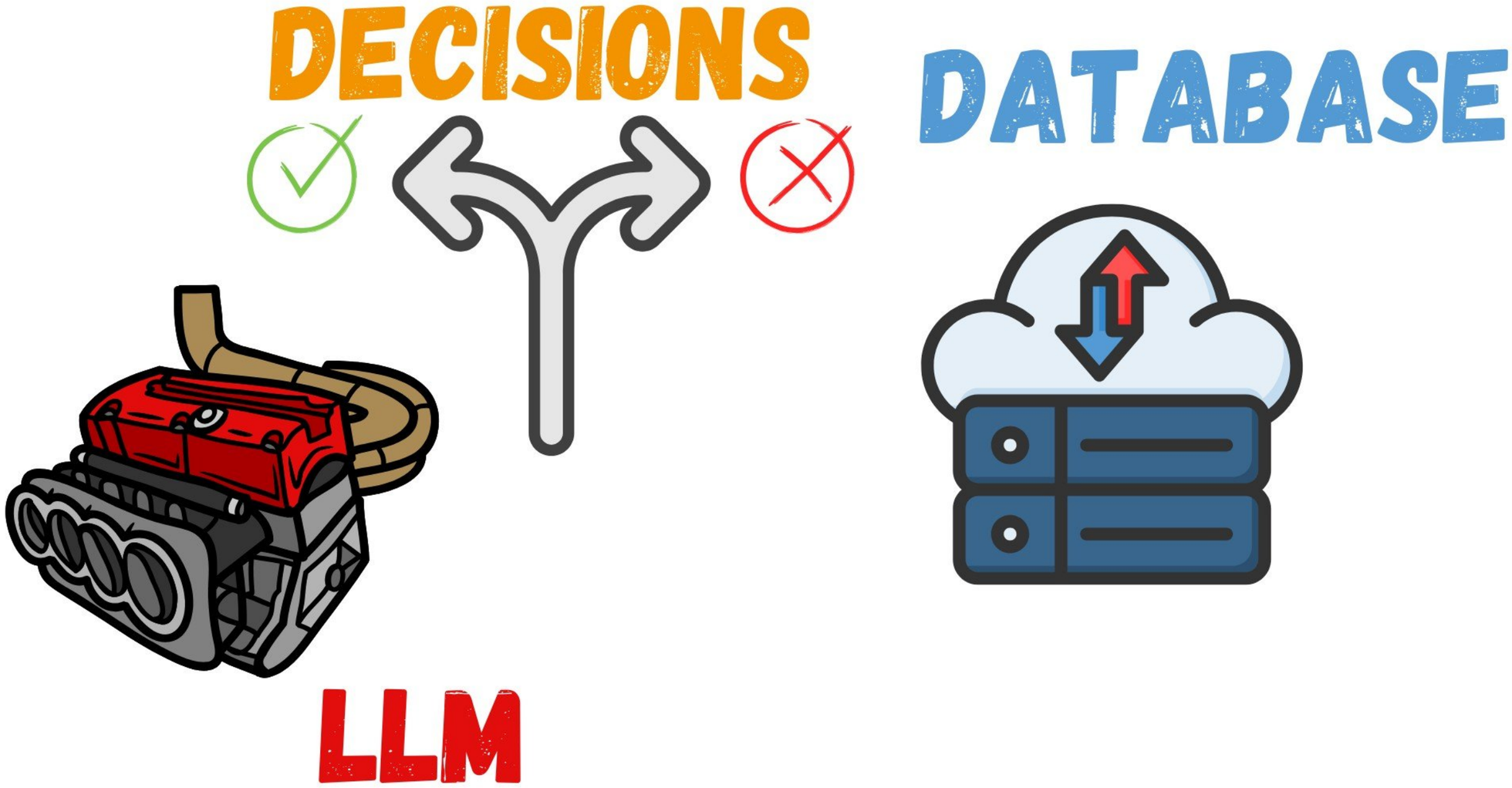
Building LLM apps the LangChain way...



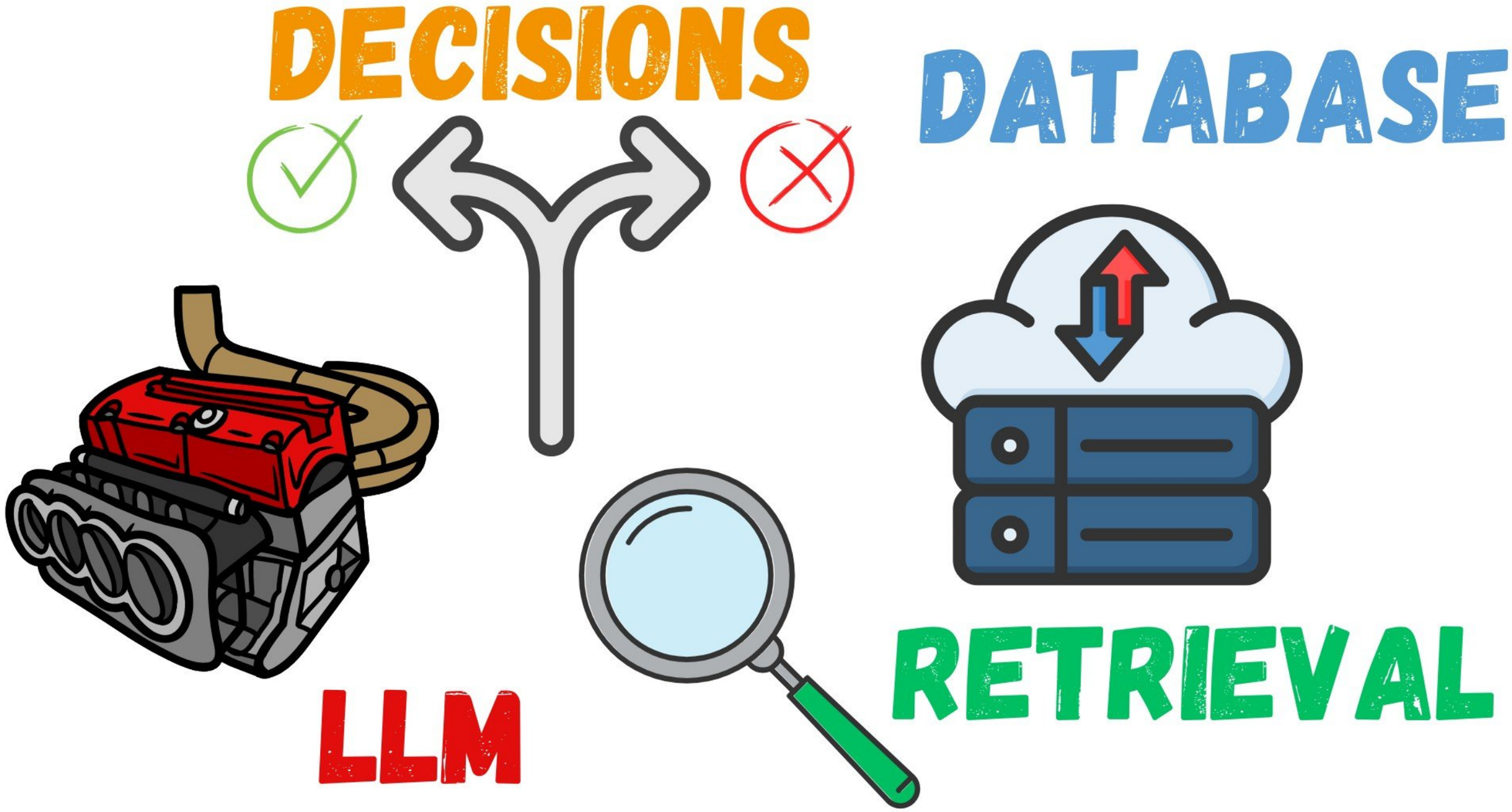
Building LLM apps the LangChain way...



Building LLM apps the LangChain way...



Building LLM apps the LangChain way...



Prompting OpenAI models

```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(
    model="gpt-4o-mini",
    api_key='...'
)

llm.invoke("What is LangChain?")
```

```
LangChain is a framework designed for developing applications...
```

- Additional parameters: `max_completion_tokens` , `temperature` , etc.

[1] https://platform.openai.com/docs/quickstart

🤗 Prompting Hugging Face models

```
from langchain_huggingface import HuggingFacePipeline

llm = HuggingFacePipeline.from_model_id(
    model_id="meta-llama/Llama-3.2-3B-Instruct",
    task="text-generation",
    pipeline_kwargs={"max_new_tokens": 100}
)

llm.invoke("What is Hugging Face?")
```

```
Hugging Face is a popular open-source artificial intelligence (AI) library...
```

Let's practice!

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Prompt templates

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Jonathan Bennion
AI Engineer & LangChain Contributor

datacamp

Prompt templates

- **Recipes** for defining prompts for LLMs
- Can contain: instructions, examples, and additional context

Prompt templates

```
from langchain_core.prompts import PromptTemplate

template = "Expain this concept simply and concisely: {concept}"
prompt_template = PromptTemplate.from_template(
    template=template
)

prompt = prompt_template.invoke({"concept": "Prompting LLMs"})
print(prompt)
```

```
text='Expain this concept simply and concisely: Prompting LLMs'
```

```
llm = HuggingFacePipeline.from_model_id(
    model_id="meta-llama/Llama-3.3-70B-Instruct",
    task="text-generation"
)
llm_chain = prompt_template | llm

concept = "Prompting LLMs"
print(llm_chain.invoke({"concept": concept}))
```

```
Prompting LLMs (Large Language Models) refers to the process of giving a model a specific input or question to generate a response.
```

- LangChain Expression Language (**LCEL**): `|` (pipe) operator
- **Chain**: connect calls to different components

Chat models

- **Chat roles:** `system` , `human` , `ai`

```
from langchain_core.prompts import ChatPromptTemplate

template = ChatPromptTemplate.from_messages(
    [
        ("system", "You are a calculator that responds with math."),
        ("human", "Answer this math question: What is two plus two?"),
        ("ai", "2+2=4"),
        ("human", "Answer this math question: {math}")
    ]
)
```

Integrating ChatPromptTemplate

```
llm = ChatOpenAI(model="gpt-4o-mini", api_key='<OPENAI_API_TOKEN>')


llm_chain = template | llm
math='What is five times five?'


response = llm_chain.invoke({"math": math})
print(response.content)
```

```
5x5=25
```

Let's practice!

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Few-shot prompting

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Jonathan Bennion
AI Engineer & LangChain Contributor

Limitations of standard prompt templates

- `PromptTemplate` + `ChatPromptTemplate`
- Handling small numbers of examples
- Don't scale for many examples
- `FewShotPromptTemplate`

```
examples = [
    {
        "question": "..."
        "answer": "..."
    },
    ...
]
```

Building an example set

```
examples = [
    {
        "question": "Does Henry Campbell have any pets?",
        "answer": "Henry Campbell has a dog called Pluto."
    },
    ...
]
```

```
# Convert pandas DataFrame to list of dicts
examples = df.to_dict(orient="records")
```

Formatting the examples

```
from langchain_core.prompts import FewShotPromptTemplate, PromptTemplate


example_prompt = PromptTemplate.from_template("Question: {question}\n{answer}")
```

```
prompt = example_prompt.invoke({"question": "What is the capital of Italy?"
                                "answer": "Rome"})
print(prompt.text)
```

```
Question: What is the capital of Italy?
Rome
```

FewShotPromptTemplate

```
prompt_template = FewShotPromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
    suffix="Question: {input}",
    input_variables=["input"]
)
```

- `examples` : the list of dicts
- `example_prompt` : formatted template
- `suffix` : suffix to add to the input
- `input_variables`

Invoking the few-shot prompt template

```
prompt = prompt_template.invoke({"input": "What is the name of Henry Campbell's dog?"})
print(prompt.text)
```

```
Question: Does Henry Campbell have any pets?
Henry Campbell has a dog called Pluto.
...

Question: What is the name of Henry Campbell's dog?
```

Integration with a chain

```
llm = ChatOpenAI(model="gpt-4o-mini", api_key="...")


llm_chain = prompt_template | llm
response = llm_chain.invoke({"input": "What is the name of Henry Campbell's dog?"})
print(response.content)
```

```
The name of Henry Campbell's dog is Pluto.
```

Let's practice!

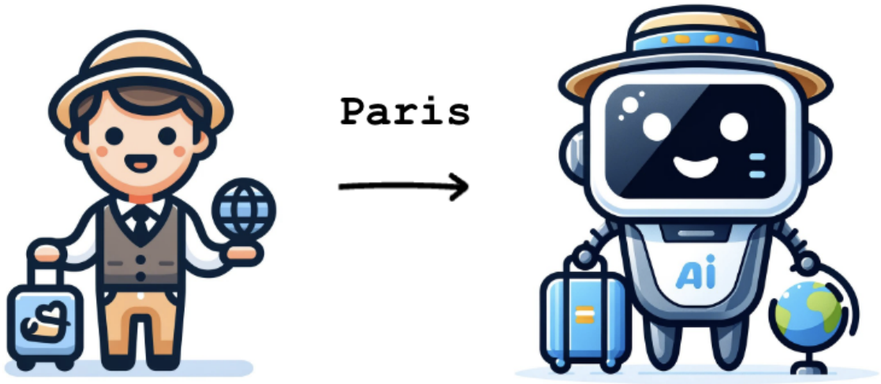
DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

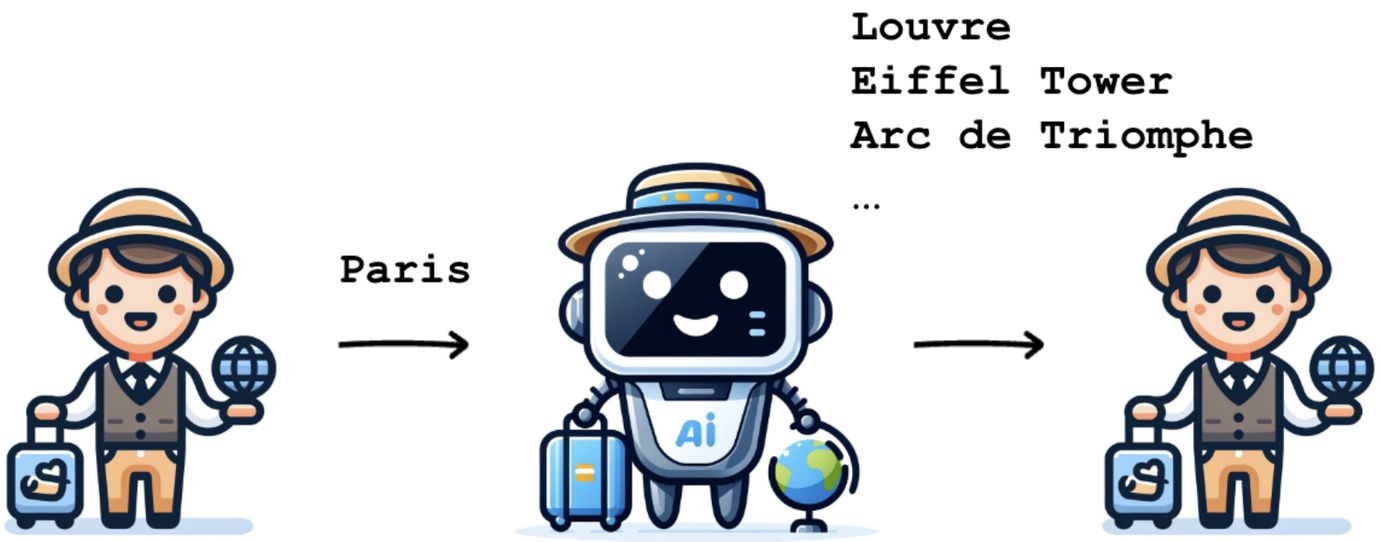
Sequential chains

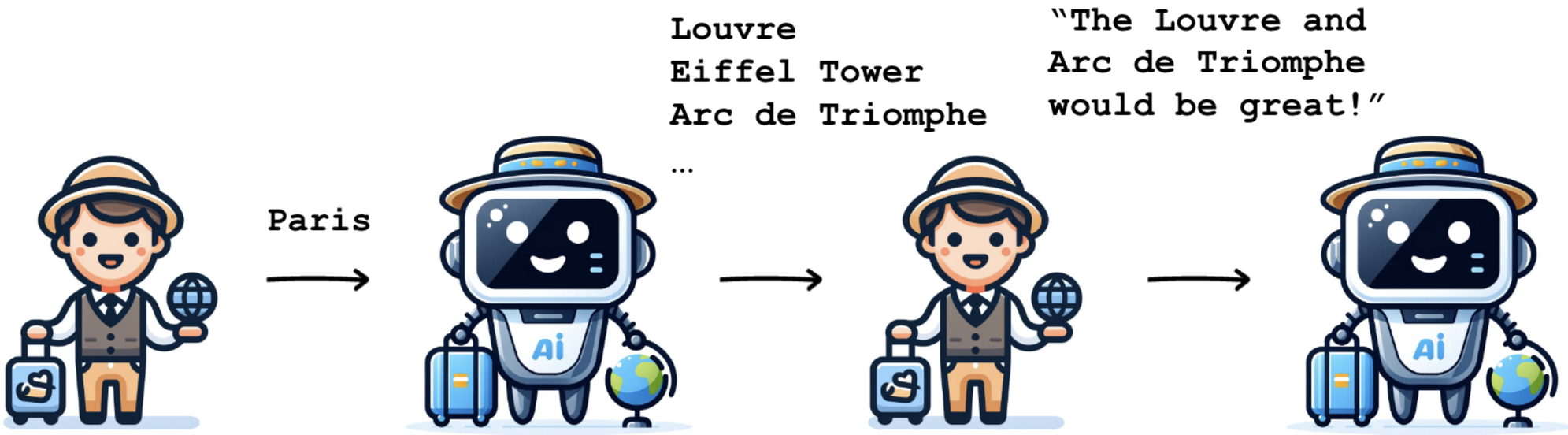
DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

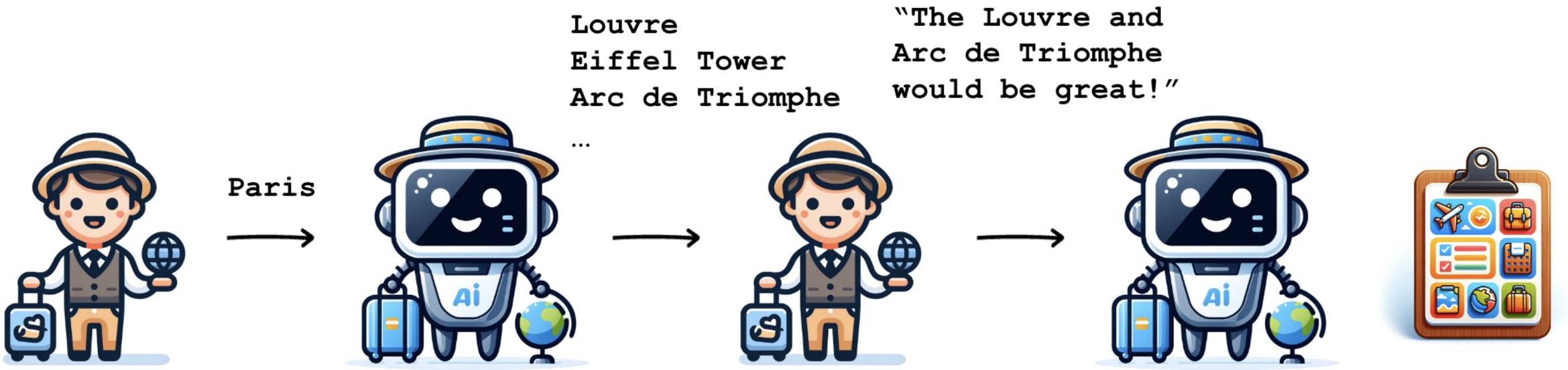
Jonathan Bennion
AI Engineer & LangChain Contributor

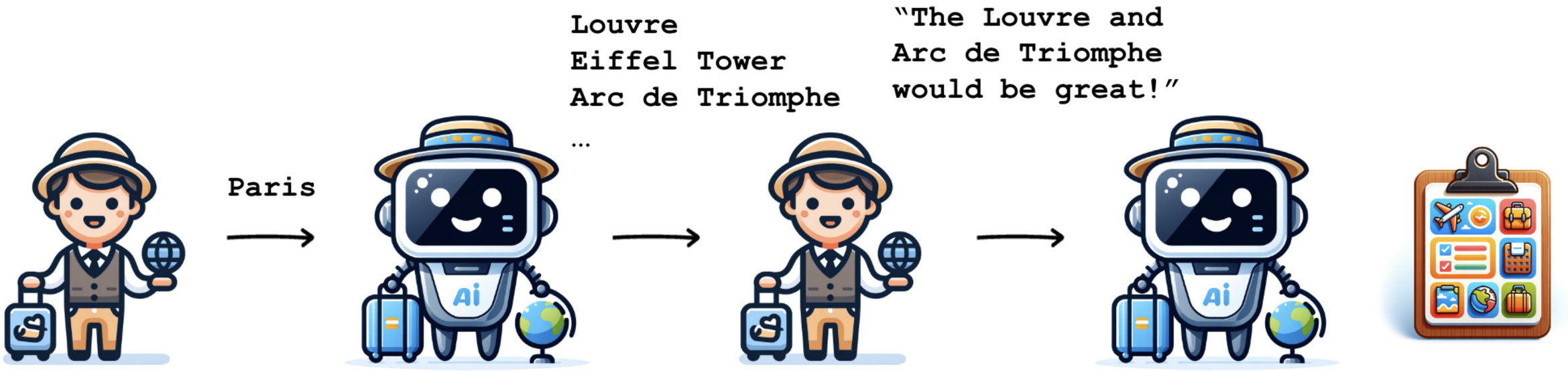

Ai


Paris
Ai


Paris
Louvre
Eiffel Tower
Arc de Triomphe
...
Ai


Paris
Louvre
Eiffel Tower
Arc de Triomphe
...
"The Louvre and
Arc de Triomphe
would be great!"
Ai
Ai


Paris
Louvre
Eiffel Tower
Arc de Triomphe
...
"The Louvre and
Arc de Triomphe
would be great!"
Ai
Ai



SEQUENTIAL PROBLEM

Sequential chains

- Output → input

```
destination_prompt = PromptTemplate(
    input_variables=["destination"],
    template="I am planning a trip to {destination}. Can you suggest some activities to do there?"
)
activities_prompt = PromptTemplate(
    input_variables=["activities"],
    template="I only have one day, so can you create an itinerary from your top three activities: {activities}."
)

llm = ChatOpenAI(model="gpt-4o-mini", api_key=openai_api_key)

seq_chain = ({"activities": destination_prompt | llm | StrOutputParser()}
    | activities_prompt
    | llm
    | StrOutputParser())
```

```
print(seq_chain.invoke({"destination": "Rome"}))
```

```
- Morning:
1. Start your day early with a visit to the Colosseum. Take a guided tour to learn about its history and significance.
2. After exploring the Colosseum, head to the Roman Forum and Palatine Hill to see more of ancient Rome's ruins.

- Lunch:
3. Enjoy a delicious Italian lunch at a local restaurant near the historic center.

- Afternoon:
4. Visit the Vatican City and explore St. Peter's Basilica, the Vatican Museums, and the Sistine Chapel.
5. Take some time to wander through the charming streets of Rome, stopping at landmarks like the Pantheon, Trevi Fountain, and Piazza Navona.

- Evening:
6. Relax in one of Rome's beautiful parks, such as Villa Borghese or the Orange Garden, for a peaceful escape from the bustling city.
7. End your day with a leisurely dinner at a local restaurant, indulging in more Italian cuisine and maybe some gelato.
```

Let's practice!

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Introduction to LangChain agents

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Jonathan Bennion
AI Engineer & LangChain Contributor

datacamp

What are agents?

Agents: use LLMs to take *actions*

Tools: *functions* called by the agent

- **Now** → *ReAct* Agent

ReAct agents

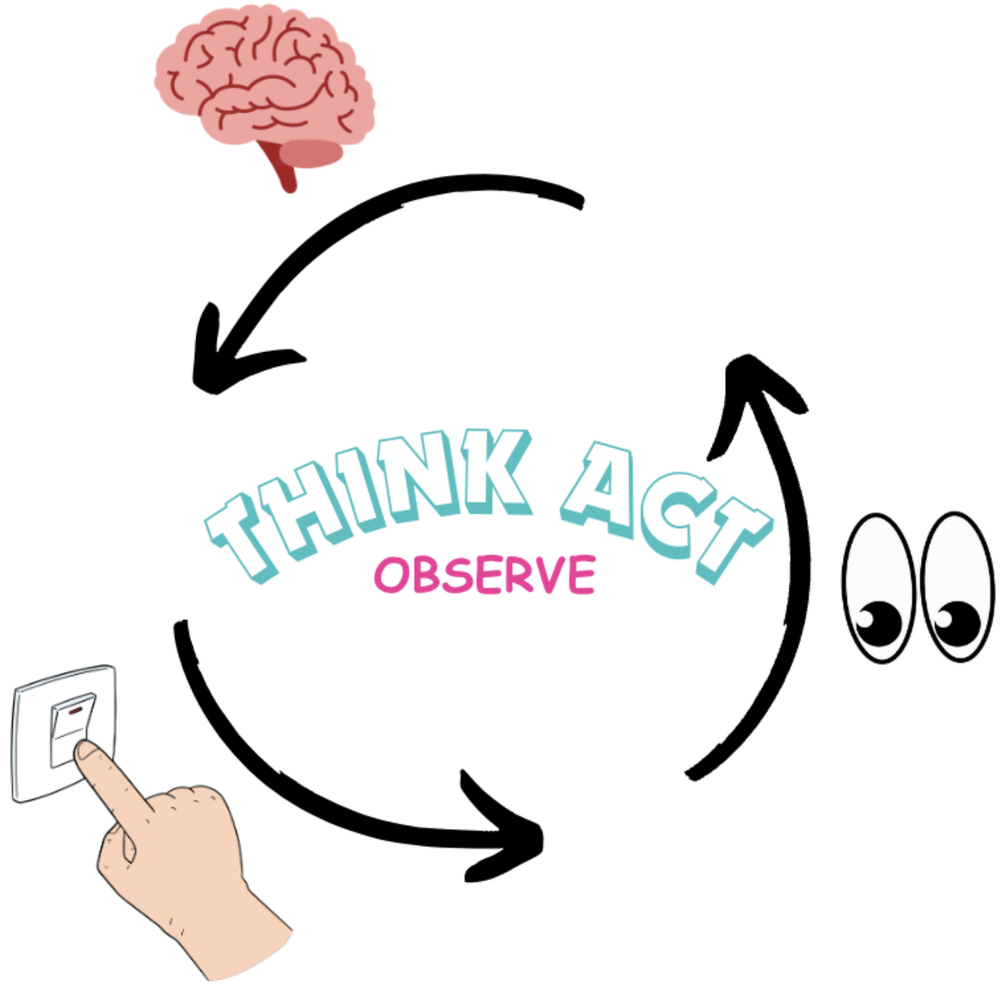
- Reason + Act

What is the weather like in Kingston, Jamaica?

```
Thought: I should call Weather() to find the
weather in Kingston, Jamaica.

Act: Weather("Kingston, Jamaica")

Observe: The weather is mostly sunny with
temperatures of 82°F.
```



LangGraph

- Branch of LangChain centered around designing ***agent systems***
- Unified, tool-agnostic syntax
- `pip install langgraph==0.2.74`

ReAct agent

```
from langgraph.prebuilt import create_react_agent
from langchain_community.agent_toolkits.load_tools import load_tools

llm = ChatOpenAI(model="gpt-4o-mini", api_key=openai_api_key)
tools = load_tools(["llm-math"], llm=llm)
agent = create_react_agent(llm, tools)

messages = agent.invoke({"messages": [("human", "What is the square root of 101?")]})
print(messages)
```

ReAct agent

```
{'messages': [
    HumanMessage(content='What is the square root of 101?', ...),
    AIMessage(content='', ..., tool_calls=[{'name': 'Calculator', 'args': {'__arg1': 'sqrt(101)'}, ...),
    ToolMessage(content='Answer: 10.049875621120889', ...),
    AIMessage(content='The square root of 101 is approximately 10.05.', ...)
]}
```

```
print(messages['messages'][-1].content)
```

```
The square root of 101 is approximately 10.05.
```

Let's practice!

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Custom tools for agents

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Jonathan Bennion
AI Engineer & LangChain Contributor

datacamp

Tool formats

```
from langchain_community.agent_toolkits.load_tools import load_tools


tools = load_tools(["llm-math"], llm=llm)
print(tools[0].name)
```

```
Calculator
```

```
print(tools[0].description)
```

```
Useful for when you need to answer questions about math.
```

- Used by LLM/agent as *context* to determine when to call it

Tool formats

```
print(tools[0].return_direct)
```

```
False
```

Defining a custom function

```
def financial_report(company_name: str, revenue: int, expenses: int) -> str:
    """Generate a financial report for a company that calculates net income."""
    net_income = revenue - expenses

    report = f"Financial Report for {company_name}:\n"
    report += f"Revenue: ${revenue}\n"
    report += f"Expenses: ${expenses}\n"
    report += f"Net Income: ${net_income}\n"
    return report
```

Calling the function

```
print(financial_report(company_name="LemonadeStand", revenue=100, expenses=50))
```

```
Financial Report for LemonadeStand:
Revenue: $100
Expenses: $50
Net Income: $50
```

From functions to tools

```python
from langchain_core.tools import tool


@tool
def financial_report(company_name: str, revenue: int, expenses: int) -> str:
    """Generate a financial report for a company that calculates net income."""
    net_income = revenue - expenses

    report = f"Financial Report for {company_name}:\n"
    report += f"Revenue: ${revenue}\n"
    report += f"Expenses: ${expenses}\n"
    report += f"Net Income: ${net_income}\n"
    return report
```

Examining our new tool

```
print(financial_report.name)
print(financial_report.description)
print(financial_report.return_direct)
print(financial_report.args)
```

```
financial_report
Generate a financial report for a company that calculates net income.
False
{'company_name': {'title': 'Company Name', 'type': 'string'},
 'revenue': {'title': 'Revenue', 'type': 'integer'},
 'expenses': {'title': 'Expenses', 'type': 'integer'}}
```

Integrating the custom tool

```
from langgraph.prebuilt import create_react_agent

llm = ChatOpenAI(model="gpt-4o-mini", api_key=openai_api_key, temperature=0)
agent = create_react_agent(llm, [financial_report])

messages = agent.invoke({"messages": [("human", "TechStack generated made $10 millior
with $8 million of costs. Generate a financial report.")]})
print(messages)
```

Integrating the custom tool

```
{'messages': [
    HumanMessage(content='TechStack generated made $10 million dollars with $8 million of...', ...),
    AIMessage(content='', ..., tool_calls=[{'name': 'financial_report',
                                            'args': {'company_name': 'TechStack',
                                                     'revenue': 10000000, 'expenses': 8000000}, ...),
    ToolMessage(content='Financial Report for TechStack:\nRevenue: $10000000\nExpenses...', ...),
    AIMessage(content='Here is the financial report for TechStack...', ...)
]}
```

Tool outputs

```
print(messages['messages'][-1].content)
```

```
Here is the financial report for TechStack:
- Revenue: $10,000,000
- Expenses: $8,000,000
- Net Income: $2,000,000
```

```
Financial Report for TechStack:
Revenue: $10000000
Expenses: $8000000
Net Income: $2000000
```

Let's practice!

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Integrating document loaders

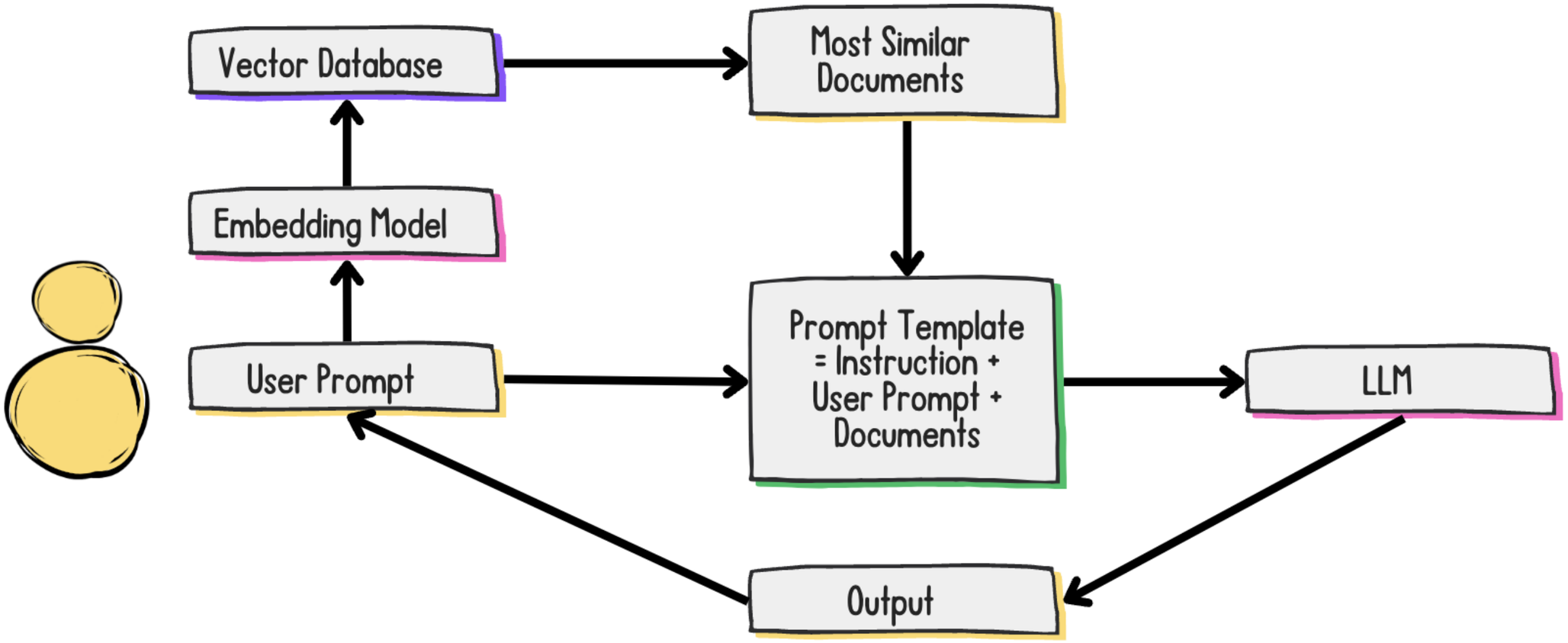
DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Jonathan Bennion
AI Engineer & LangChain Contributor

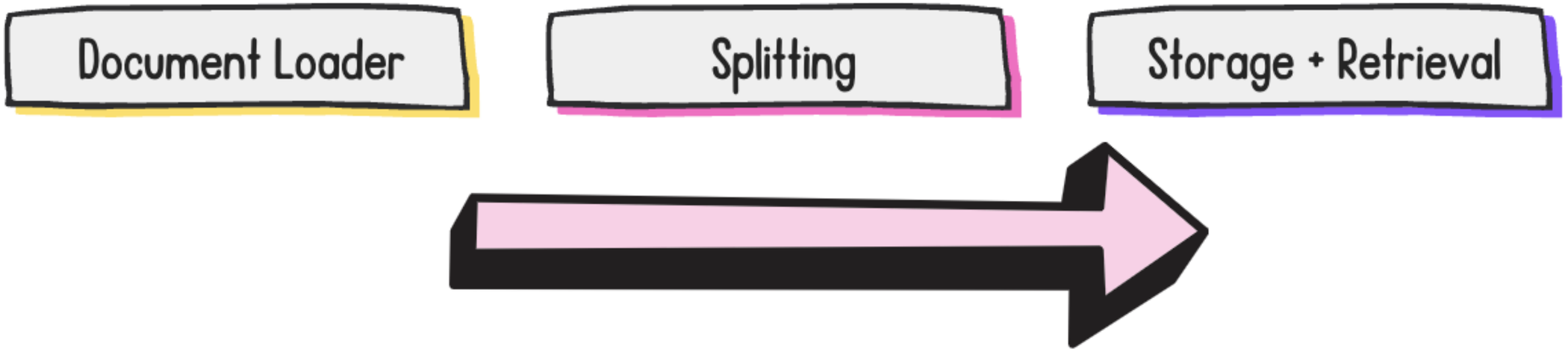
datacamp

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

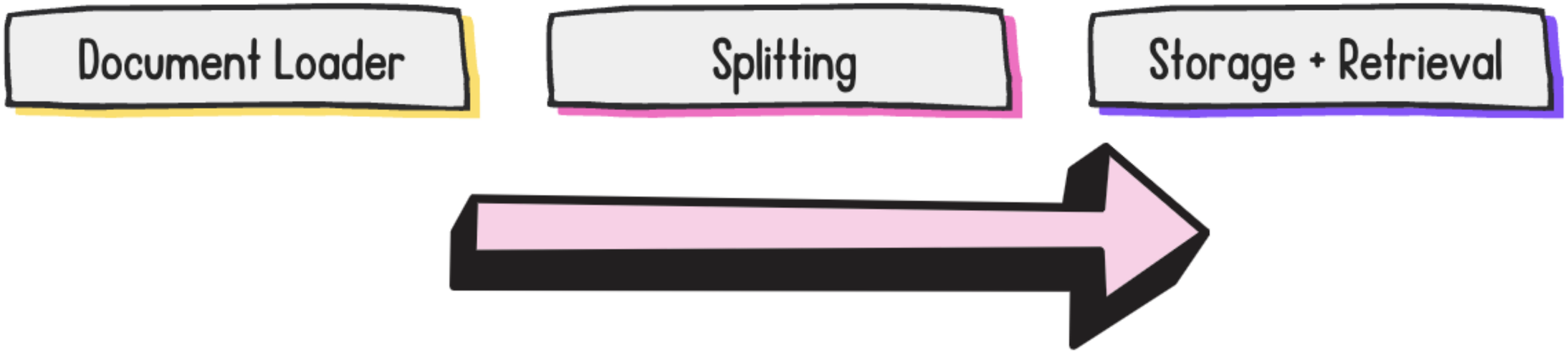
Splitting external data for retrieval

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:

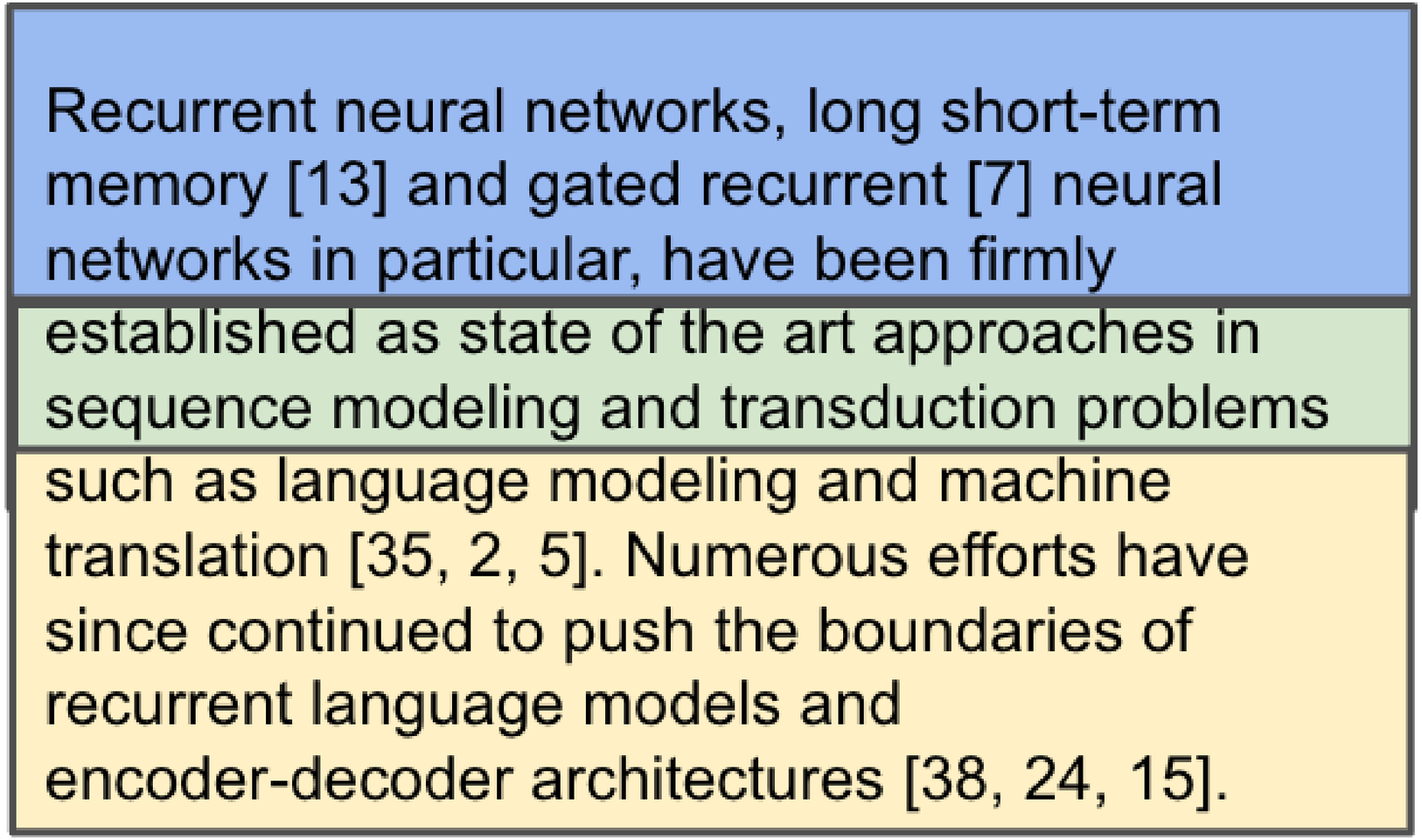
```
Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks
```

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

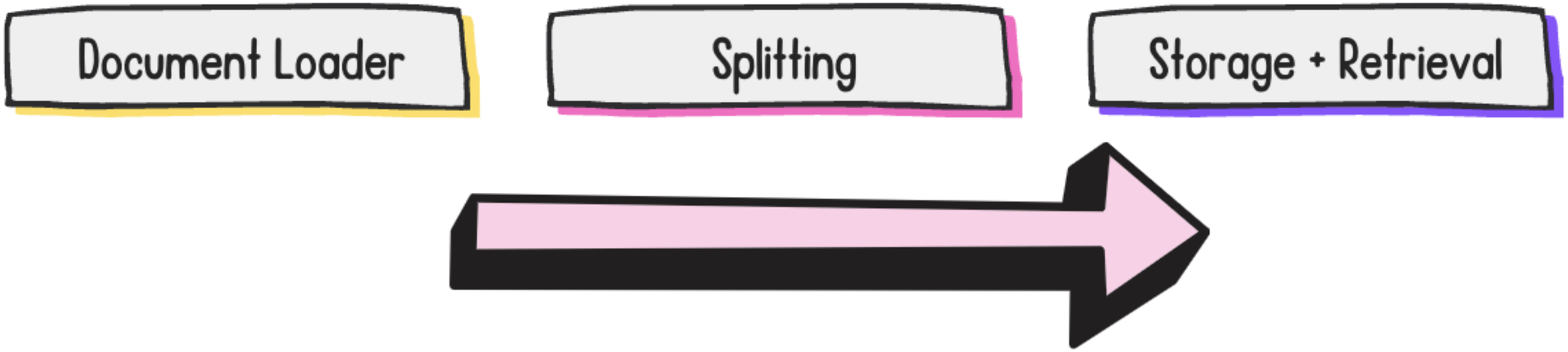
RAG storage and retrieval using vector databases

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Jonathan Bennion
AI Engineer & LangChain Contributor

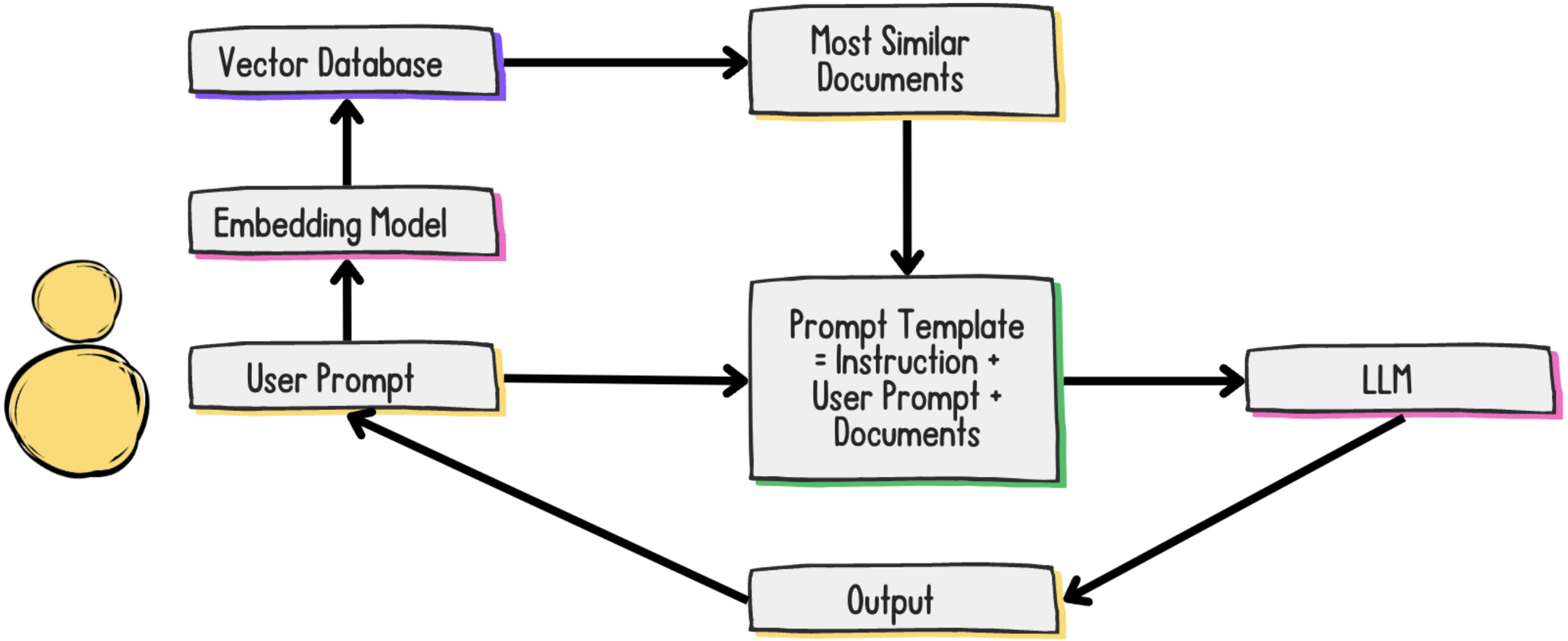
datacamp

RAG development steps

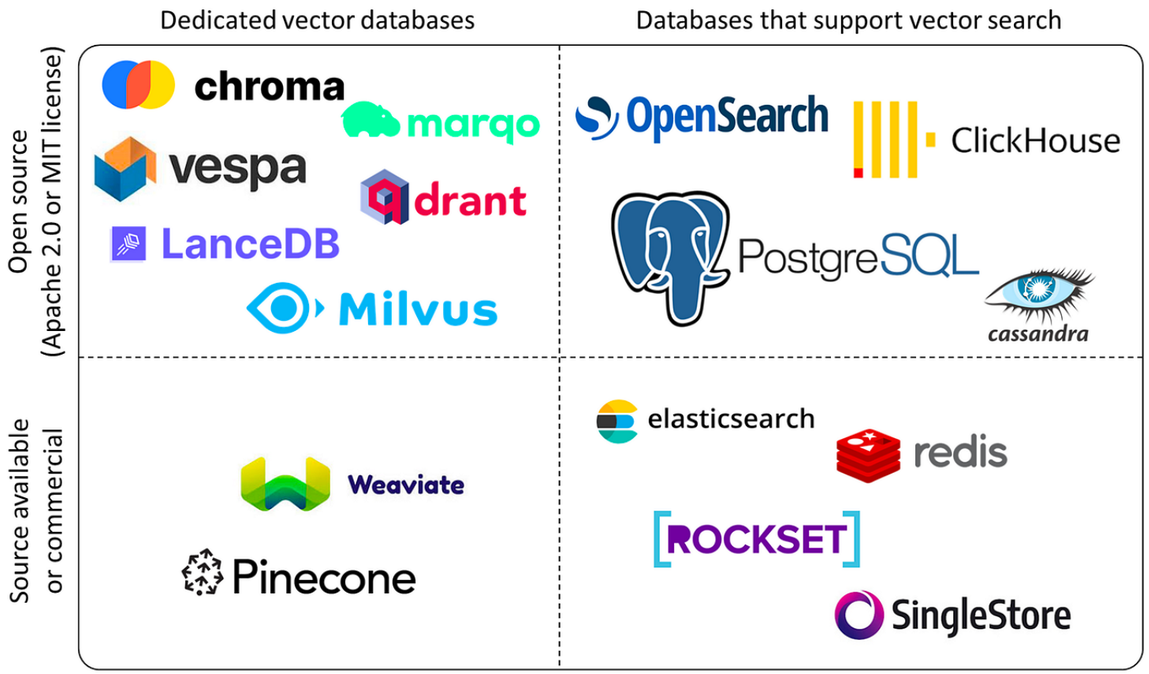


- **Focus of this video:** *storage* and *retrieval*

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!

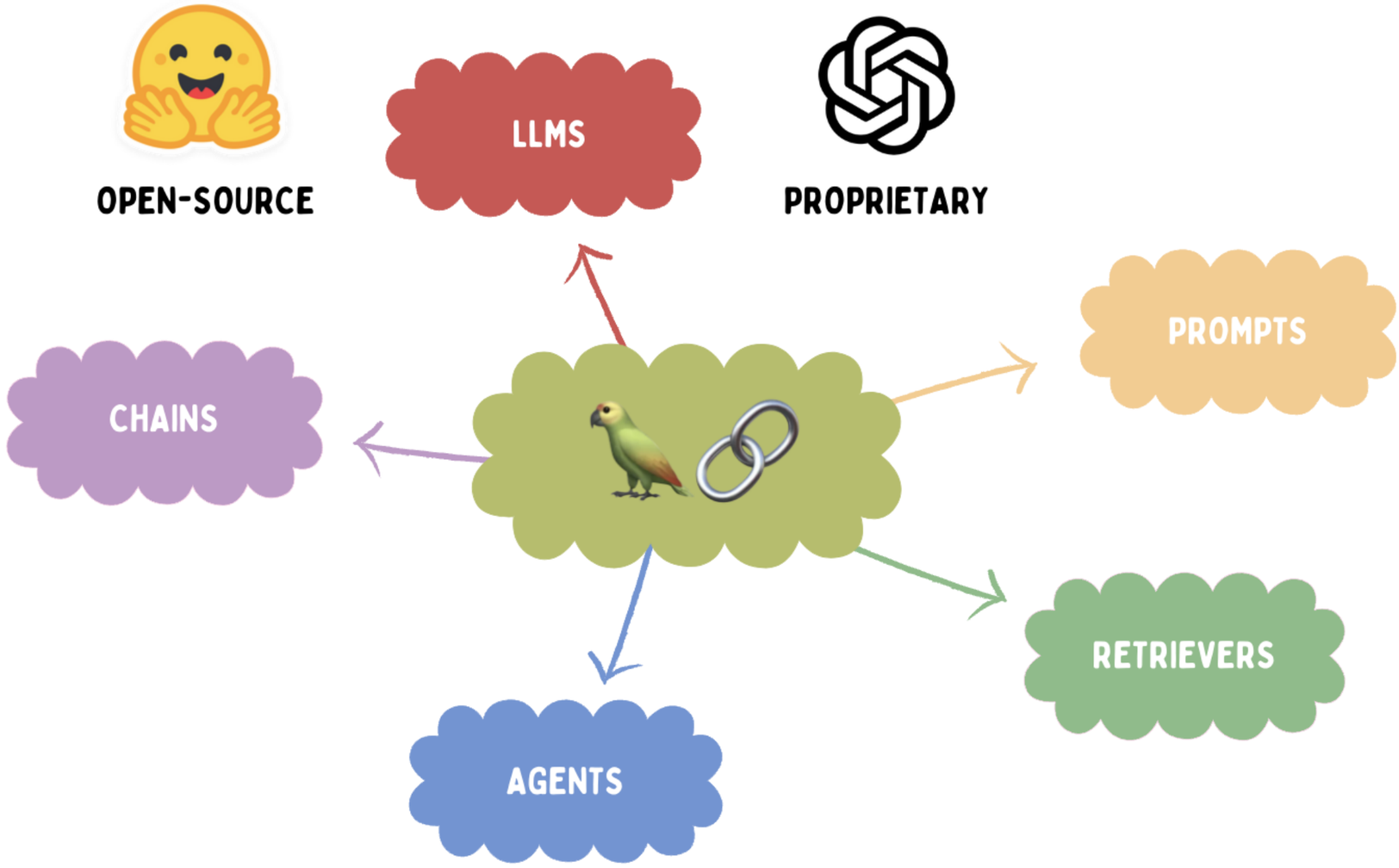
DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Wrap-up!

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

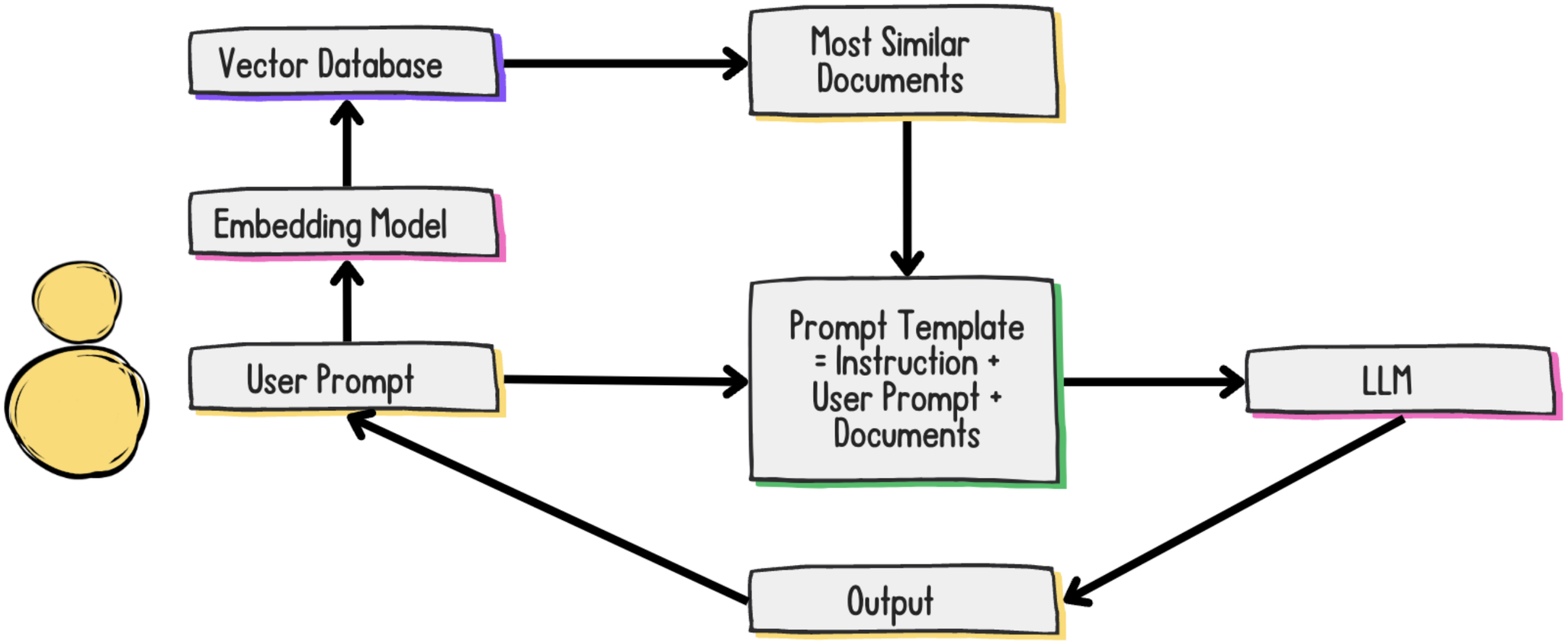
Jonathan Bennion
AI Engineer & LangChain Contributor

LangChain's core components

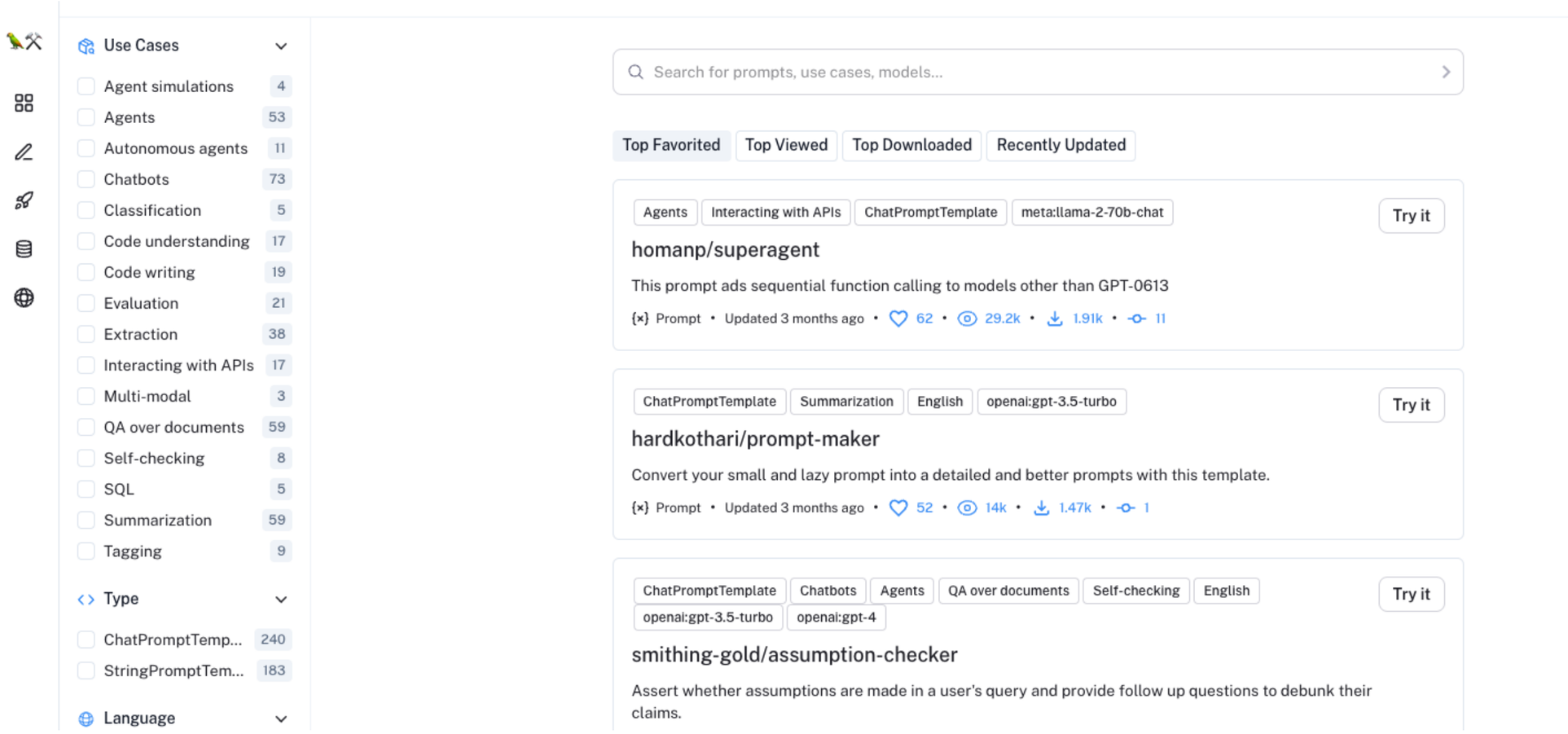


Chains and agents

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