Loading Documents for RAG with LangChain

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Meri Nova
Machine Learning Engineer

datacamp

Meet your instructor...

Meri Nova

- Founder at **Break Into Data**
- Machine Learning Engineer
- Content Creator on Linkedin and YouTube

Retrieval Augmented Generation (RAG)

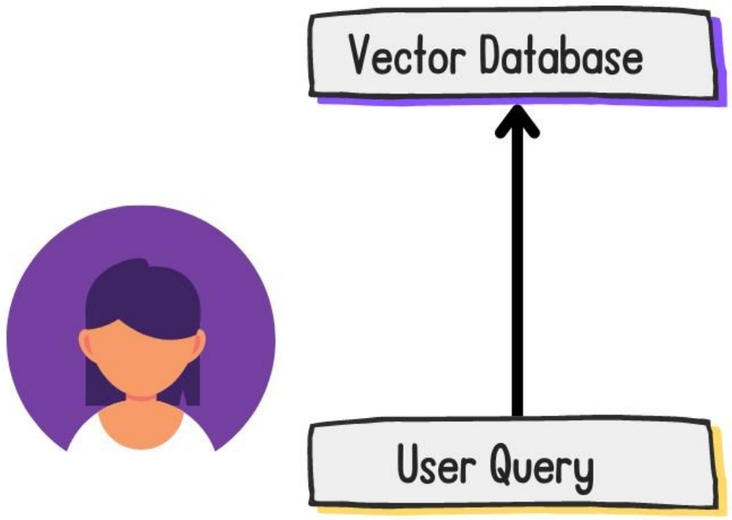
- **LLM Limitation:** knowledge constraints

→ ***RAG:*** *Integrating external data with LLMs*

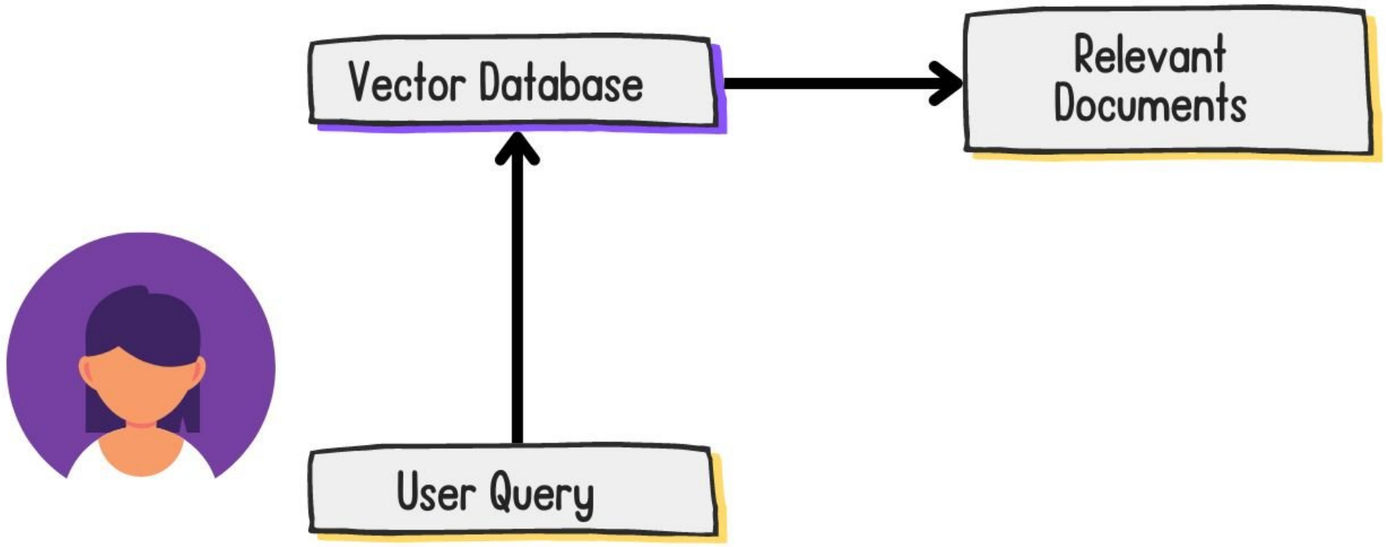
[1] Generated with DALL·E 3

The standard RAG workflow

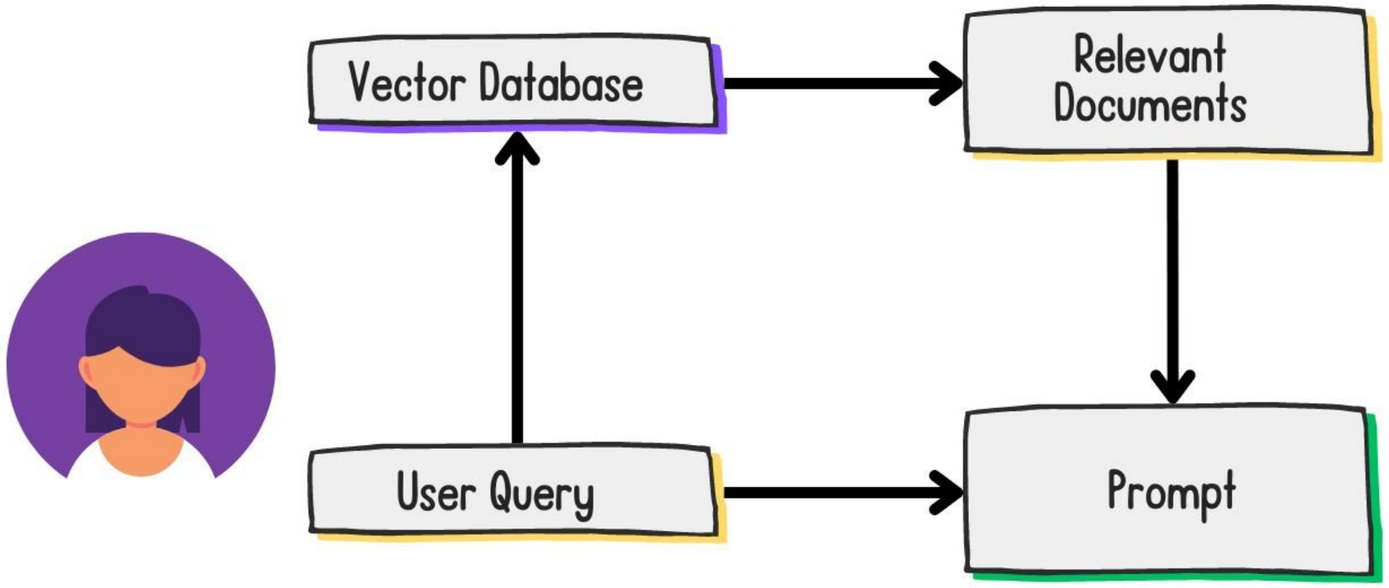
The standard RAG workflow



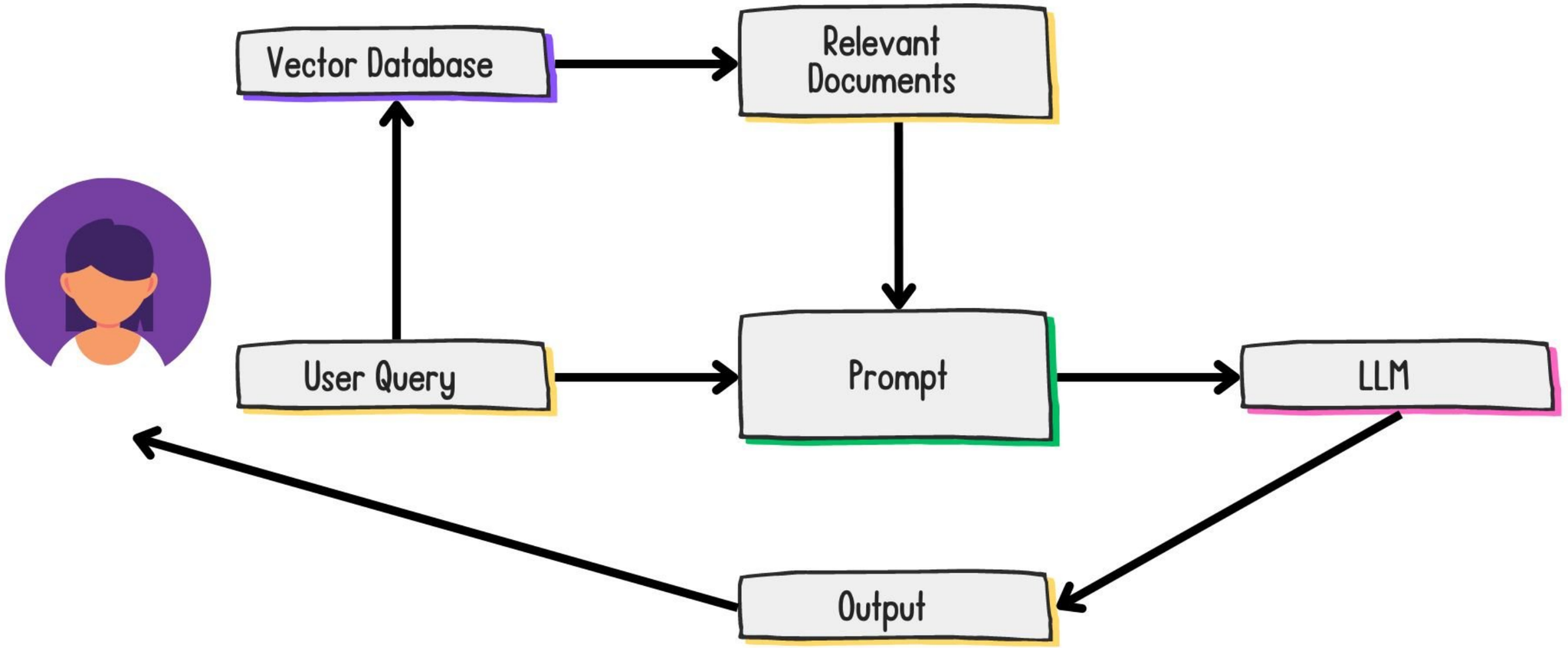
The standard RAG workflow



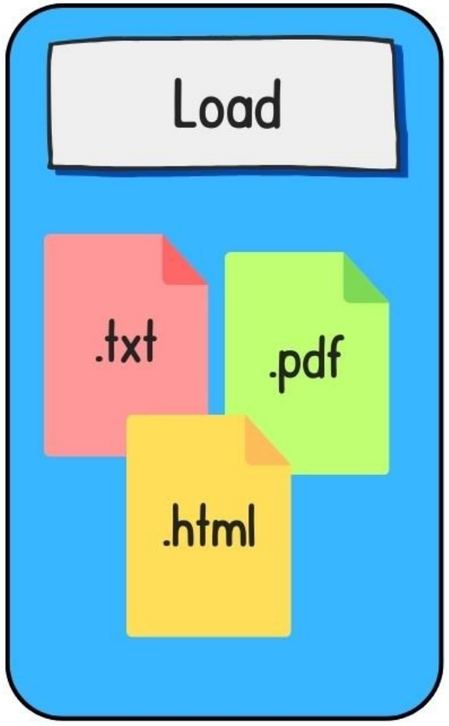
The standard RAG workflow



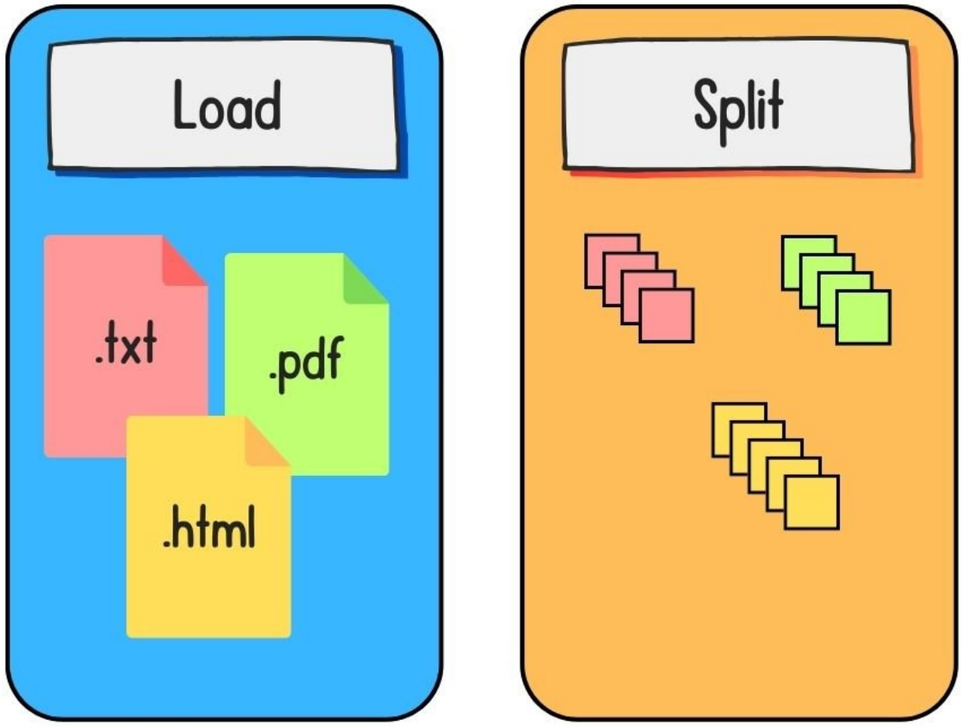
The standard RAG workflow



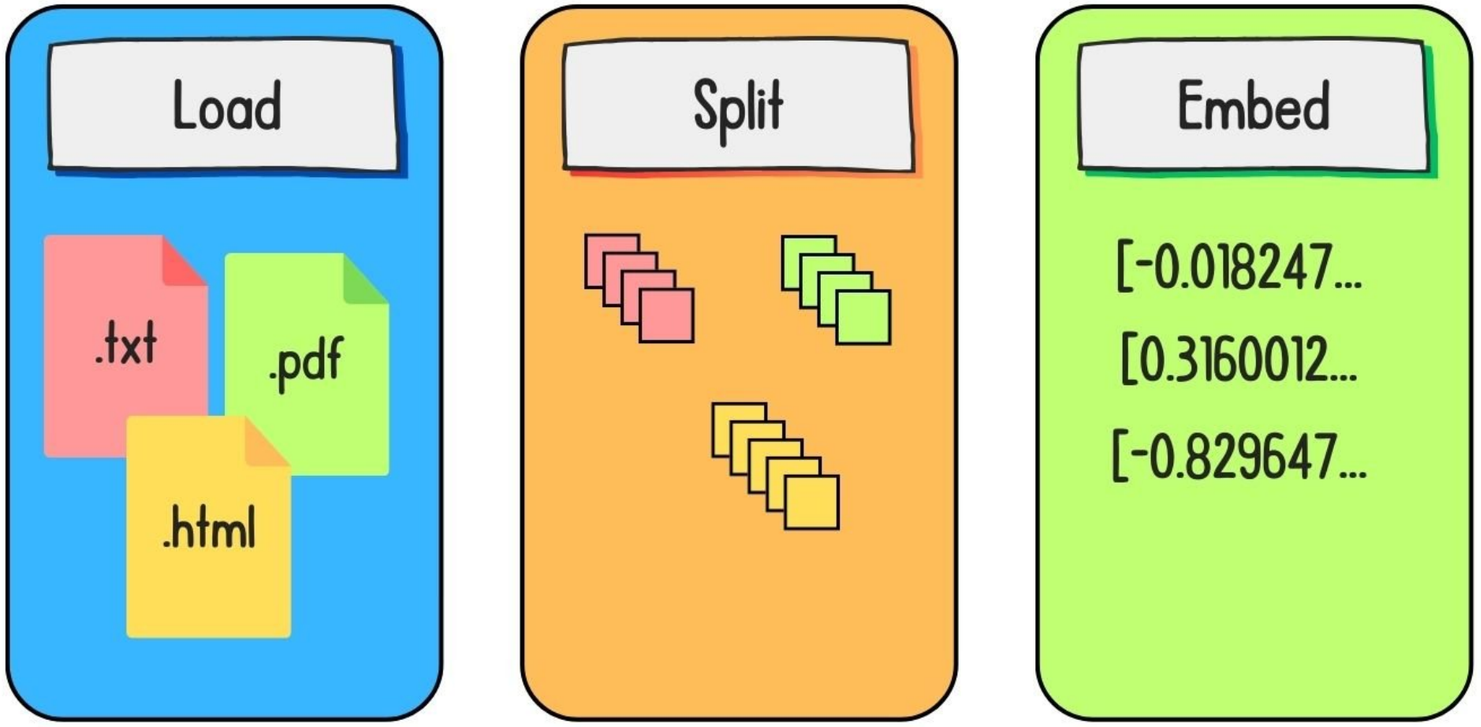
Preparing data for retrieval



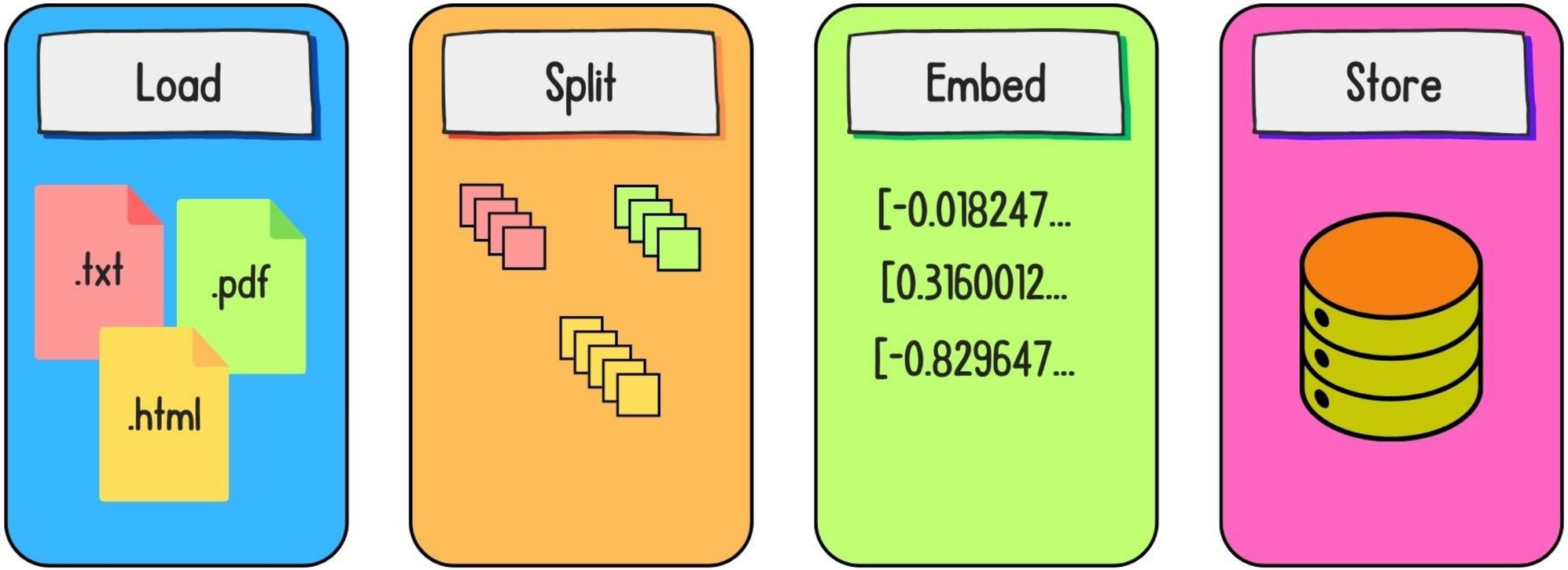
Preparing data for retrieval



Preparing data for retrieval

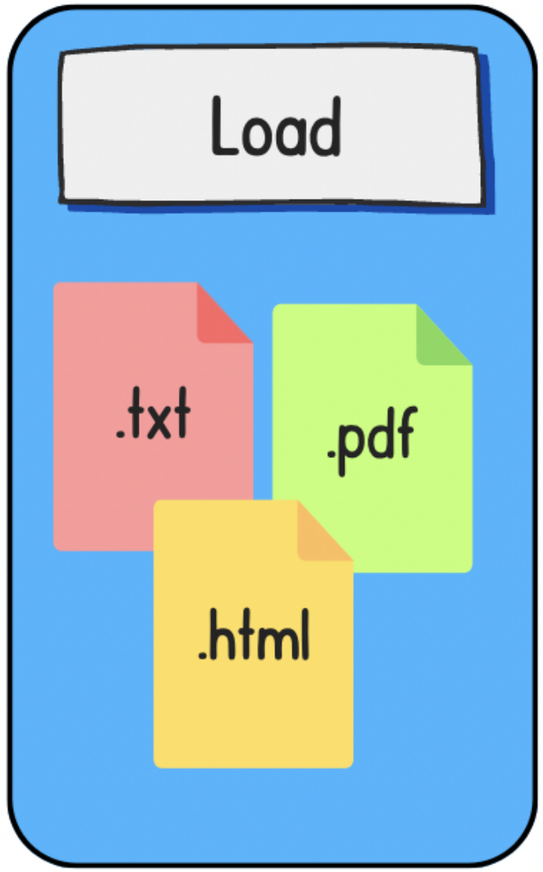


Preparing data for retrieval



Document loaders

- Integrate documents with AI systems
- Support for many common file formats
- Third party document loaders



- `CSVLoader`
- `PyPDFLoader`
- `UnstructuredHTMLLoader`

Loading CSV Files

```
from langchain_community.document_loaders.csv_loader import CSVLoader

csv_loader = CSVLoader(file_path='path/to/your/file.csv')
documents = csv_loader.load()
print(documents)
```

```
[Document(page_content='Team: Nationals\n"Payroll (millions)": 81.34\n"Wins": 98',
          metadata={'source': 'path/to/your/file.csv', 'row': 0}),
 Document(page_content='Team: Reds\n"Payroll (millions)": 82.20\n"Wins": 97',
          metadata={'source': 'path/to/your/file.csv', 'row': 1}),
 Document(page_content='Team: Yankees\n"Payroll (millions)": 197.96\n"Wins": 95',
          metadata={'source': 'path/to/your/file.csv', 'row': 2})]
```

Loading PDF Files

```
from langchain_community.document_loaders import PyPDFLoader

pdf_loader = PyPDFLoader('rag_paper.pdf')
documents = pdf_loader.load()
print(documents)
```

```
[Document(page_content='Retrieval-Augmented Generation for\nKnowledge-Intensive...',
          metadata={'source': 'Rag Paper.pdf', 'page': 0})]
```

Loading HTML Files

```
from langchain_community.document_loaders import UnstructuredHTMLLoader

html_loader = UnstructuredHTMLLoader(file_path='path/to/your/file.html')
documents = html_loader.load()
first_document = documents[0]

print("Content:", first_document.page_content)
print("Metadata:", first_document.metadata)
```

```
Content: Welcome to Our Website
Metadata: {'source': 'path/to/your/file.html', 'section': 0}
```

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

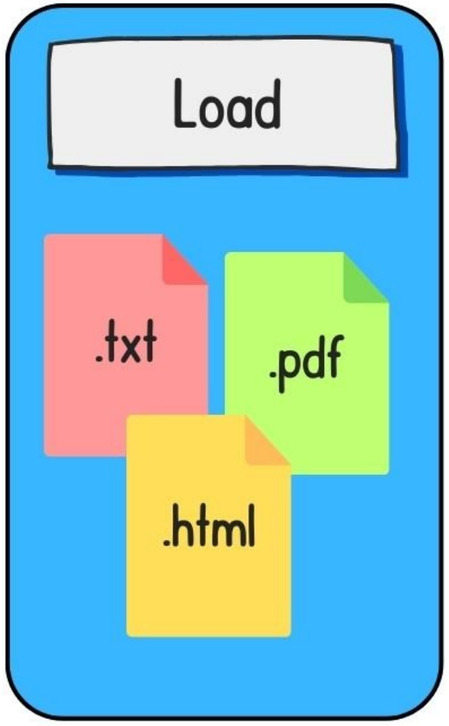
Text splitting, embeddings, and vector storage

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

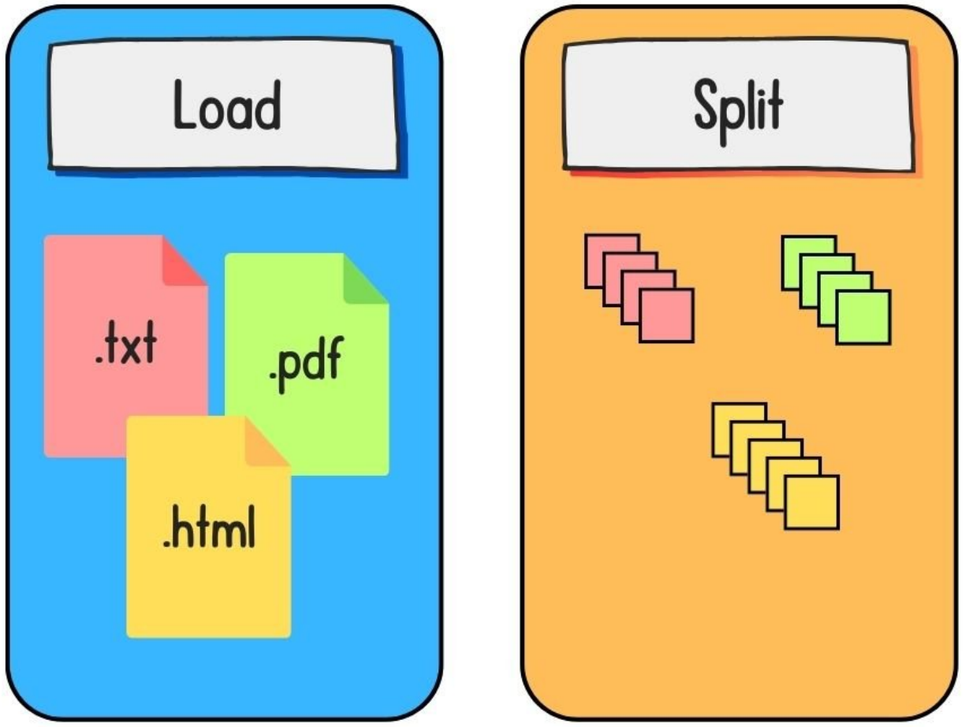
Meri Nova
Machine Learning Engineer

datacamp

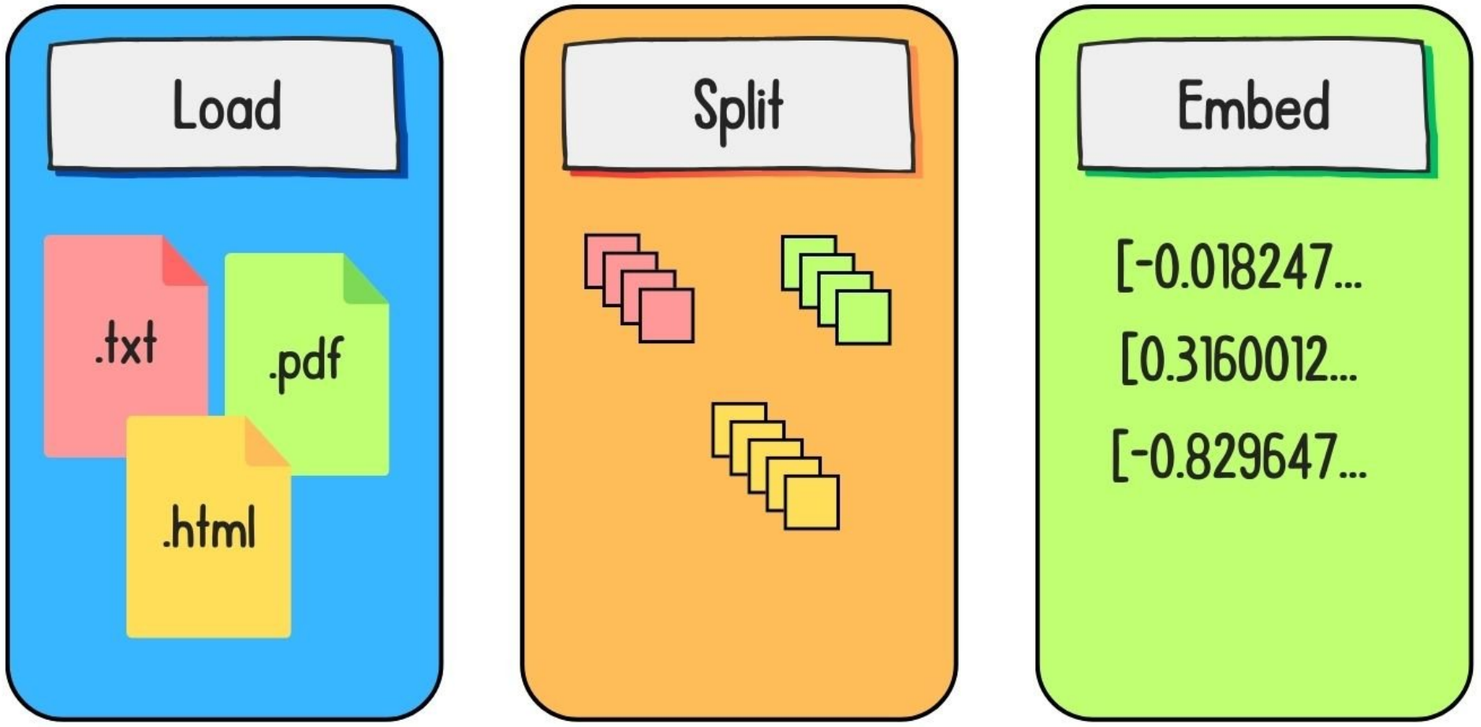
Preparing data for retrieval



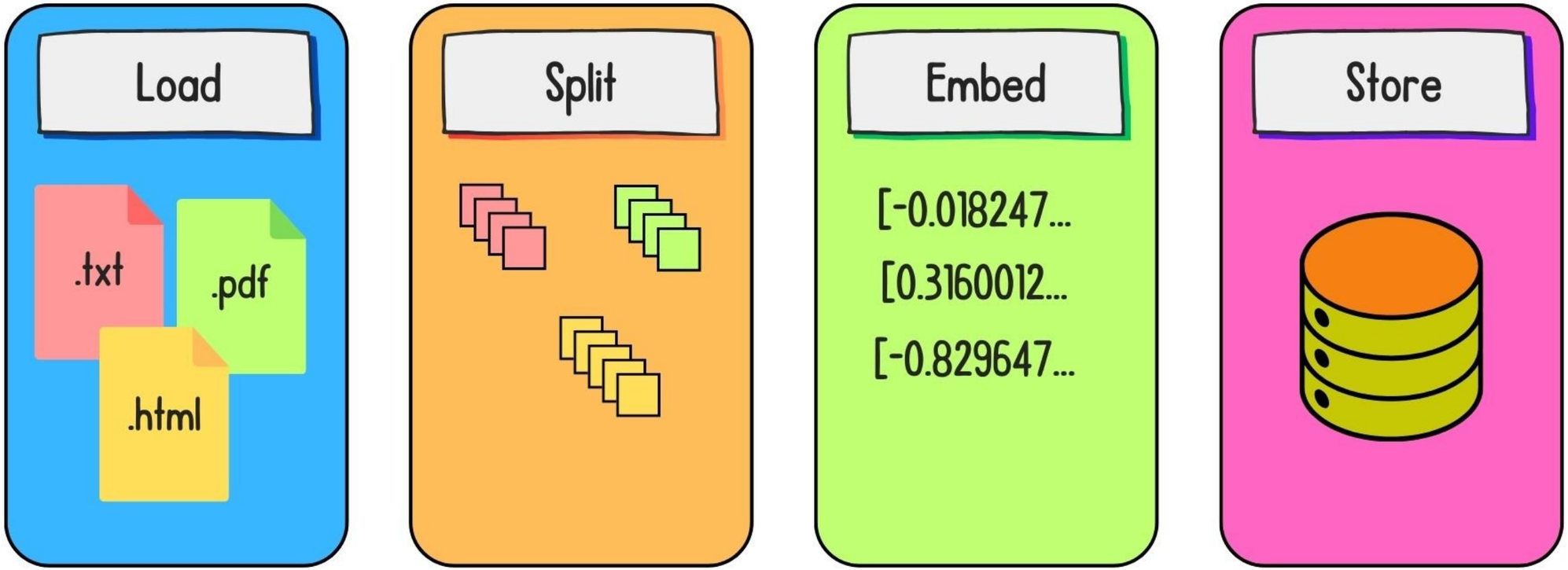
Preparing data for retrieval



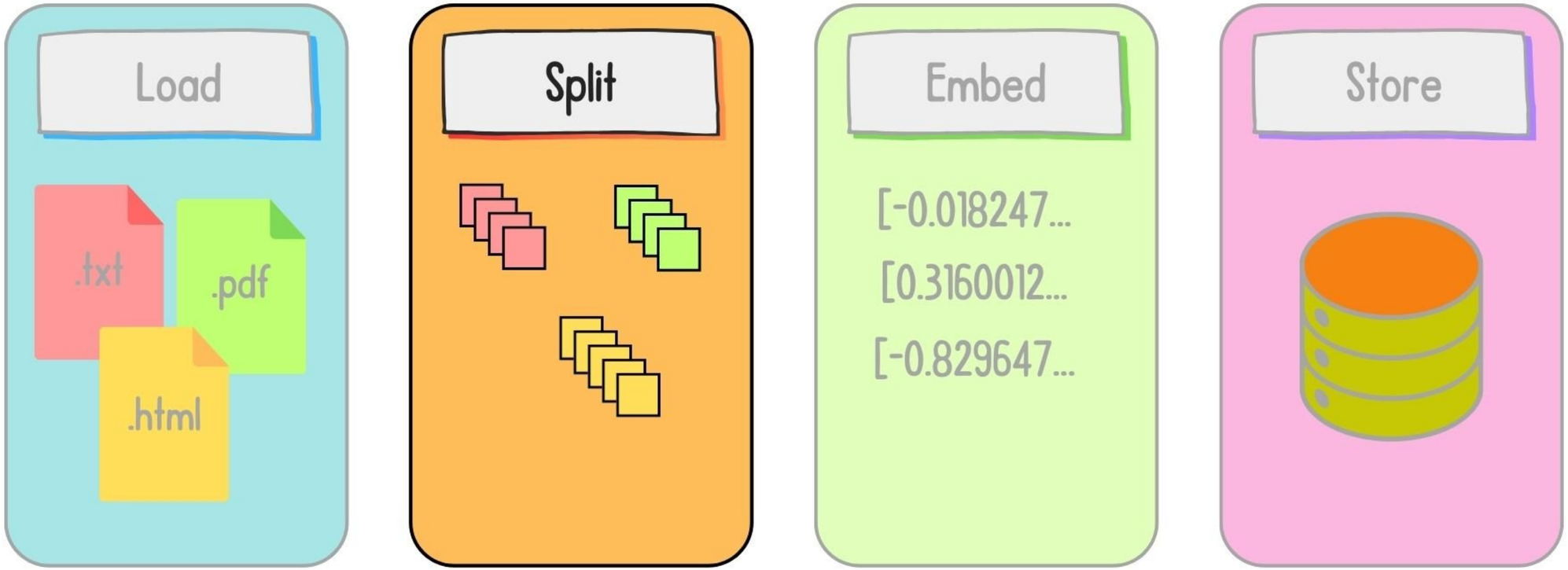
Preparing data for retrieval



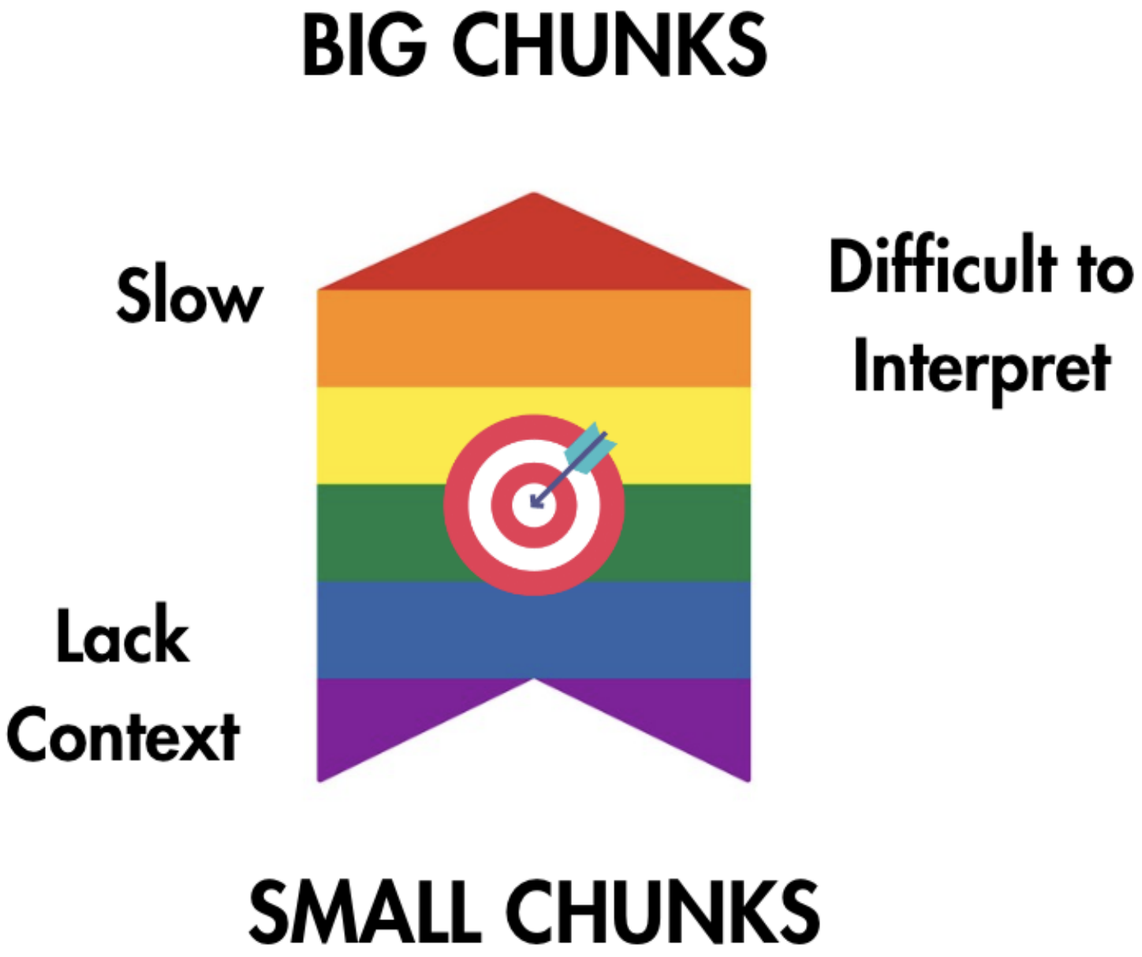
Preparing data for retrieval



Preparing data for retrieval

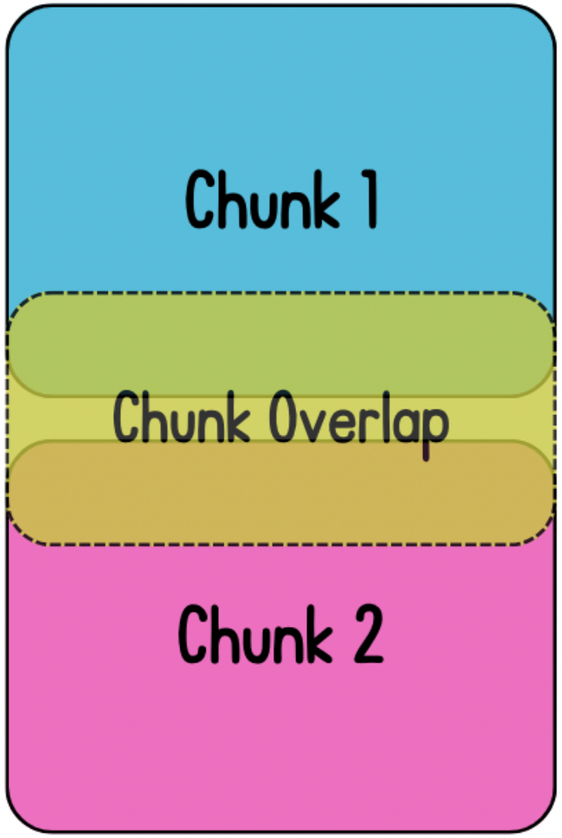


`chunk_size`



`chunk_overlap`

- Include information *beyond* the **boundary**



CharacterTextSplitter

```
from langchain_text_splitters import CharacterTextSplitter

text = """Machine learning is a fascinating field.\n\nIt involves algorithms and models that can learn from data. These models can then make predictions or decisions without being explicitly programmed to perform the task.\nThis capability is increasingly valuable in various industries, from finance to healthcare.\n\nThere are many types of machine learning, including supervised, unsupervised, and reinforcement learning.\nEach type has its own strengths and applications."""

text_splitter = CharacterTextSplitter(
    separator="\n\n",
    chunk_size=100,
    chunk_overlap=10
)
```

CharacterTextSplitter

```
chunks = text_splitter.split_text(text)
print(chunks)
print([len(chunk) for chunk in chunks])
```

```
['Machine learning is a fascinating field.',
 'It involves algorithms and models that can learn from data. These models can...',
 'There are many types of machine learning, including supervised, unsupervised...']
[40, 260, 155]
```

- Chunks may lack context
- Chunks may be larger than `chunk_size`

RecursiveCharacterTextSplitter

```
from langchain_text_splitters import RecursiveCharacterTextSplitter


splitter = RecursiveCharacterTextSplitter(
    separators=["\n\n", "\n", " ", ""],
    chunk_size=100,
    chunk_overlap=10
)
```

RecursiveCharacterTextSplitter

```
chunks = splitter.split_text(text)


print(chunks)
print([len(chunk) for chunk in chunks])
```

```
['Machine learning is a fascinating field.',
 'It involves algorithms and models that can learn from data. These models ...',
 'or decisions without being explicitly programmed to perform the task.',
 'This capability is increasingly valuable in various industries, from ...',
 'There are many types of machine learning, including supervised, ...',
 'learning.',
 'Each type has its own strengths and applications.']
[40, 98, 69, 91, 95, 9, 49]
```

Splitting documents

```
from langchain_community.document_loaders import PyPDFLoader

loader = PyPDFLoader("research_paper.pdf")
documents = loader.load()

splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)

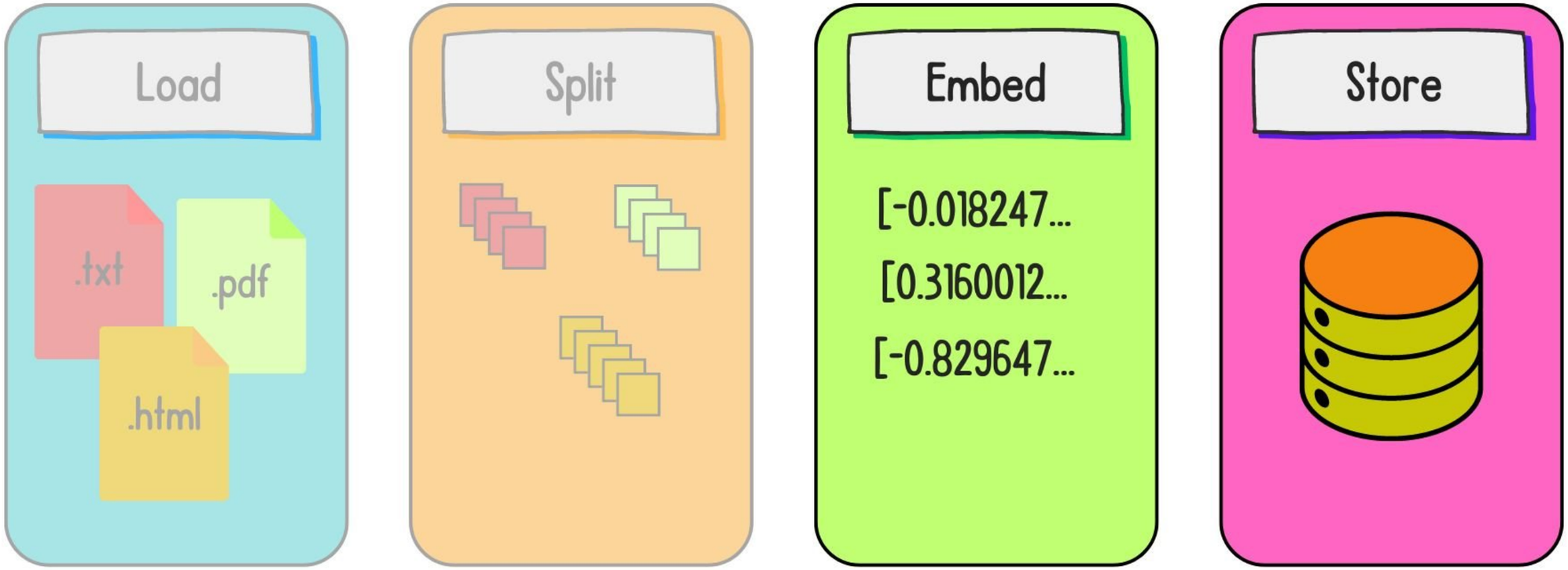
chunks = splitter.split_documents(documents)
```

Splitting documents

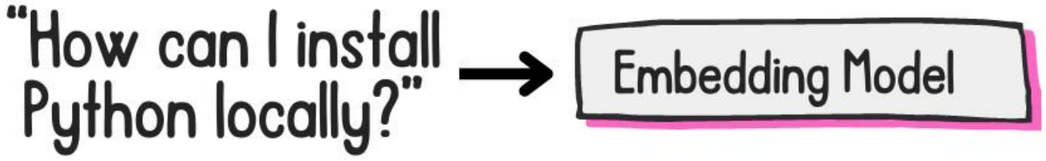
```
print(chunks)
print([len(chunk.page_content) for chunk in chunks])
```

```
[Document(metadata={'source': 'Rag Paper.pdf', 'page': 0}, page_content='...'),
 Document(metadata={'source': 'Rag Paper.pdf', 'page': 0}, page_content='...'),
 Document(metadata={'source': 'Rag Paper.pdf', 'page': 0}, page_content='...')]
[928, 946, 921,...]
```

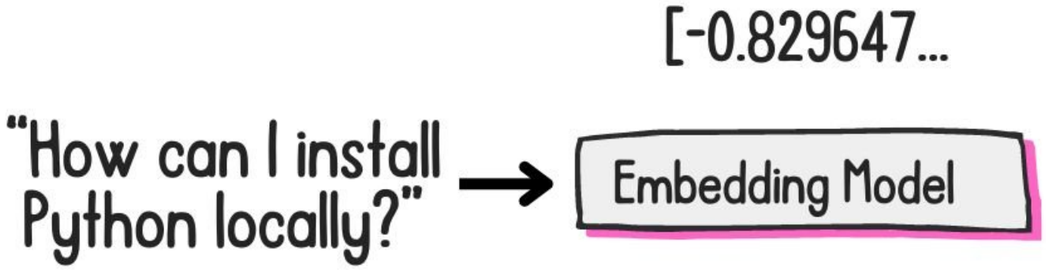
Embedding and storage



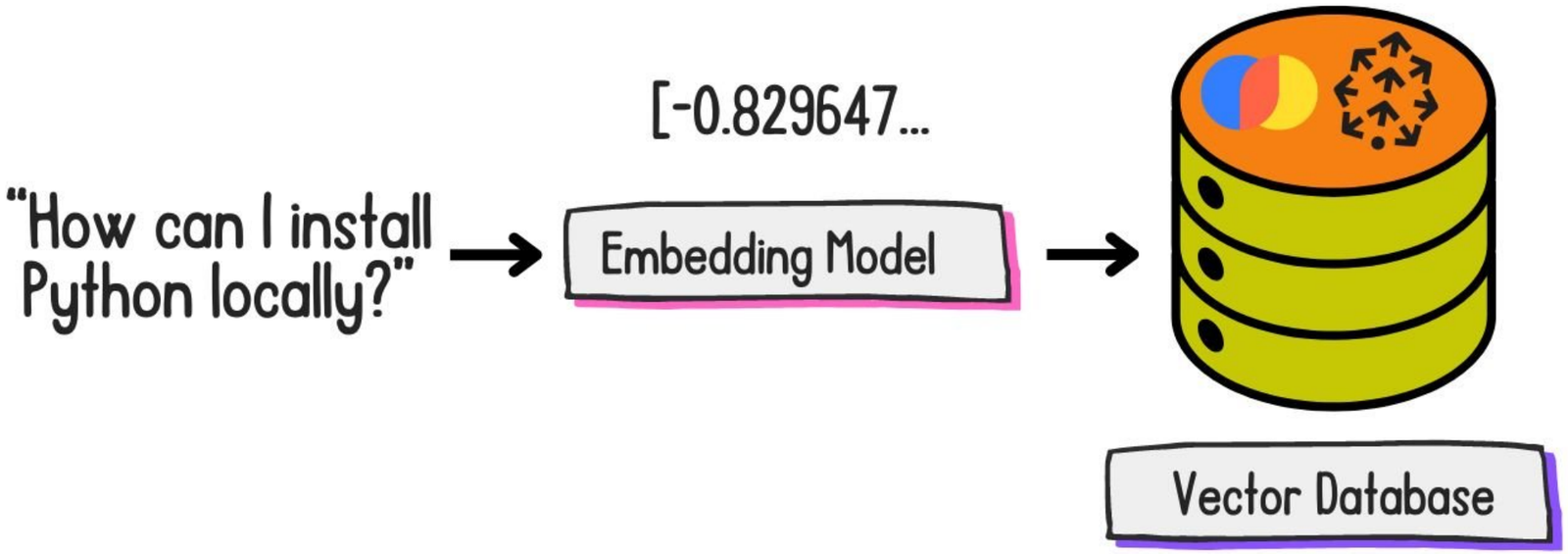
What are embeddings?



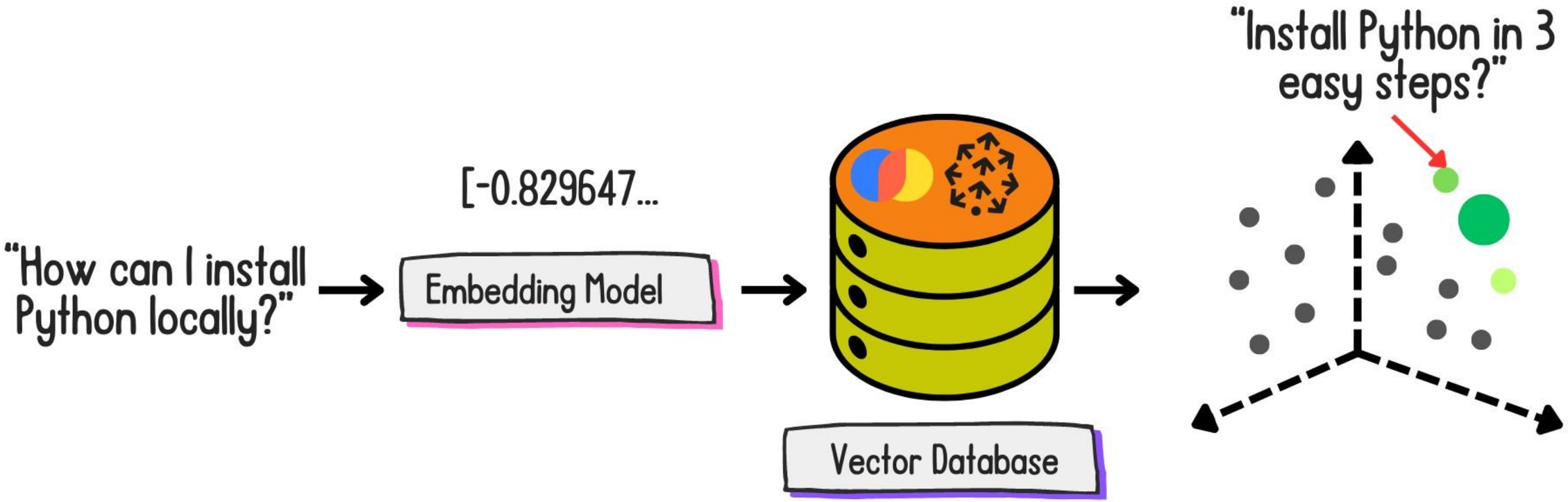
What are embeddings?



What are embeddings?



What are embeddings?



Embedding and storing the chunks

- Embed and store with: **OpenAI** and **ChromaDB**

```
from langchain_openai import OpenAIEmbeddings
from langchain_chroma import Chroma

embedding_model = OpenAIEmbeddings(
    api_key=openai_api_key,
    model="text-embedding-3-small"
)

vector_store = Chroma.from_documents(
    documents=chunks,
    embedding=embedding_model
)
```

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

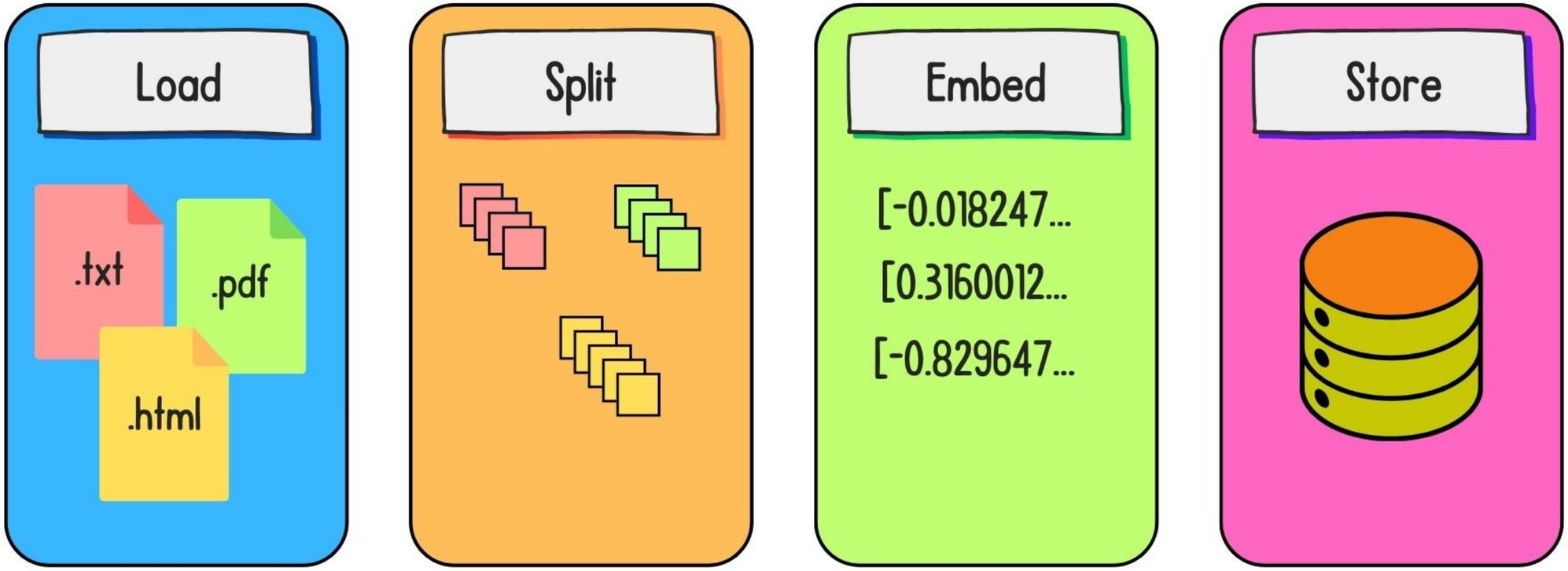
Building an LCEL retrieval chain

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

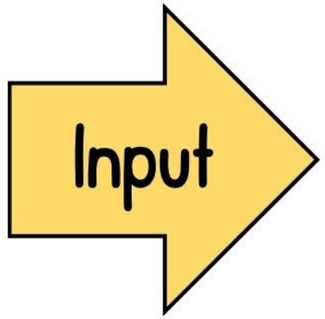
Meri Nova
Machine Learning Engineer

datacamp

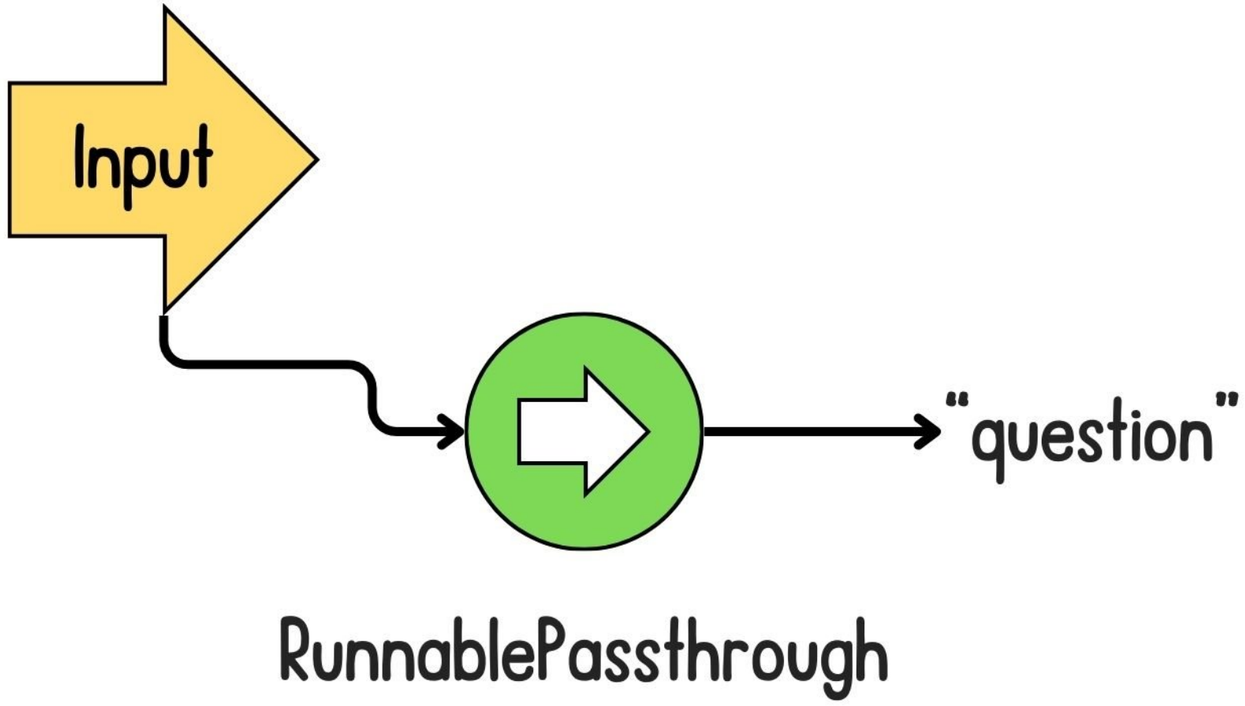
Preparing data for retrieval



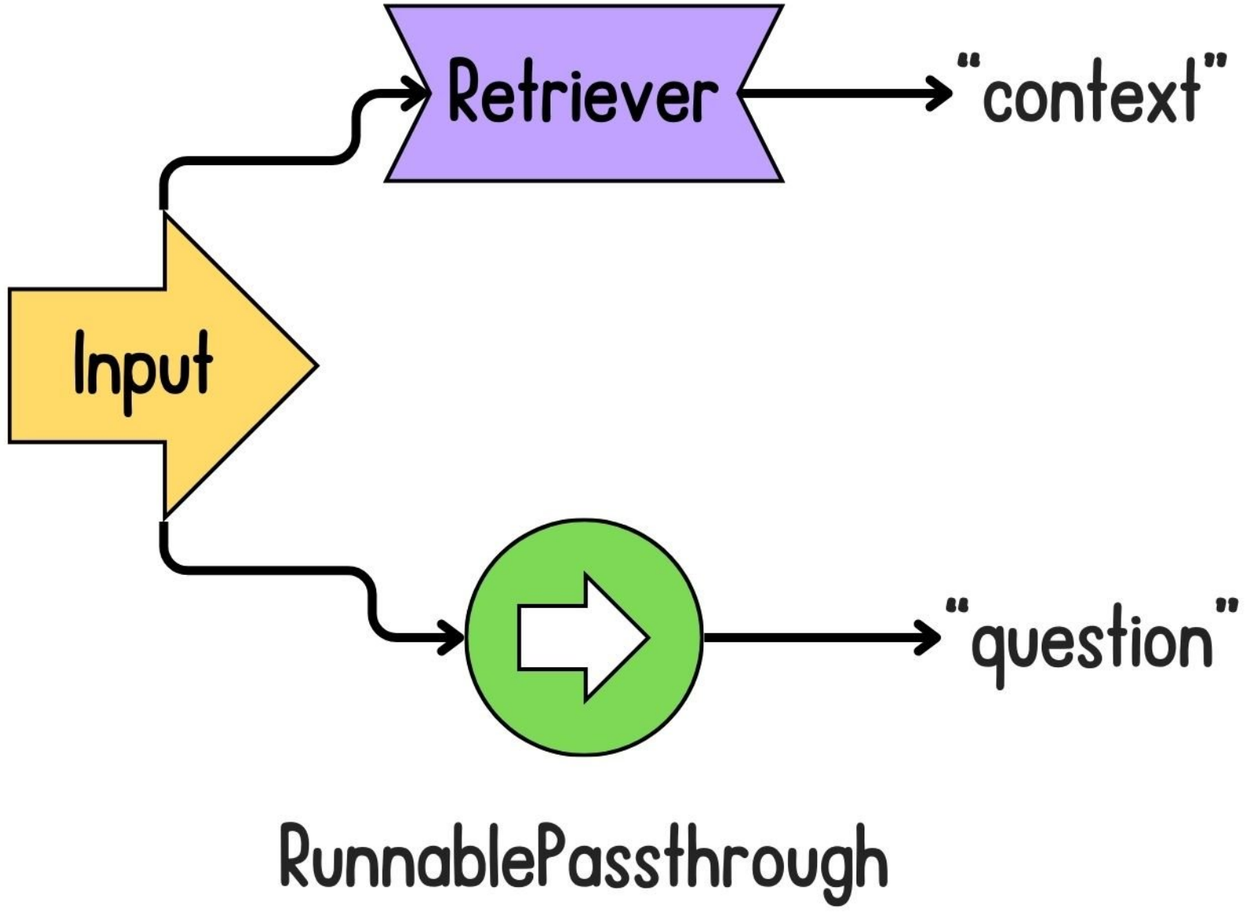
Introduction to LCEL for RAG



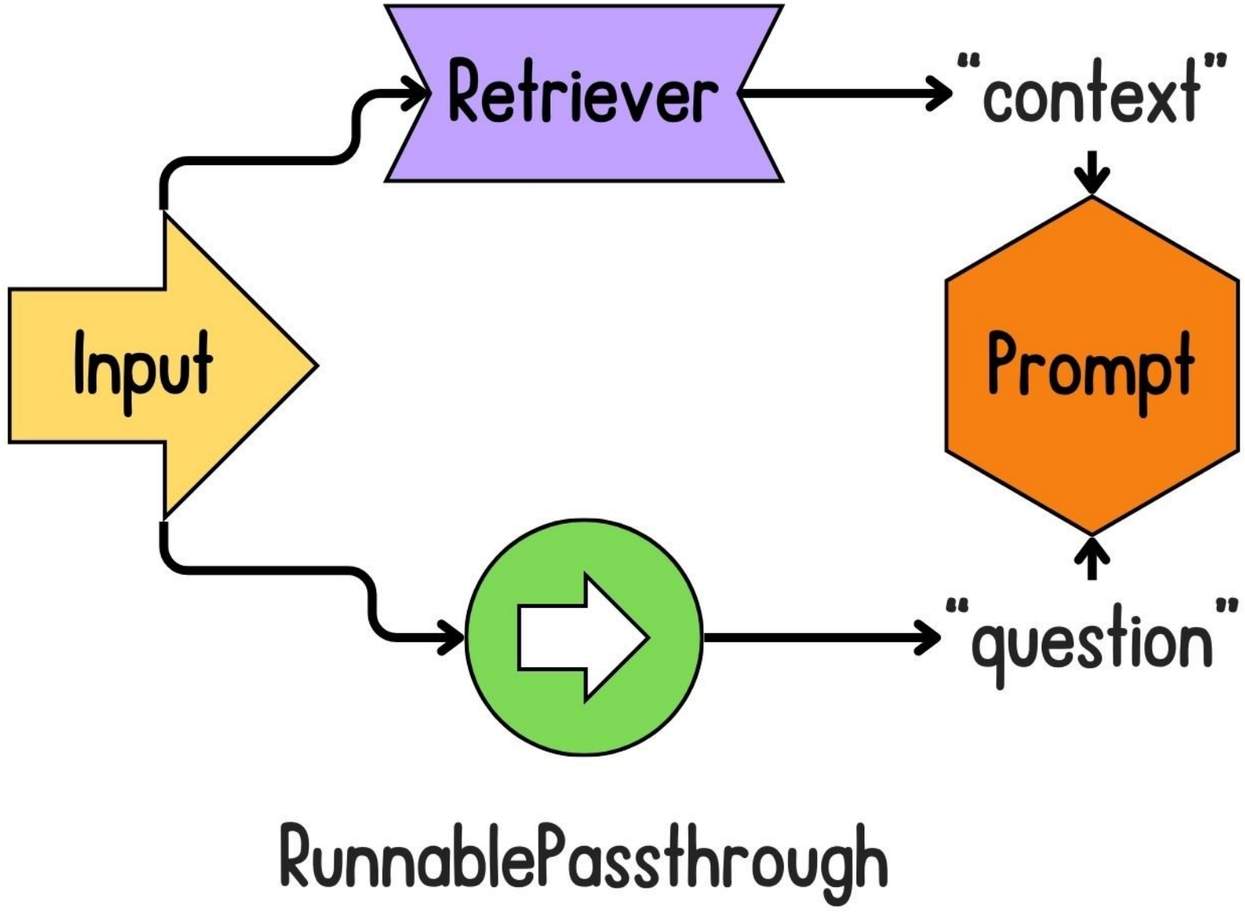
Introduction to LCEL for RAG



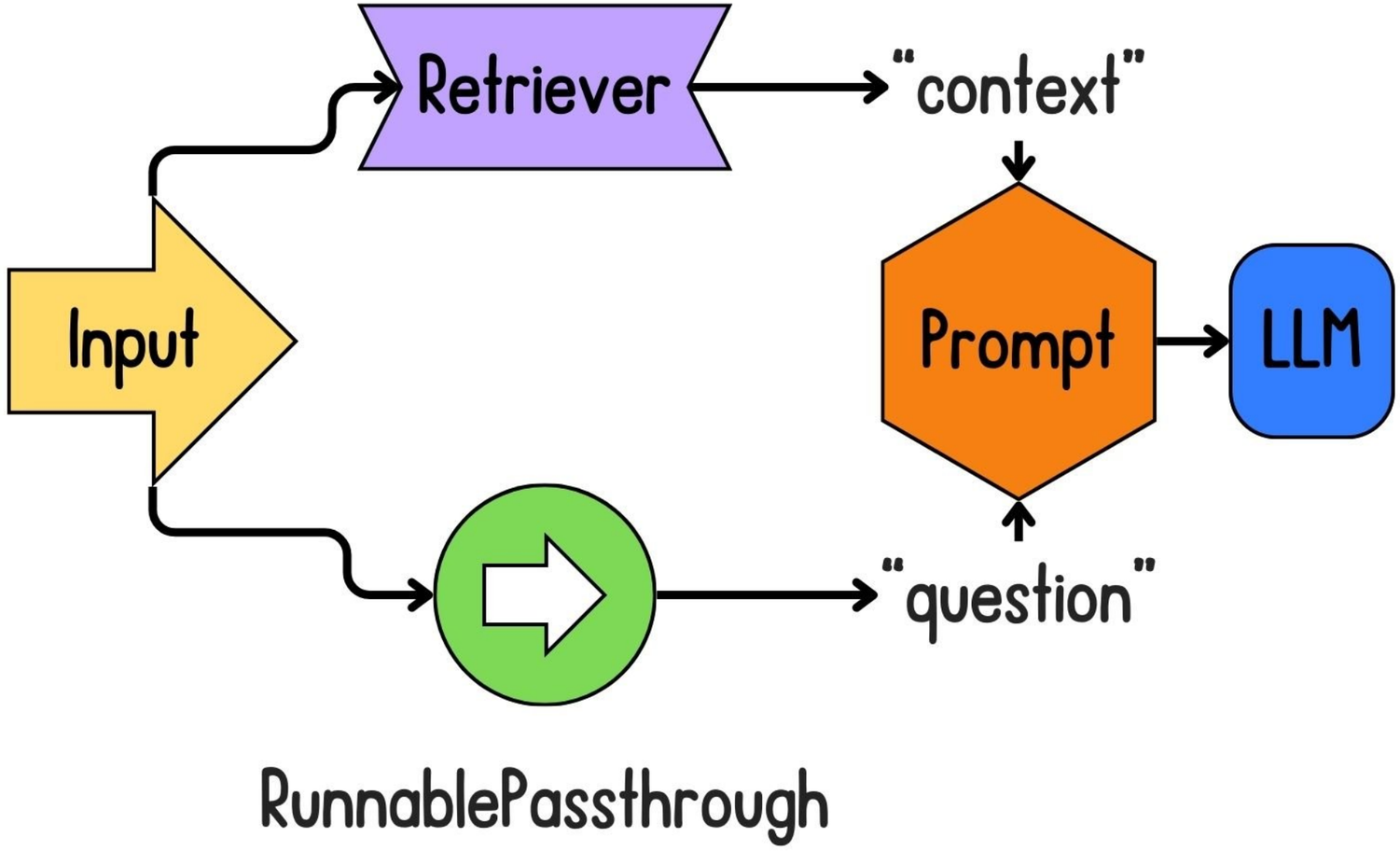
Introduction to LCEL for RAG



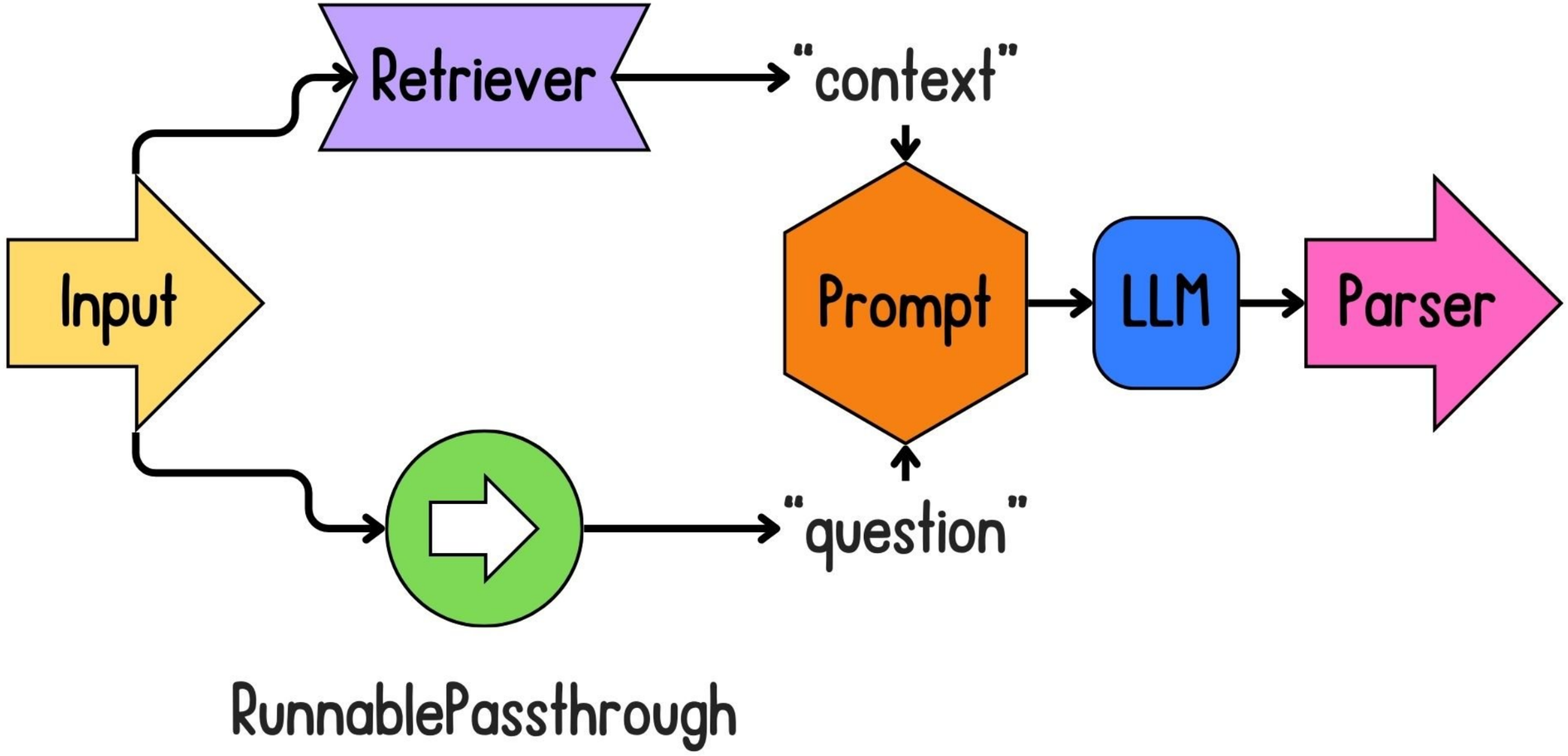
Introduction to LCEL for RAG



Introduction to LCEL for RAG



Introduction to LCEL for RAG



Instantiating a retriever

```
vector_store = Chroma.from_documents(
    documents=chunks,
    embedding=embedding_model
)

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

Creating a prompt template

```
from langchain_core.prompts import ChatPromptTemplate


prompt = ChatPromptTemplate.from_template("""
Use the following pieces of context to answer the question at the end.
If you don't know the answer, say that you don't know.
Context: {context}
Question: {question}
""")
```

```
llm = ChatOpenAI(model="gpt-4o-mini", api_key="...", temperature=0)
```

Building an LCEL retrieval chain

```
from langchain_core.runnables import RunnablePassthrough
from langchain_core.output_parsers import StrOutputParser

chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)
```

Invoking the retrieval chain

```
result = chain.invoke({"question": "What are the key findings or results presented in the paper?"})
print(result)
```

```
- Top Performance: RAG models set new records on open-domain question answering tasks...
- Better Generation: RAG models produce more specific, diverse, and factual language...
- Dynamic Knowledge Use: The non-parametric memory allows RAG models to access and ...
```

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Loading and splitting code files

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Meri Nova
Machine Learning Engineer

datacamp

More document loaders...

```
# 🦜🔗 LangChain

⚡ Build context-aware reasoning applications ⚡

[![Release Notes](https://img.shields.io/github/release/langchain-ai/langchain?style=flat-square)](https://github.com/langchain-ai/langchain/relea
[![CI](https://github.com/langchain-ai/langchain/actions/workflows/check_diffs.yml/badge.svg)](https://github.com/langchain-ai/langchain/actions/v
[![PyPI - License](https://img.shields.io/pypi/l/langchain-core?style=flat-square)](https://opensource.org/licenses/MIT)
[![PyPI - Downloads](https://img.shields.io/pypi/dm/langchain-core?style=flat-square)](https://pypistats.org/packages/langchain-core)
[![GitHub star chart](https://img.shields.io/github/stars/langchain-ai/langchain?style=flat-square)](https://star-history.com/#langchain-ai/langcl
[![Open Issues](https://img.shields.io/github/issues-raw/langchain-ai/langchain?style=flat-square)](https://github.com/langchain-ai/langchain/issu
[![Open in Dev Containers](https://img.shields.io/static/v1?label=Dev%20Containers&message=Open&color=blue&logo=visualstudiocode&style=flat-square
remote.remote-containers/cloneInVolume?url=https://github.com/langchain-ai/langchain)
[![Open in GitHub Codespaces](https://github.com/codespaces/badge.svg)](https://codespaces.new/langchain-ai/langchain)
[![Twitter](https://img.shields.io/twitter/url/https/twitter.com/langchainai.svg?style=social&label=Follow%20%40LangChainAI)](https://twitter.com/

Looking for the JS/TS library? Check out [LangChain.js](https://github.com/langchain-ai/langchainjs).

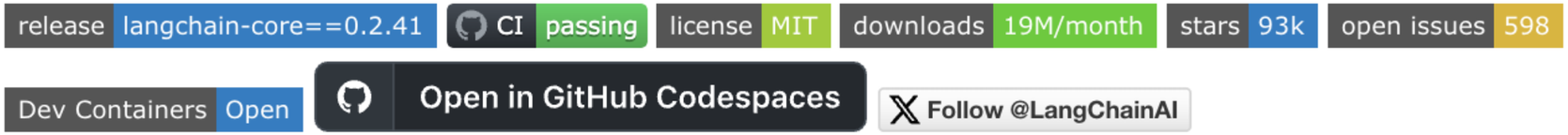
To help you ship LangChain apps to production faster, check out [LangSmith](https://smith.langchain.com).
[LangSmith](https://smith.langchain.com) is a unified developer platform for building, testing, and monitoring LLM applications.
Fill out [this form](https://www.langchain.com/contact-sales) to speak with our sales team.

## Quick Install

With pip:
```bash
pip install langchain
```
```

🦜🔗 LangChain

⚡ Build context-aware reasoning applications ⚡



Looking for the JS/TS library? Check out LangChain.js.

To help you ship LangChain apps to production faster, check out LangSmith. LangSmith is a unified developer platform for building, testing, and monitoring LLM applications. Fill out this form to speak with our sales team.

Quick Install

With pip:

```
pip install langchain
```

Loading Markdown files (.md)

```
from langchain_community.document_loaders import UnstructuredMarkdownLoader

loader = UnstructuredMarkdownLoader("README.md")
markdown_content = loader.load()
print(markdown_content[0])
```

```
Document(page_content='# Discord Text Classification ![Python Version](https...'
         metadata={'source': 'README.md'})
```

Loading Python files (.py)

```
from abc import ABC, abstractmethod

class LLM(ABC):
  @abstractmethod
  def complete_sentence(self, prompt):
    pass

...
```

- Integrated into RAG applications for writing or fixing code, creating docs, etc.
- Imports, classes, functions, etc.

```
from langchain_community.document_loaders \
    import PythonLoader

loader = PythonLoader('chatbot.py')

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

```
Document(page_content='from abc import ABC, ..

class LLM(ABC):
  @abstractmethod
...',
metadata={'source': 'chatbot.py'})
```

Splitting code files

```
python_splitter = RecursiveCharacterTextSplitter(
    chunk_size=150, chunk_overlap=10
)

chunks = python_splitter.split_documents(python_data)
for i, chunk in enumerate(chunks[:3]):
    print(f"Chunk {i+1}:\n{chunk.page_content}\n")
```

```
Chunk 1:
from abc import ABC, abstractmethod

class LLM(ABC):
  @abstractmethod
  def complete_sentence(self, prompt):
    pass

Chunk 2:
class OpenAI(LLM):
  def complete_sentence(self, prompt):
    return prompt + " ... OpenAI end of sentence."

class Anthropic(LLM):

Chunk 3:
def complete_sentence(self, prompt):
    return prompt + " ... Anthropic end of sentence."
```

Splitting by language

- `separators`
  - `["\n\n", "\n", " ", ""]`
  - `["\nclass ", "\ndef ", "\n\tdef ", "\n\n", " ", ""]`

```
from langchain_text_splitters import RecursiveCharacterTextSplitter, Language

python_splitter = RecursiveCharacterTextSplitter.from_language(
    language=Language.PYTHON, chunk_size=150, chunk_overlap=10
)

chunks = python_splitter.split_documents(data)

for i, chunk in enumerate(chunks[:3]):
    print(f"Chunk {i+1}:\n{chunk.page_content}\n")
```

```
Chunk 1:
from abc import ABC, abstractmethod

Chunk 2:
class LLM(ABC):
  @abstractmethod
  def complete_sentence(self, prompt):
    pass

Chunk 3:
class OpenAI(LLM):
  def complete_sentence(self, prompt):
```

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Advanced splitting methods

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Meri Nova
Machine Learning Engineer

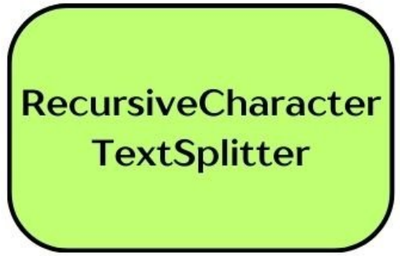
datacamp

Limitations of our current splitting strategies

1. Splits are naive (not context-aware) → `SemanticChunker`
   - Ignores context of surrounding text
2. Splits are made using characters vs. **tokens** → `TokenTextSplitter`
   - Tokens are processed by models
   - Risk exceeding the **context window**

Splitting on tokens

How is text split into tokens?



['How', 'is', 'text', 'spli', 'lit', 'into', 'toke', 'kens?']

chunk_size=5
chunk_overlap=2

Splitting on tokens

How is text split into tokens?

RecursiveCharacterTextSplitter

['How', 'is', 'text', 'spli', 'lit', 'into', 'toke', 'kens?']

chunk_size=5
chunk_overlap=2

How is text split into tokens?

TokenTextSplitter

['How is text split into', 'split into tokens?']

Splitting on tokens

How is text split into tokens?

RecursiveCharacter TextSplitter

['How', 'is', 'text', 'spli', 'lit', 'into', 'toke', 'kens?']

chunk_size=5
chunk_overlap=2

How is text split into tokens?

TokenTextSplitter

1 2 3 4 5
['How is text split into',
'split into tokens?']
1 2 3 4

Splitting on tokens

```
import tiktoken
from langchain_text_splitters import TokenTextSplitter
example_string = "Mary had a little lamb, it's fleece was white as snow."

encoding = tiktoken.encoding_for_model('gpt-4o-mini')
splitter = TokenTextSplitter(encoding_name=encoding.name,
                             chunk_size=10,
                             chunk_overlap=2)

chunks = splitter.split_text(example_string)

for i, chunk in enumerate(chunks):
    print(f"Chunk {i+1}:\n{chunk}\n")
```

Splitting on tokens

```
Chunk 1:
Mary had a little lamb, it's fleece

Chunk 2:
 fleece was white as snow.
```

Splitting on tokens

```
for i, chunk in enumerate(chunks):
    print(f"Chunk {i+1}:\nNo. tokens: {len(encoding.encode(chunk))}\n{chunk}\n")
```

```
Chunk 1:
No. tokens: 10
Mary had a little lamb, it's fleece was

Chunk 2:
No. tokens: 6
 fleece was white as snow.
```

Semantic splitting

RAG applications has numerous use cases, but are most frequently deployed in chatbots. Domestic dogs are descendants from an extinct population of Pleistocene wolves over 14,000 years ago.

Semantic splitting

RAG applications has numerous use cases,
but are most frequently deployed in chatbots.
Domestic dogs are descendants from an
extinct population of Pleistocene wolves over
14,000 years ago.

Semantic splitting

RAG applications has numerous use cases, but are most frequently deployed in chatbots.

Domestic dogs are descendants from an extinct population of Pleistocene wolves over 14,000 years ago.

Semantic splitting

```
from langchain_openai import OpenAIEmbeddings
from langchain_experimental.text_splitter import SemanticChunker

embeddings = OpenAIEmbeddings(api_key="...", model='text-embedding-3-small')

semantic_splitter = SemanticChunker(
    embeddings=embeddings,
    breakpoint_threshold_type="gradient",
    breakpoint_threshold_amount=0.8
)
```

[1] https://api.python.langchain.com/en/latest/text_splitter/langchain_experimental.text_splitter.SemanticChunker.html

Semantic splitting

```
chunks = semantic_splitter.split_documents(data)
print(chunks[0])
```

```
page_content='Retrieval-Augmented Generation for\nKnowledge-Intensive NLP Tasks\ Patrick Lewis,
Ethan Perez,\nAleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich
Küttler,\nMike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, Douwe Kiela\nFacebook AI
Research; University College London;New York University;\nplewis@fb.com\nAbstract\nLarge
pre-trained language models have been shown to store factual knowledge\nin their parameters,
and achieve state-of-the-art results when ?ne-tuned on down-\nstream NLP tasks. However, their
ability to access and precisely manipulate knowl-\nedge is still limited, and hence on
knowledge-intensive tasks, their performance\nlags behind task-specific architectures.'
metadata={'source': 'rag_paper.pdf', 'page': 0}
```

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

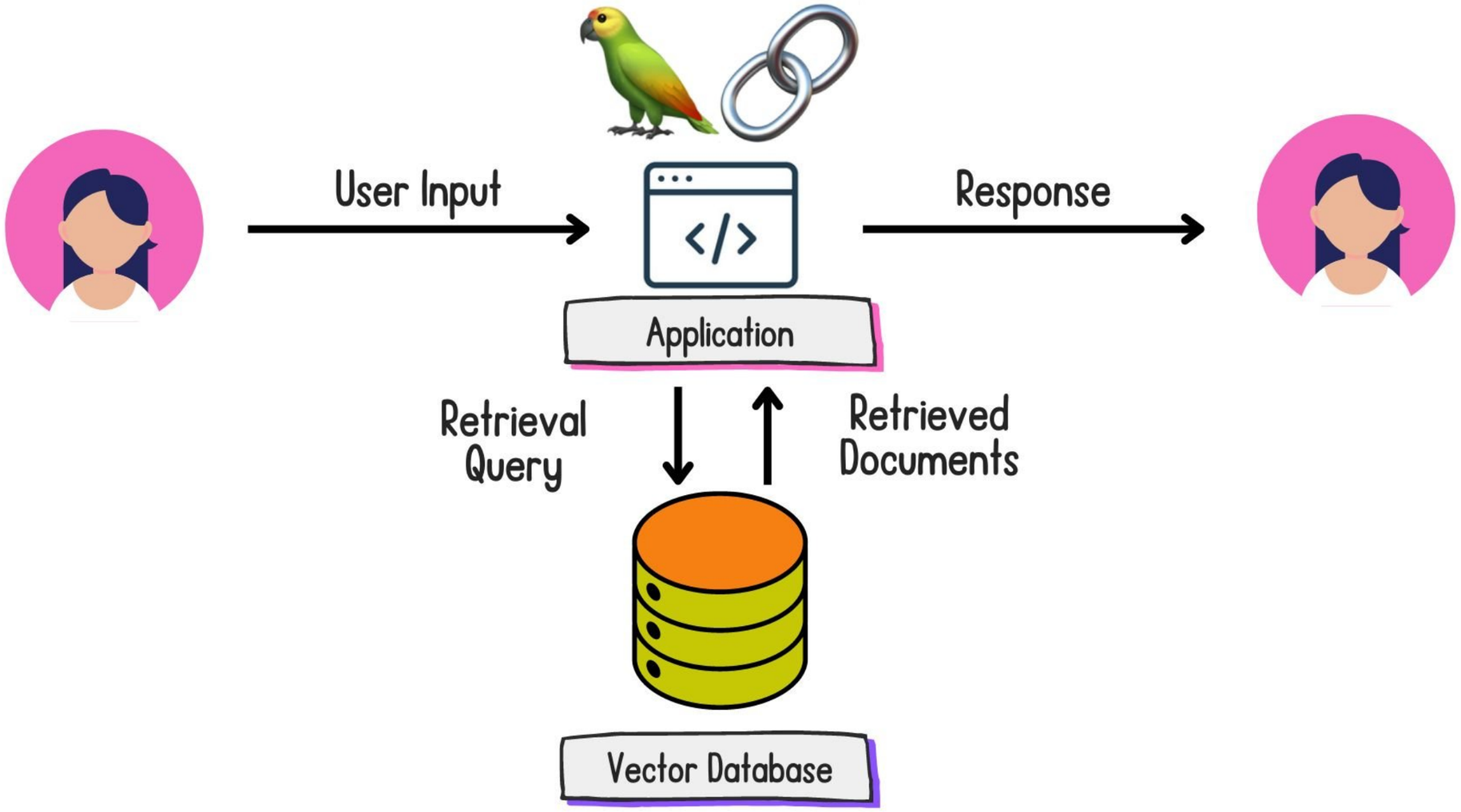
Optimizing document retrieval

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Meri Nova
Machine Learning Engineer

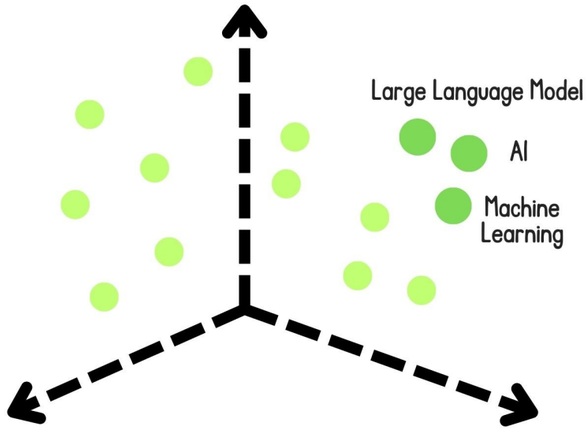
datacamp

Putting the R in RAG...



Dense

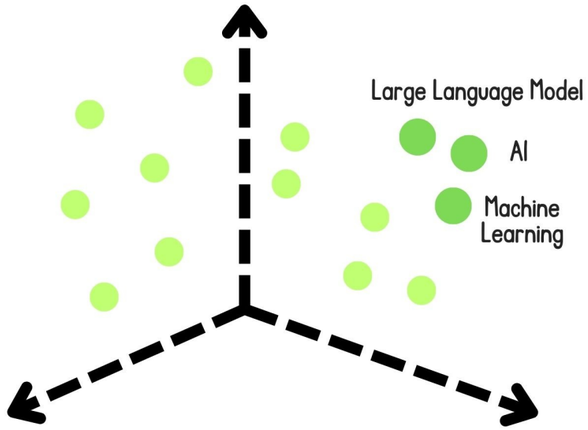
*Encode chunks as a single vector with **non-zero** components*



- **Pros:** Capturing semantic meaning
- **Cons:** Computationally expensive

Dense

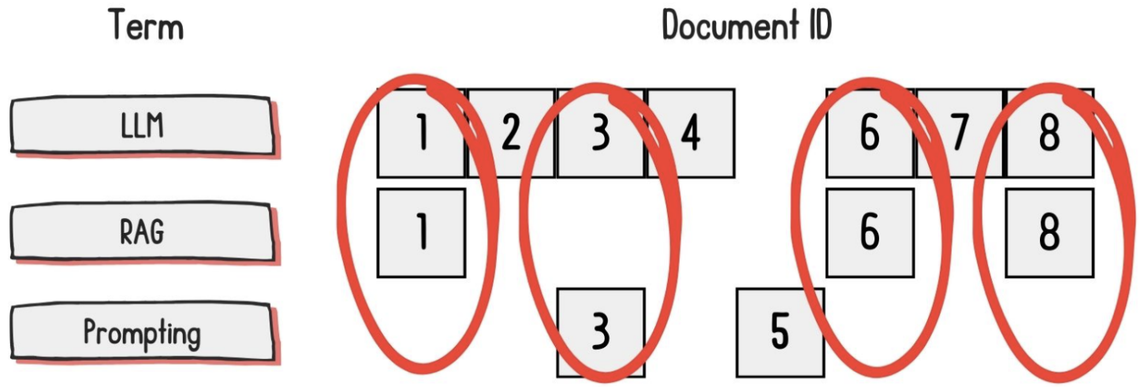
*Encode chunks as a single vector with **non-zero** components*



- **Pros:** Capturing semantic meaning
- **Cons:** Computationally expensive

Sparse

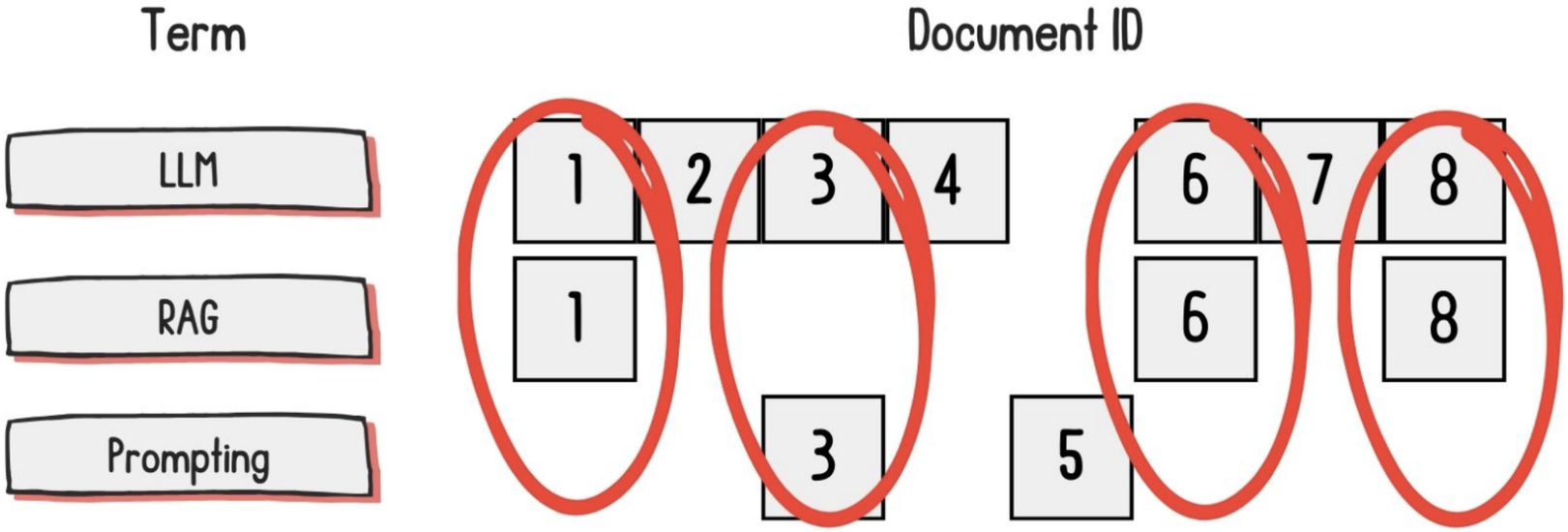
*Encode using **word matching** with mostly **zero** components*



- **Pros:** Precise, explainable, rare-word handling
- **Cons:** Generalizability

Sparse retrieval methods

TF-IDF: Encodes documents using the words that make the document unique



BM25: Helps mitigate high-frequency words from saturating the encoding

BM25 retrieval

```
from langchain_community.retrievers import BM25Retriever

chunks = [
    "Python was created by Guido van Rossum and released in 1991.",
    "Python is a popular language for machine learning (ML).",
    "The PyTorch library is a popular Python library for AI and ML."
]

bm25_retriever = BM25Retriever.from_texts(chunks, k=3)
```

BM25 retrieval

```
results = bm25_retriever.invoke("When was Python created?")
print("Most Relevant Document:")
print(results[0].page_content)
```

```
Most Relevant Document:
Python was created by Guido van Rossum and released in 1991.
```

- Python was created by Guido van Rossum and released in 1991."
- "Python is a popular language for machine learning (ML)."
- "The PyTorch library is a popular Python library for AI/ML."

BM25 in RAG

```
retriever = BM25Retriever.from_documents(
    documents=chunks,
    k=5
)

chain = ({"context": retriever, "question": RunnablePassthrough()}
         | prompt
         | llm
         | StrOutputParser()
)
```

[1] https://www.datacamp.com/blog/what-is-retrieval-augmented-generation-rag

BM25 in RAG

```
print(chain.invoke("How can LLM hallucination impact a RAG application?"))
```

```
The RAG application may generate responses that are off-topic or inaccurate.
```

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

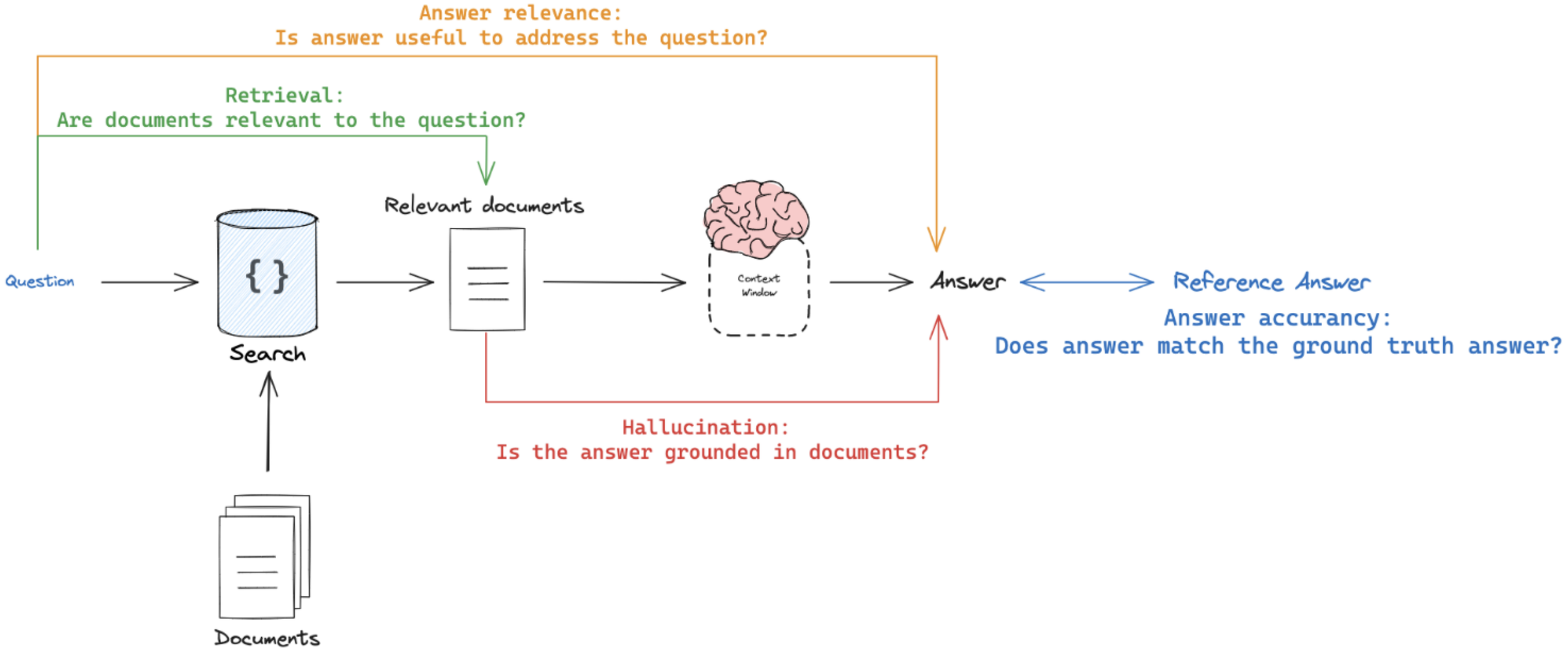
Introduction to RAG evaluation

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Meri Nova
Machine Learning Engineer

datacamp

Types of RAG evaluation



[1] Image Credit: LangSmith

Output accuracy: string evaluation

```
query = "What are the main components of RAG architecture?"
predicted_answer = "Training and encoding"
ref_answer = "Retrieval and Generation"
```

Output accuracy: string evaluation

```
prompt_template = """You are an expert professor specialized in grading students' answers to q
You are grading the following question:{query}
Here is the real answer:{answer}
You are grading the following predicted answer:{result}
Respond with CORRECT or INCORRECT:
Grade:"""

prompt = PromptTemplate(
    input_variables=["query", "answer", "result"],
    template=prompt_template
)

eval_llm = ChatOpenAI(temperature=0, model="gpt-4o-mini", openai_api_key='...')
```

Output accuracy: string evaluation

```
from langsmith.evaluation import LangChainStringEvaluator

qa_evaluator = LangChainStringEvaluator(
    "qa",
    config={
        "llm": eval_llm,
        "prompt": PROMPT
    }
)

score = qa_evaluator.evaluator.evaluate_strings(
    prediction=predicted_answer,
    reference=ref_answer,
    input=query
)
```

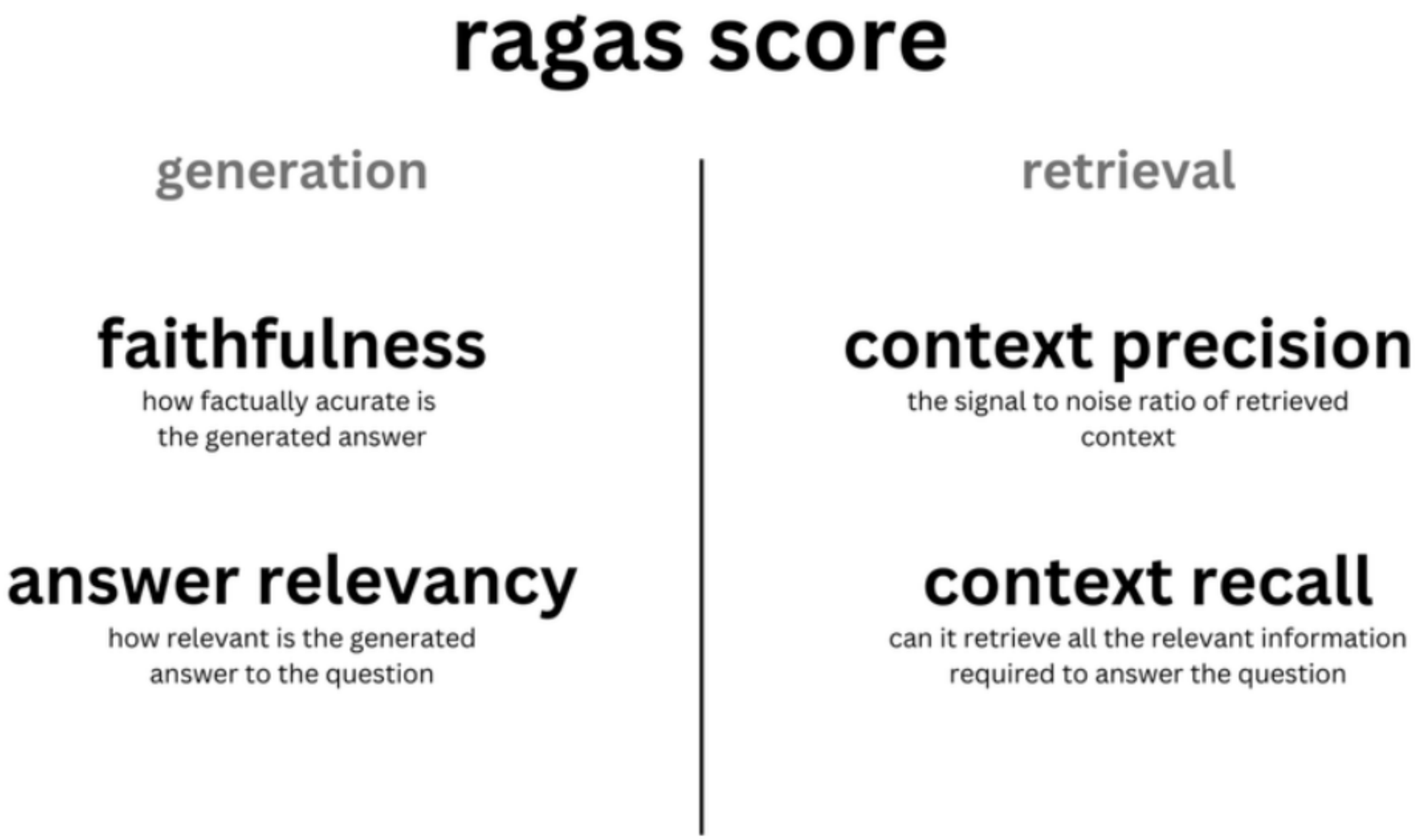
Output accuracy: string evaluation

```
print(f"Score: {score}")
```

```
Score: {'reasoning': 'INCORRECT', 'value': 'INCORRECT', 'score': 0}
```

```
query = "What are the main components of RAG architecture?"
predicted_answer = "Training and encoding"
ref_answer = "Retrieval and Generation"
```

Ragas framework



[1] Image Credit: Ragas

Faithfulness

- *Does the generated output faithfully represent the context?*

$$\text{Faithfulness} = \frac{\text{No. of claims made that can be inferred from the context}}{\text{Total no. of claims}}$$

- **Normalized** to `(0, 1)`

Evaluating faithfulness

```
from langchain_openai import ChatOpenAI, OpenAIEmbeddings
from ragas.integrations.langchain import EvaluatorChain
from ragas.metrics import faithfulness

llm = ChatOpenAI(model="gpt-4o-mini", api_key="...")
embeddings = OpenAIEmbeddings(model="text-embedding-3-small", api_key="...")

faithfulness_chain = EvaluatorChain(
    metric=faithfulness,
    llm=llm,
    embeddings=embeddings
)
```

Evaluating faithfulness

```
eval_result = faithfulness_chain({
  "question": "How does the RAG model improve question answering with LLMs?",
  "answer": "The RAG model improves question answering by combining the retrieval of documents...",
  "contexts": [
    "The RAG model integrates document retrieval with LLMs by first retrieving relevant passages...",
    "By incorporating retrieval mechanisms, RAG leverages external knowledge sources, allowing the...",
  ]
})

print(eval_result)
```

```
'faithfulness': 1.0
```

Context precision

- *How relevant are the retrieved documents to the query?*
- **Normalized** to `(0, 1)` → `1` = highly relevant

```
from ragas.metrics import context_precision

llm = ChatOpenAI(model="gpt-4o-mini", api_key="...")
embeddings = OpenAIEmbeddings(model="text-embedding-3-small", api_key="...")

context_precision_chain = EvaluatorChain(
    metric=context_precision,
    llm=llm,
    embeddings=embeddings
)
```

Evaluating context precision

```
eval_result = context_precision_chain({
  "question": "How does the RAG model improve question answering with large language models?",
  "ground_truth": "The RAG model improves question answering by combining the retrieval of...",
  "contexts": [
    "The RAG model integrates document retrieval with LLMs by first retrieving...",
    "By incorporating retrieval mechanisms, RAG leverages external knowledge sources...",
  ]
})

print(f"Context Precision: {eval_result['context_precision']}")
```

```
Context Precision: 0.99999999995
```

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

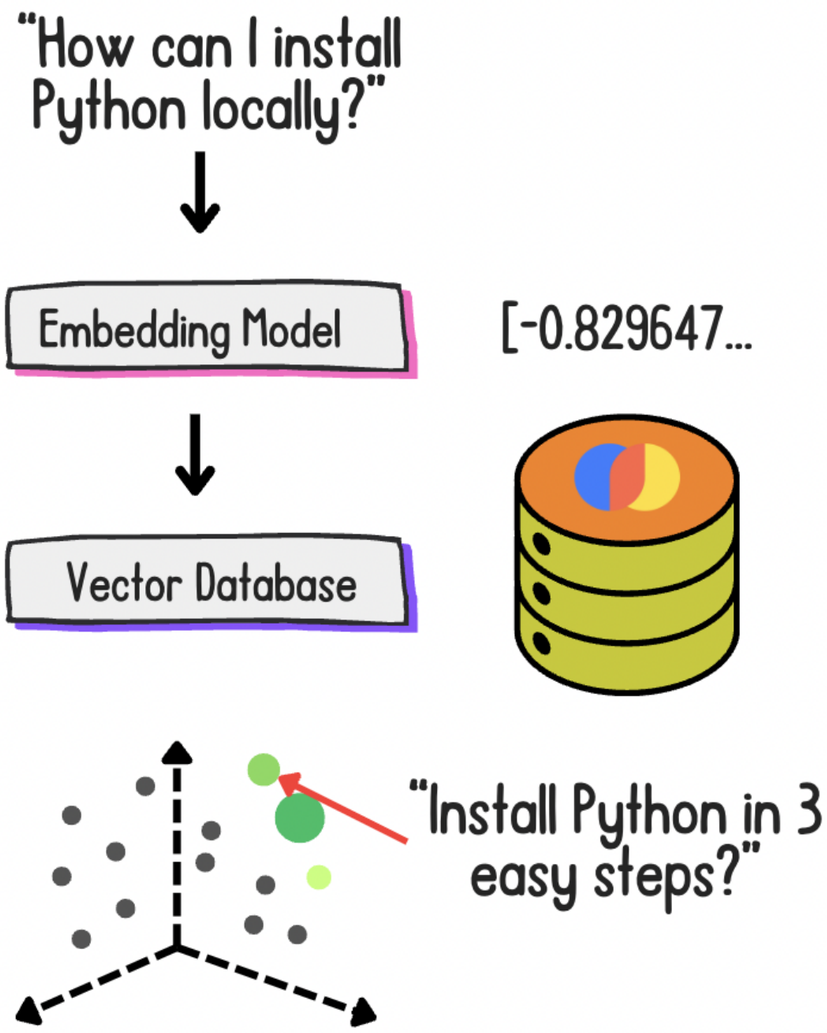
From vectors to graphs

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

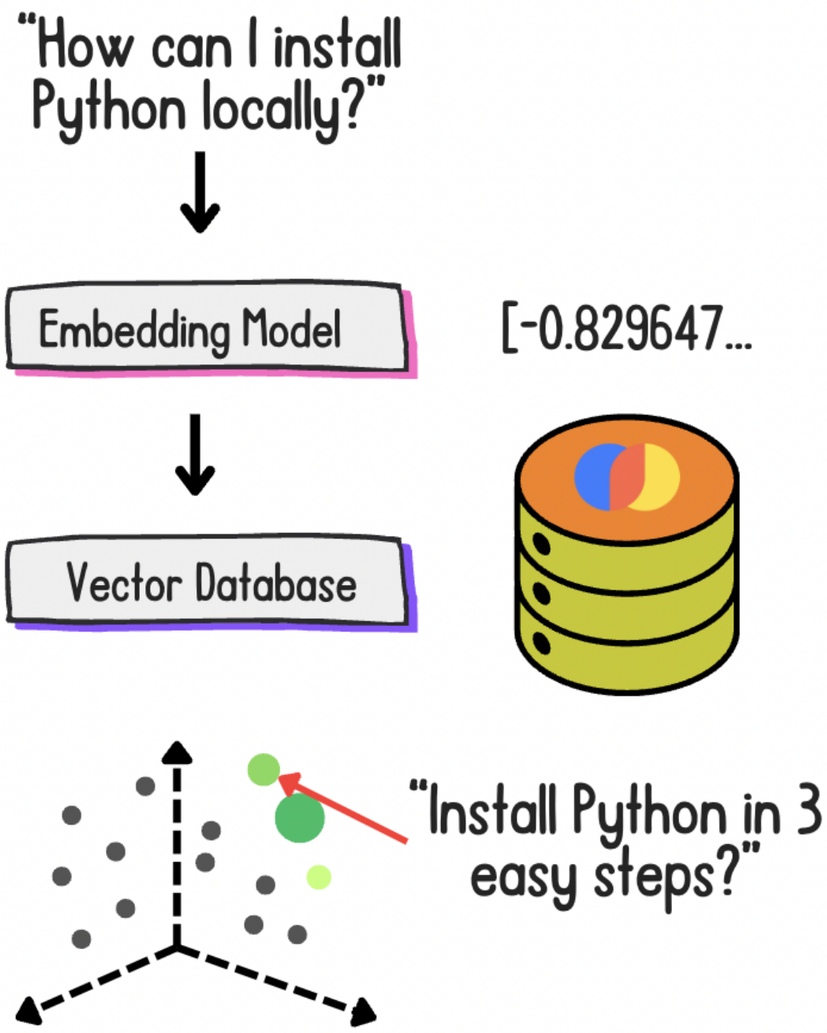
Meri Nova
Machine Learning Engineer

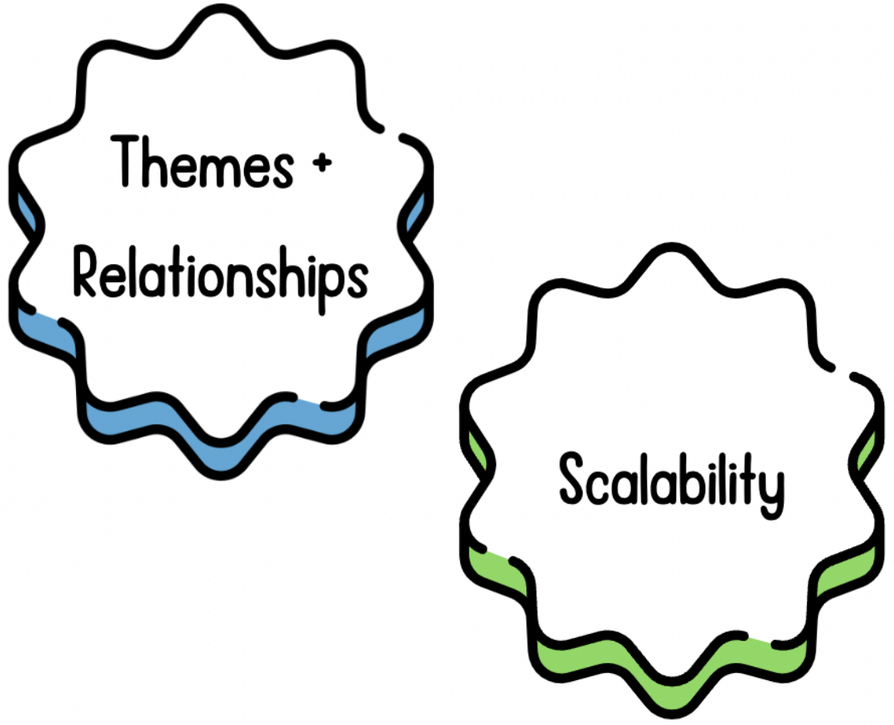
datacamp

Vector RAG limitations

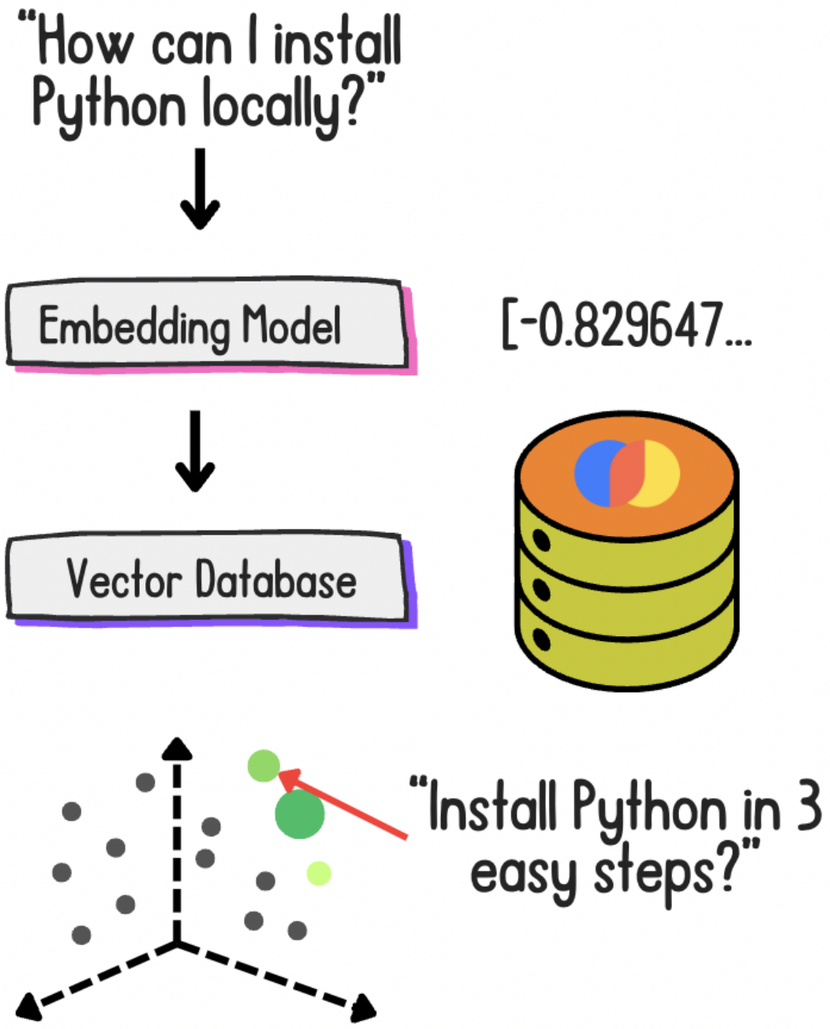


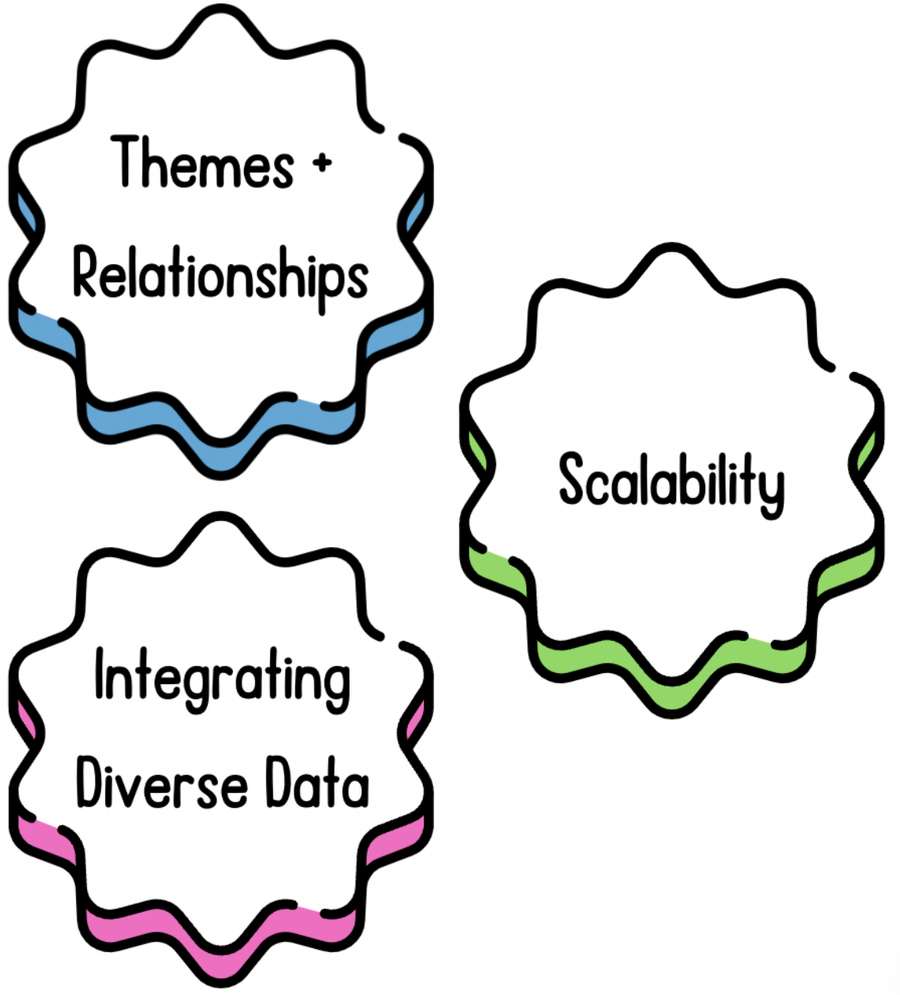
Vector RAG limitations



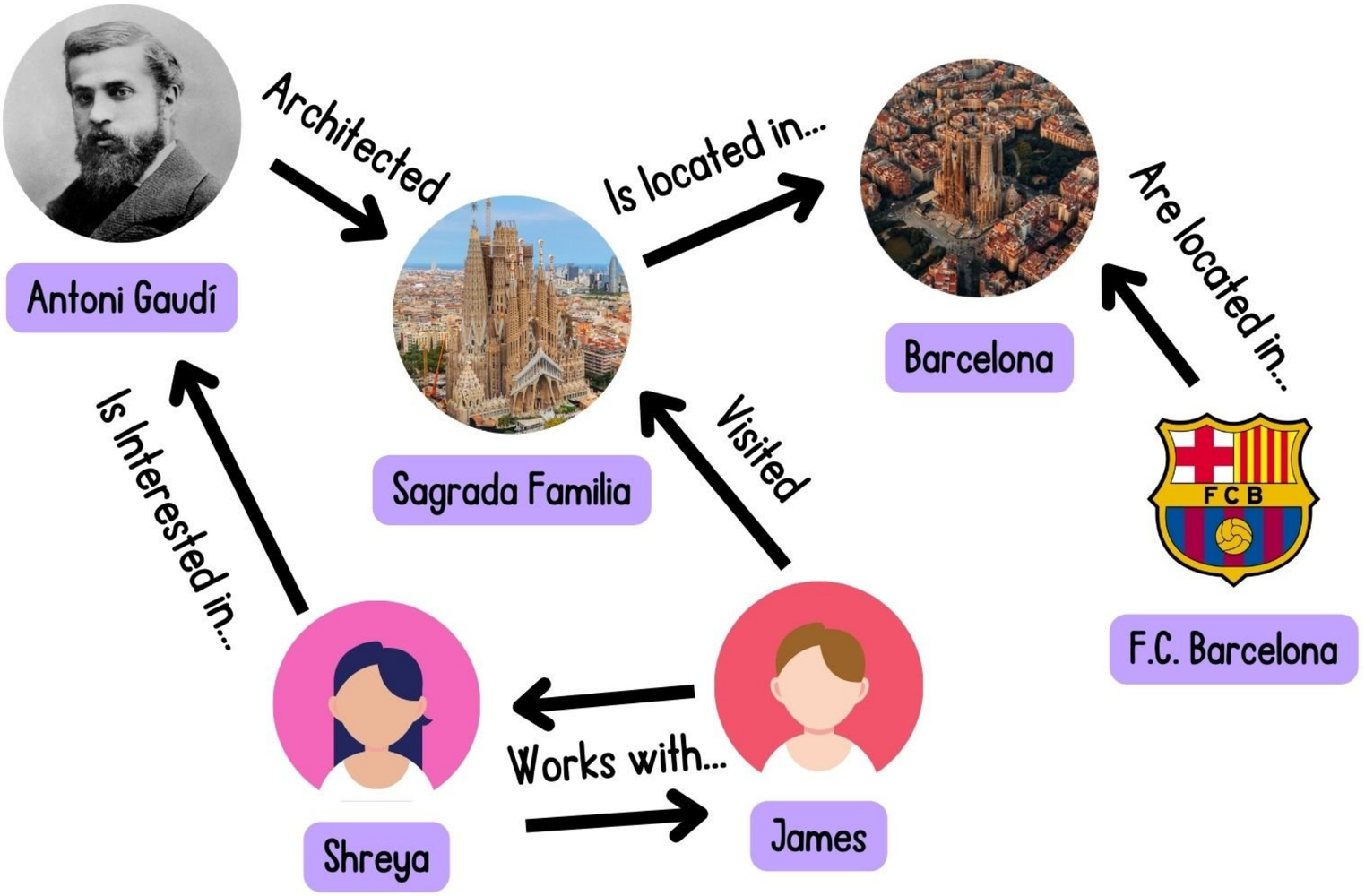


Vector RAG limitations

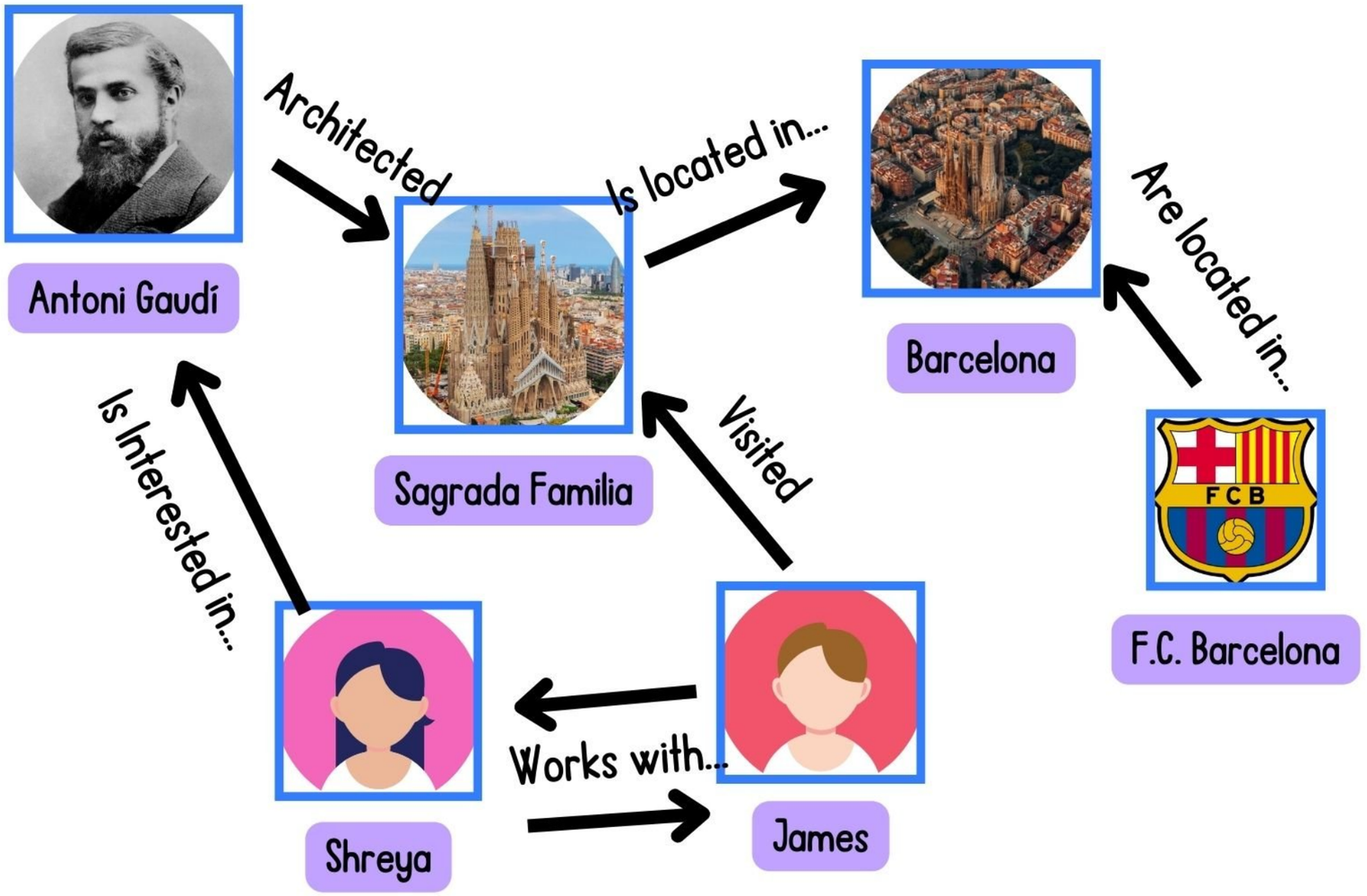




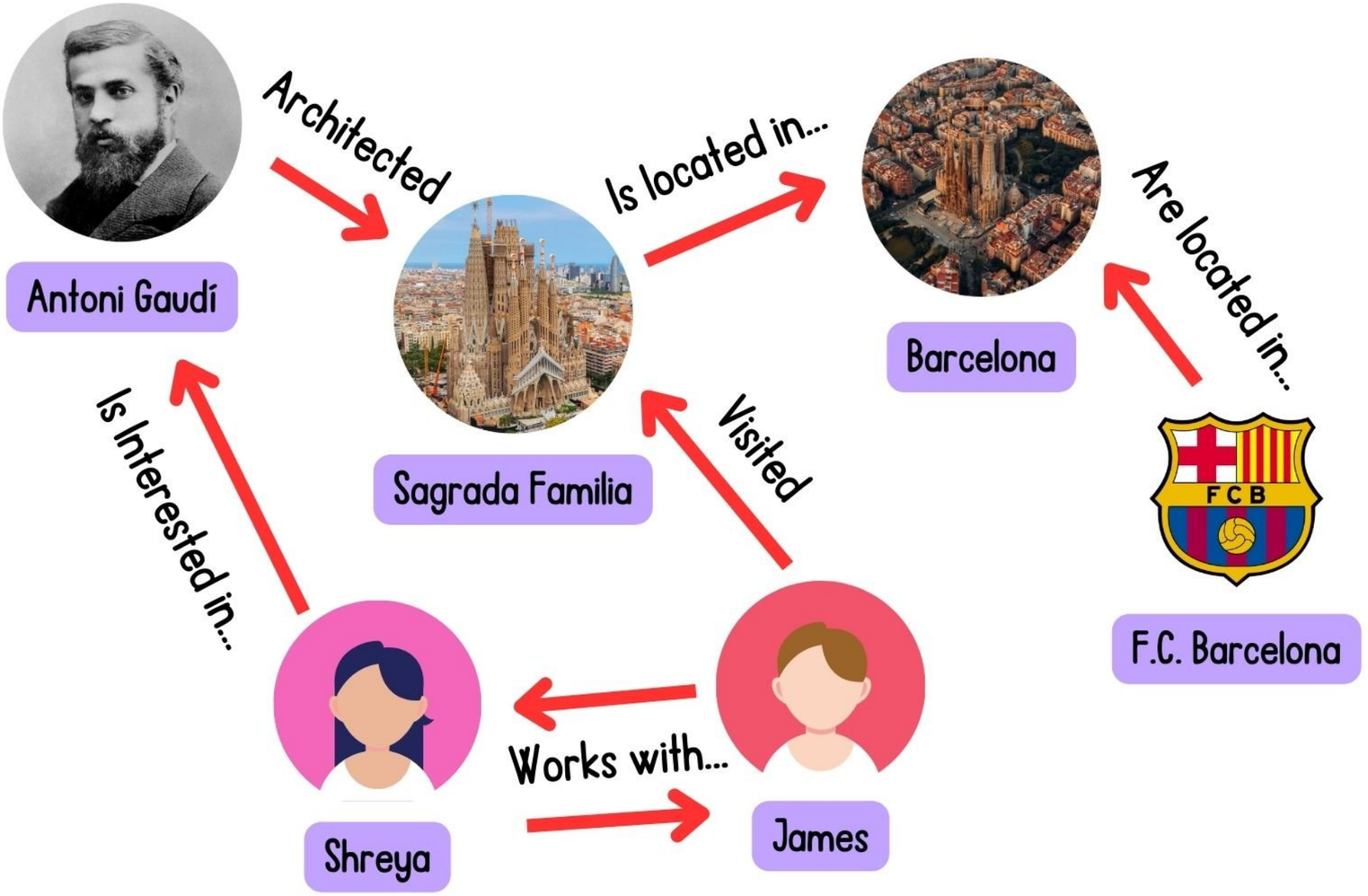
Graph databases



Graph databases - nodes



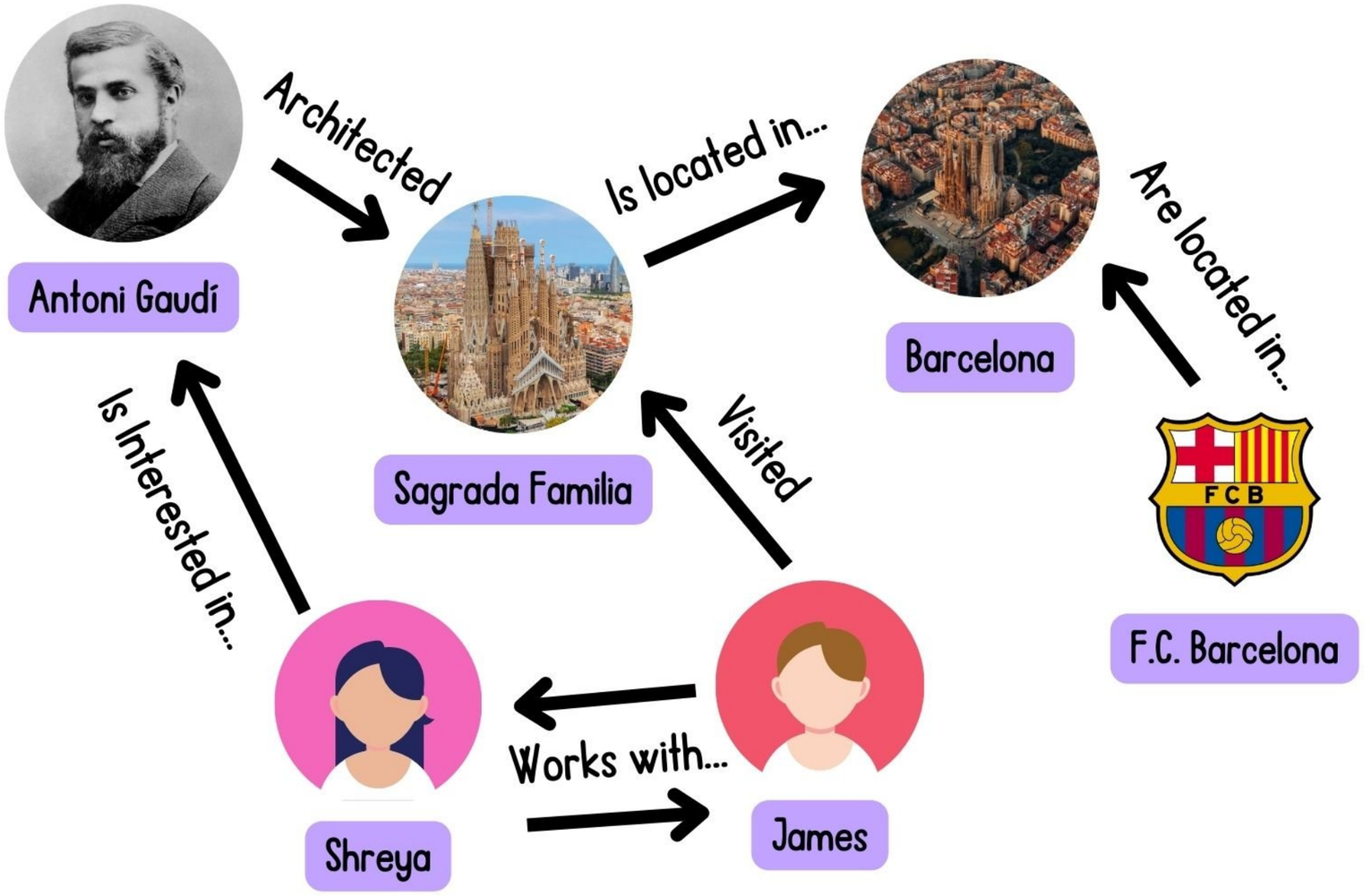
Graph databases - edges



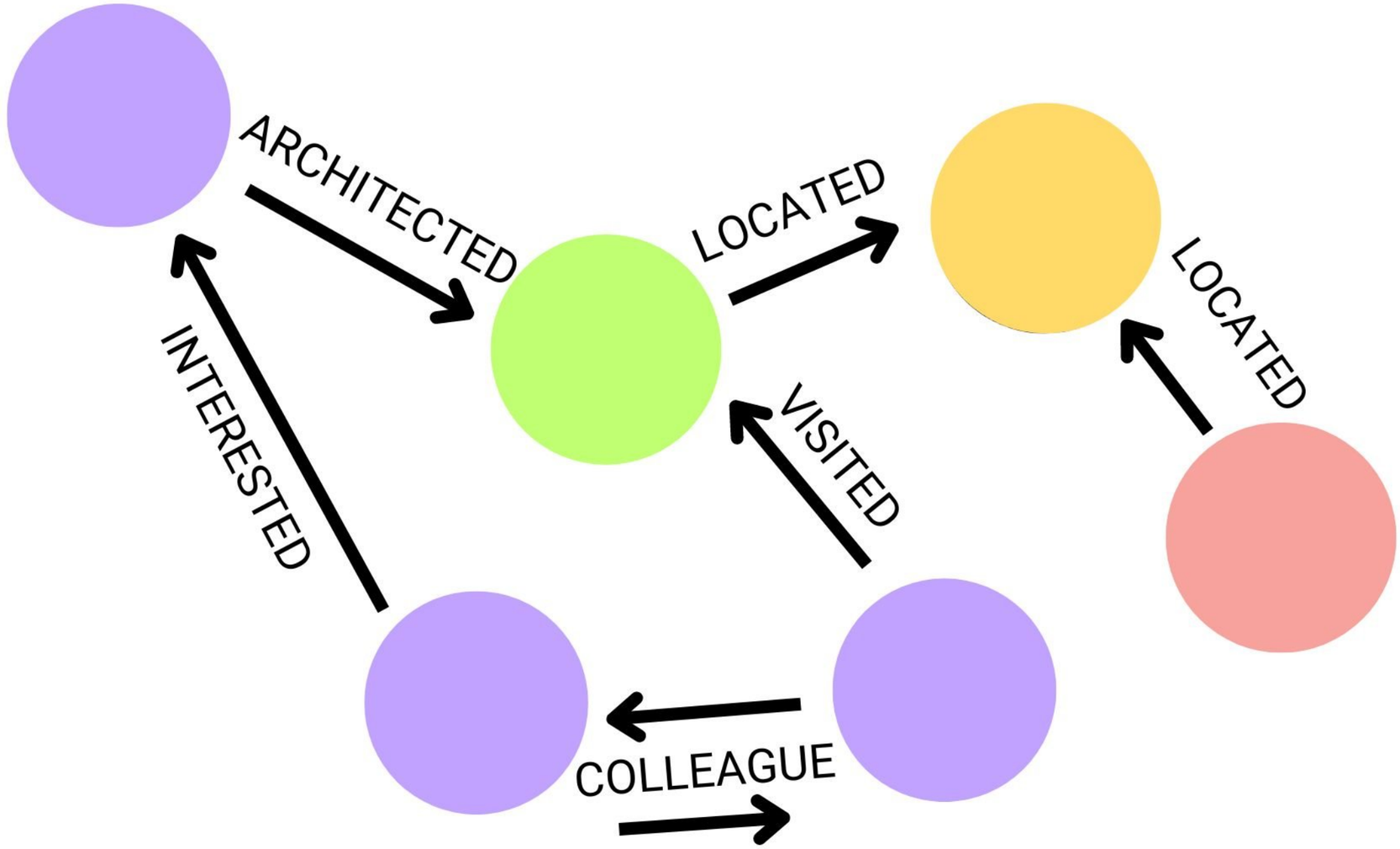
Neo4j graph databases

LangChain

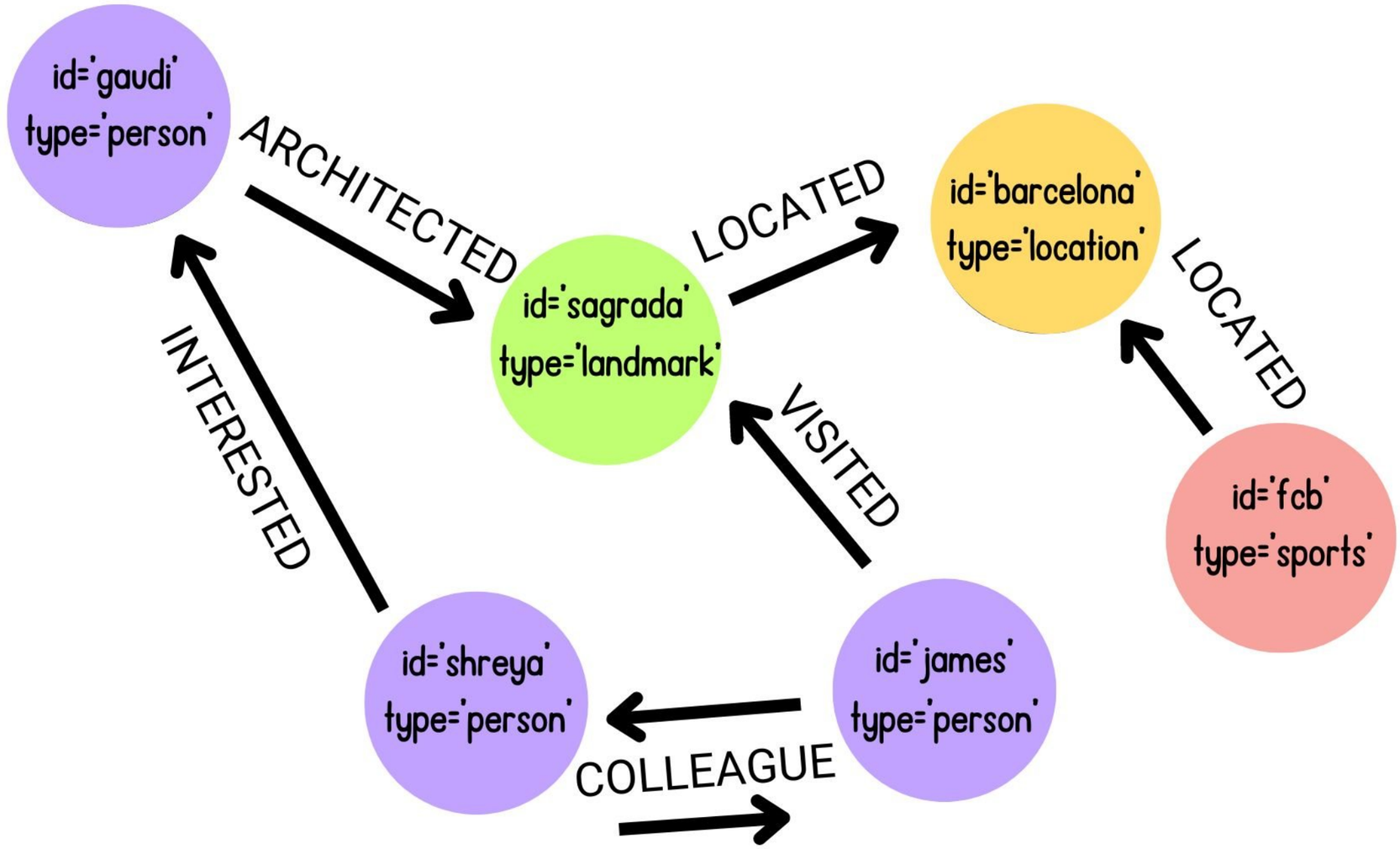
From graphics to graphs...



From graphics to graphs...



From graphics to graphs...



Loading and chunking Wikipedia pages

```
from langchain_community.document_loaders import WikipediaLoader
from langchain_text_splitters import TokenTextSplitter

raw_documents = WikipediaLoader(query="large language model").load()
text_splitter = TokenTextSplitter(chunk_size=100, chunk_overlap=20)
documents = text_splitter.split_documents(raw_documents[:3])

print(documents[0])
```

```
page_content='A large language model (LLM) is a computational model capable of...'
metadata={'title': 'Large language model',
          'summary': "A large language model (LLM) is...",
          'source': 'https://en.wikipedia.org/wiki/Large_language_model'}
```

From text to graphs!

```
from langchain_openai import ChatOpenAI
from langchain_experimental.graph_transformers import LLMGraphTransformer

llm = ChatOpenAI(api_key="...", temperature=0, model_name="gpt-4o-mini")
llm_transformer = LLMGraphTransformer(llm=llm)

graph_documents = llm_transformer.convert_to_graph_documents(documents)
print(graph_documents)
```

From text to graphs!

```
[GraphDocument(
    nodes=[
        Node(id='Llm', type='Computational model'),
        Node(id='Language Generation', type='Concept'),
        Node(id='Natural Language Processing Tasks', type='Concept'),
        Node(id='Llama Family', type='Computational model'),
        Node(id='Ibm', type='Organization'),
        ..., Node(id='Bert', type='Computational model')],
    relationships=[
        Relationship(source=Node(id='Llm', type='Computational model'),
                     target=Node(id='Language Generation', type='Concept'),
                     type='CAPABLE_OF'),
        ...])]
```

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Storing and querying documents

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Meri Nova
Machine Learning Engineer

datacamp

Instantiating the Neo4j database

```
from langchain_community.graphs import Neo4jGraph

graph = Neo4jGraph(url="bolt://localhost:7687", username="neo4j", password="...")
```

```
import os

url = os.environ["NEO4J_URI"]
user = os.environ["NEO4J_USERNAME"]
password = os.environ["NEO4J_PASSWORD"]

graph = Neo4jGraph(url=url, username=user, password=password)
```

[1] https://neo4j.com/download/

Storing graph documents

```
from langchain_experimental.graph_transformers import LLMGraphTransformer

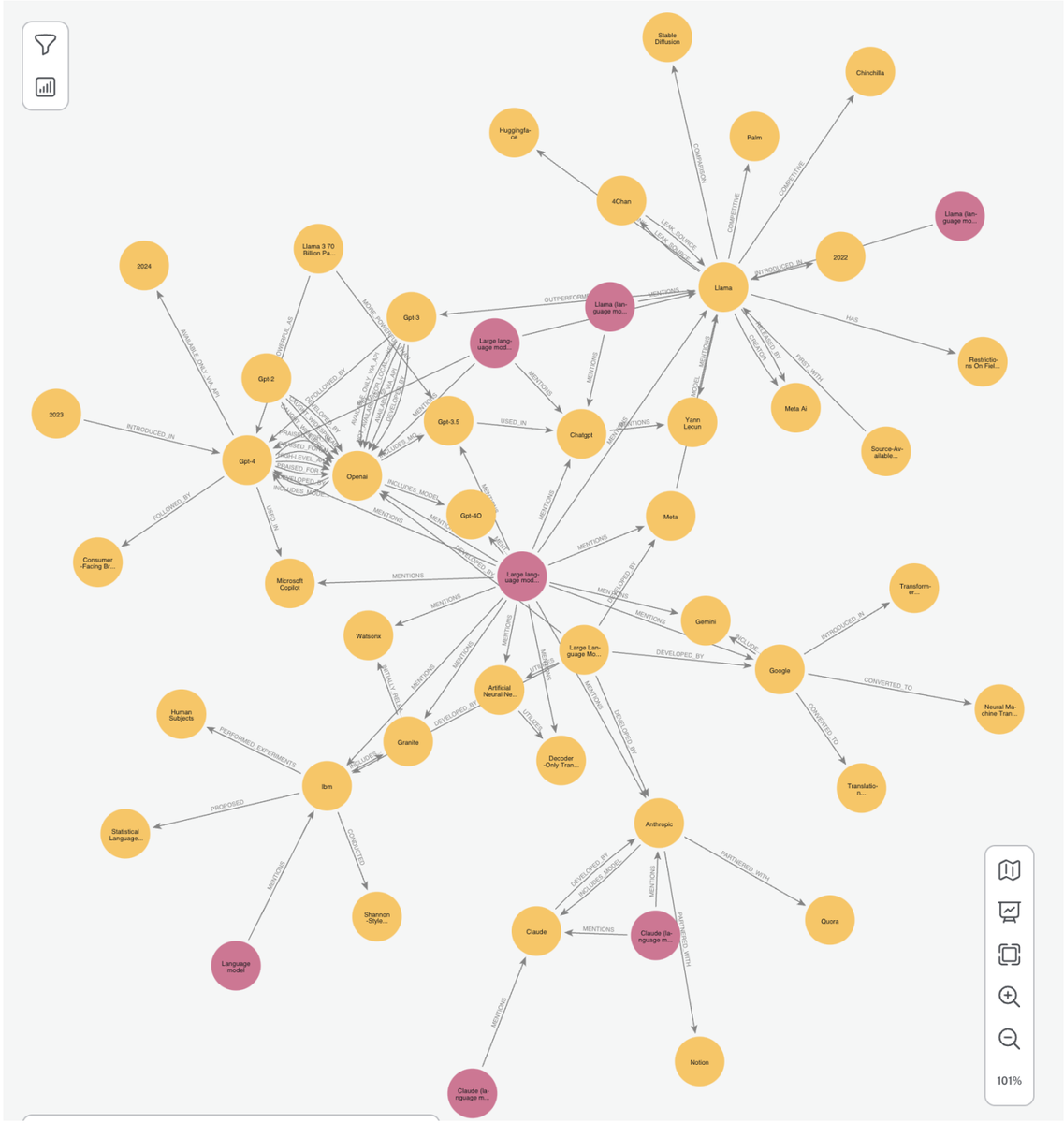
llm = ChatOpenAI(api_key="...", temperature=0, model="gpt-4o-mini")
llm_transformer = LLMGraphTransformer(llm=llm)

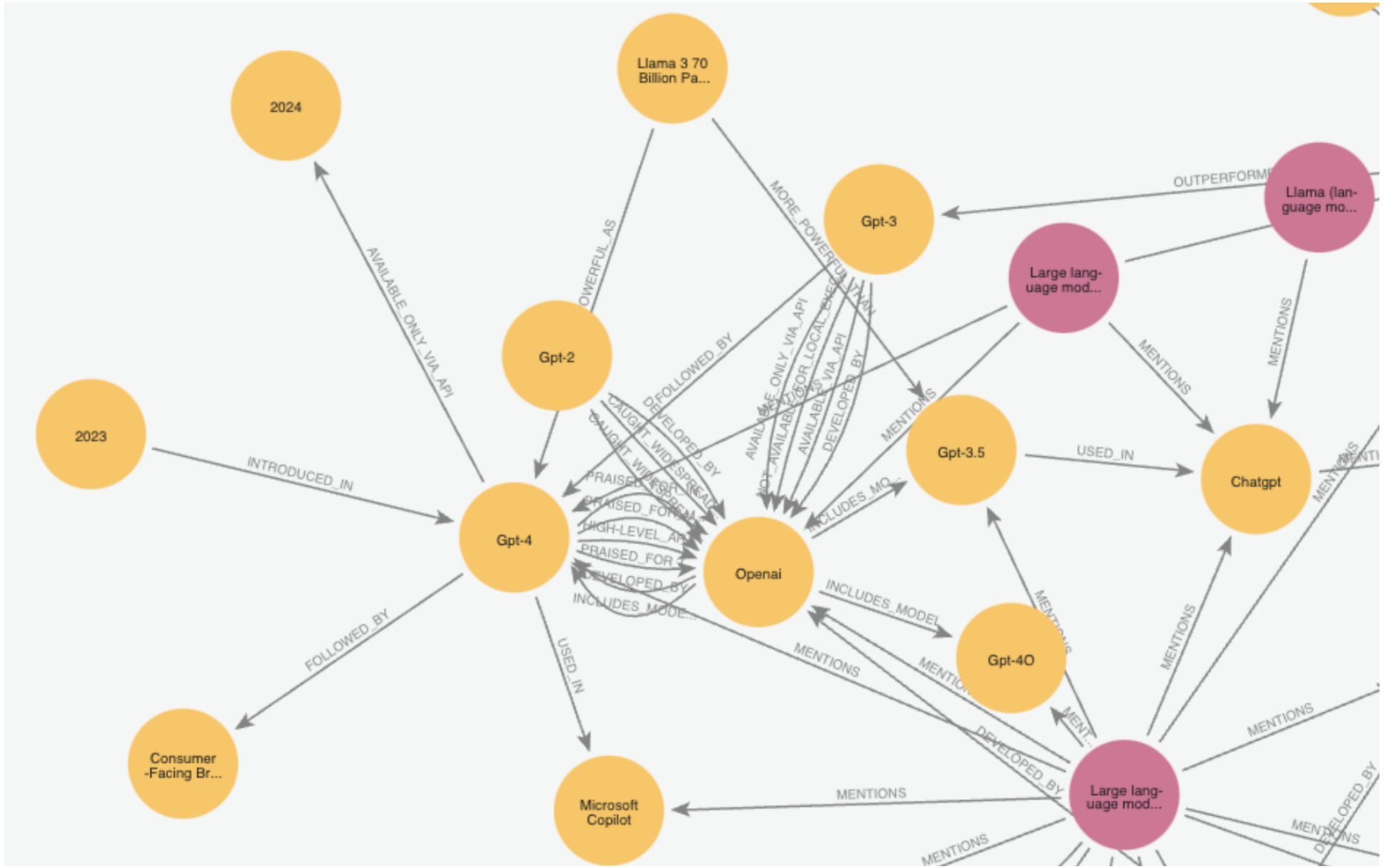
graph_documents = llm_transformer.convert_to_graph_documents(documents)
```

Storing graph documents

```
graph.add_graph_documents(
  graph_documents,
  include_source=True,
  baseEntityLabel=True
)
```

- `include_source=True` : link nodes to source documents with `MENTIONS` edge
- `baseEntityLabel=True` : add `__Entity__` label to each node


Stable Diffusion
Chinchilla
Huggingfa-ce
Palm
4Chan
Llama (lan-guage mo...
2022
Llama
2024
Llama 3 70 Billion Pa...
Gpt-3
Large lang-uage mod...
Gpt-2
Plastricto-ns On Flat...
2023
Gpt-3.5
Chatgpt
Yann Lecun
Meta AI
Source-Av-ailable...
Gpt-4
Openai
Gpt-4O
Meta
Consumer -Facing Br...
Microsoft Copilot
Large lang-uage mod...
Transform-er...
Watsonx
Gemini
Large Lan-guage Mo...
Google
Artificial Neural Ne...
Neural Ma-chine Tran...
Human Subjects
Granite
Decoder -Only Tran...
Translatio-n...
Ibm
Anthropic
Statistical Language...
Shannon -Style...
Claude
Claude (la-nguage m...
Quora
Language model
Notion
INTRODUCED_IN
FOLLOWED_BY
USED_IN
MENTIONS
DEVELOPED_BY
INCLUDES_MODEL
OUTPERFORM
HAS
CREATED_BY
FIRST_WITH
DEVELOPED_BY
INCLUDES
CONVERTED_TO
PERFORMED_EXPERIMENTS
PROPOSED
PARTNERED_WITH
101%


2024
Llama 3 70 Billion Pa...
Gpt-3
Llama (lan-guage mo...
Large lang-uage mod...
Gpt-2
2023
Gpt-3.5
Chatgpt
Gpt-4
Openai
Gpt-4O
Consumer -Facing Br...
Microsoft Copilot
Large lang-uage mod...
AVAILABLE_ONLY_VIA_API
MORE_POWERFUL_THAN
OUTPERFORMED_BY
MENTIONS
INTRODUCED_IN
FOLLOWED_BY
USED_IN
INCLUDES_MODEL
DEVELOPED_BY

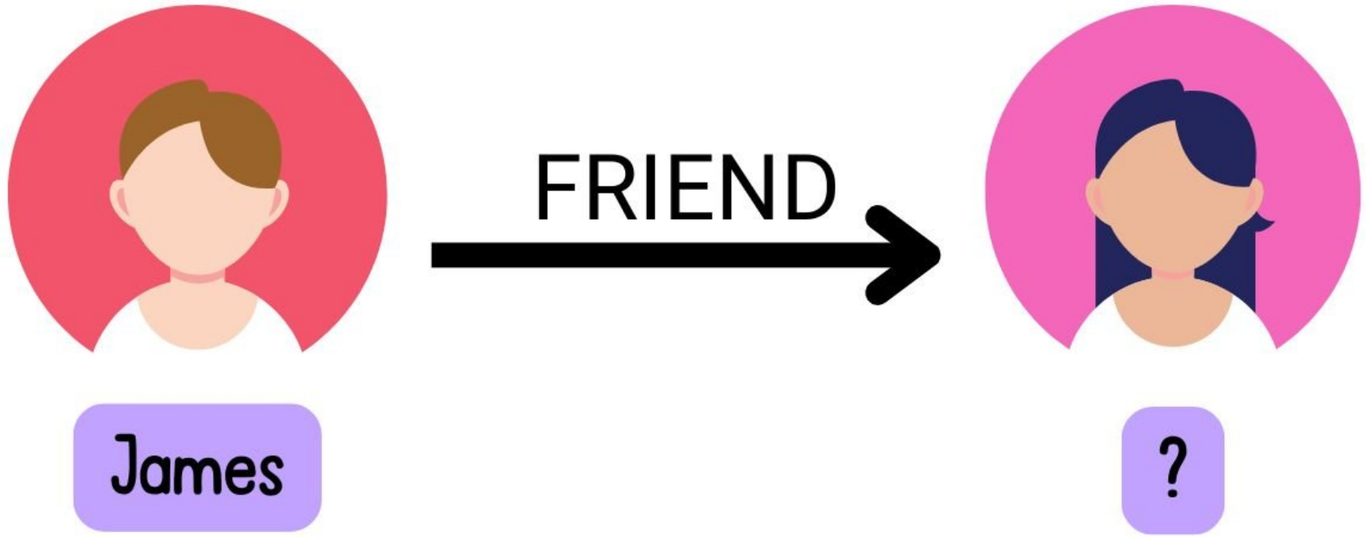
Database schema

```
print(graph.get_schema)
```

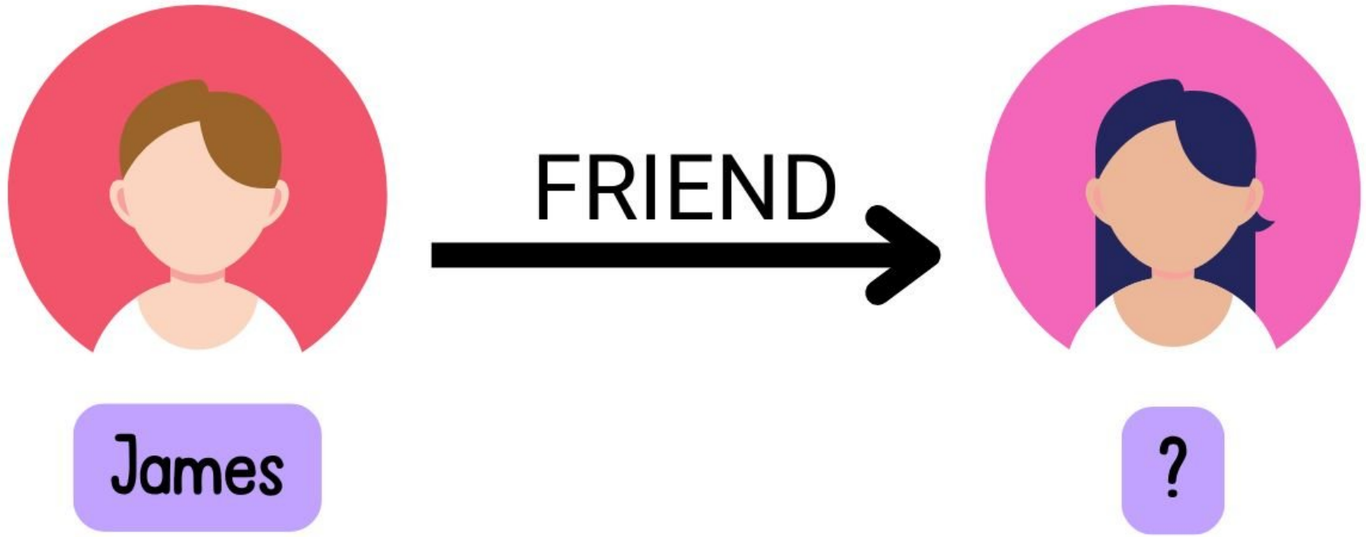
```
Node properties:
Concept {id: STRING}
Architecture {id: STRING}
Organization {id: STRING}
Event {id: STRING}
Paper {id: STRING}

The relationships:
(:Concept)-[:DEVELOPED_BY]->(:Person)
(:Architecture)-[:BASED_ON]->(:Concept)
(:Organization)-[:PROPOSED]->(:Concept)
(:Document)-[:MENTIONS]->(:Event)
(:Paper)-[:BASED_ON]->(:Concept)
```

Querying Neo4j - Cypher Query Language

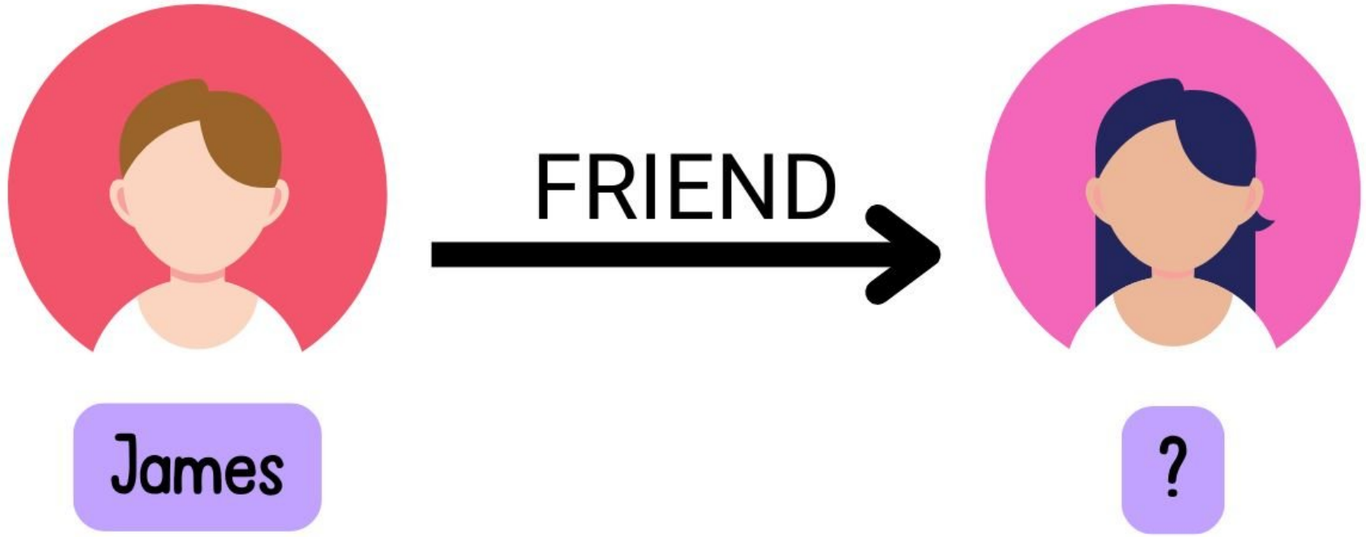


Querying Neo4j - Cypher Query Language




```
MATCH (james:Person {name: "James"})-[:FRIEND]->(friend) RETURN friend
```

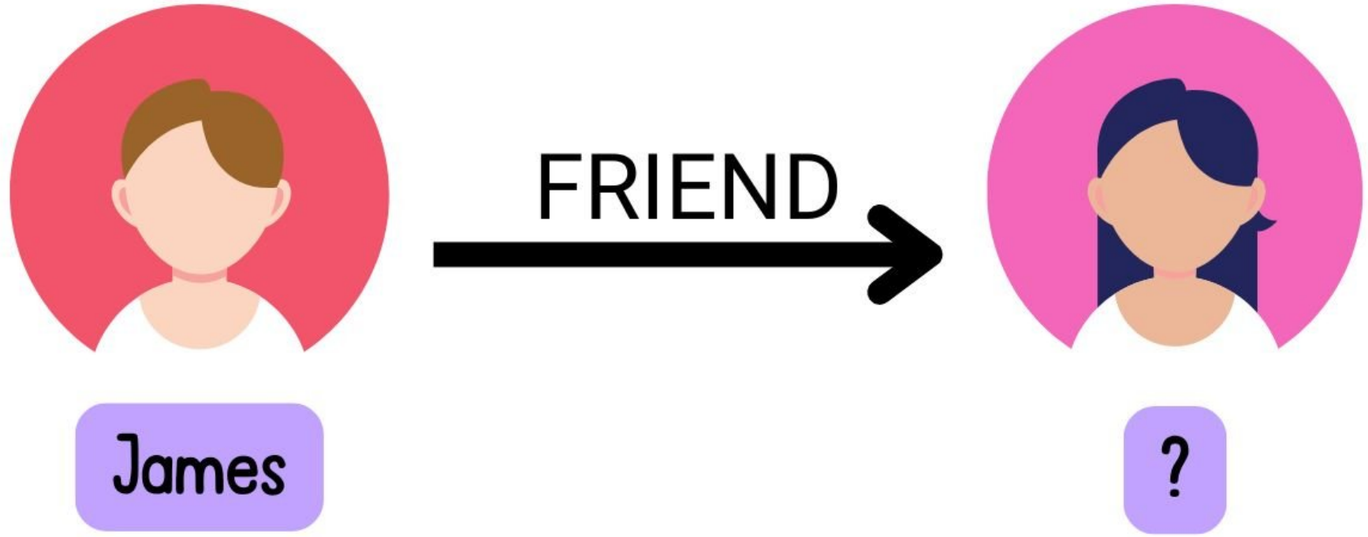
Querying Neo4j - Cypher Query Language



Node Relationship Node

```
MATCH (james:Person {name: "James"})-[:FRIEND]->(friend) RETURN friend
```

Querying Neo4j - Cypher Query Language

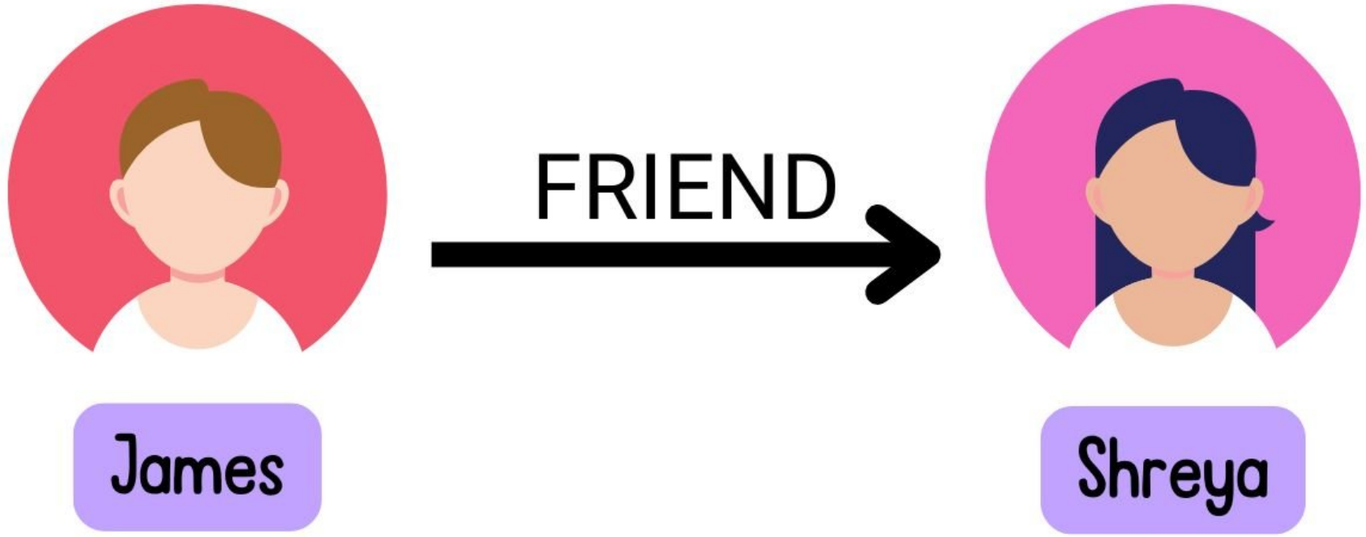


Node Relationship Node

```
MATCH (james:Person {name: "James"})-[:FRIEND]->(friend) RETURN friend
```

LABEL PROPERTY VARIABLE

Querying Neo4j - Cypher Query Language



Node Relationship Node

```
MATCH (james:Person {name: "James"})-[:FRIEND]->(friend) RETURN friend
```

LABEL PROPERTY VARIABLE

Querying the LLM graph

```
results = graph.query("""
MATCH (gpt4:Model {id: "Gpt-4"})-[:DEVELOPED_BY]->(org:Organization)
RETURN org
""")

print(results)
```

```
[{'org': {'id': 'Openai'}}]
```

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

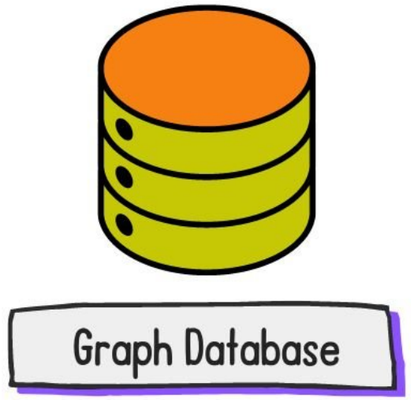
Creating the Graph RAG chain

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

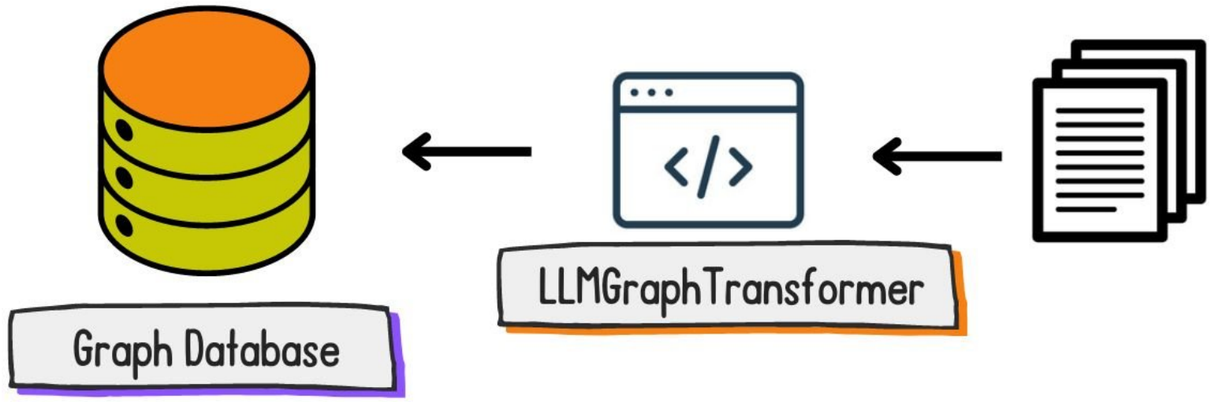
Meri Nova
Machine Learning Engineer

datacamp

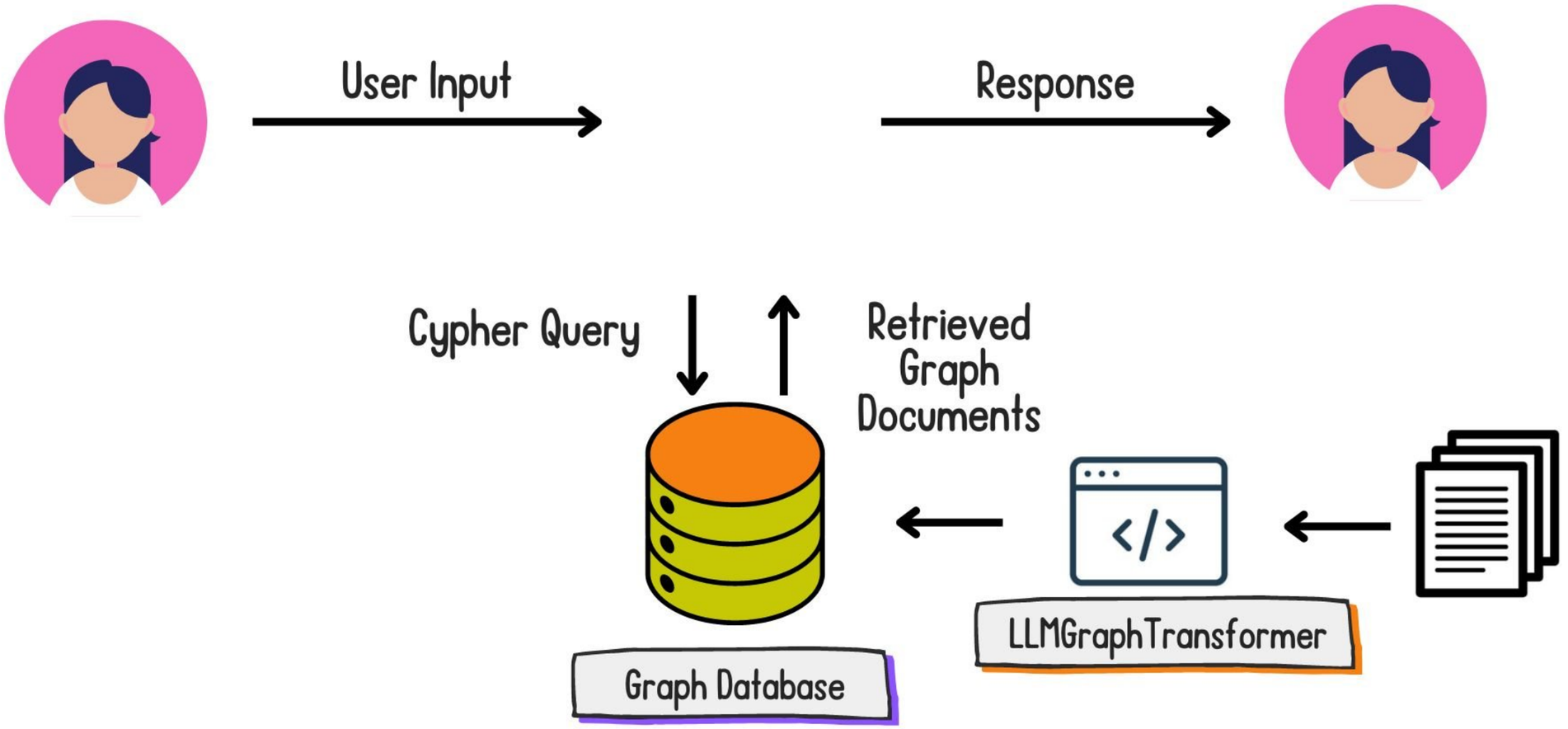
Building the Graph RAG architecture



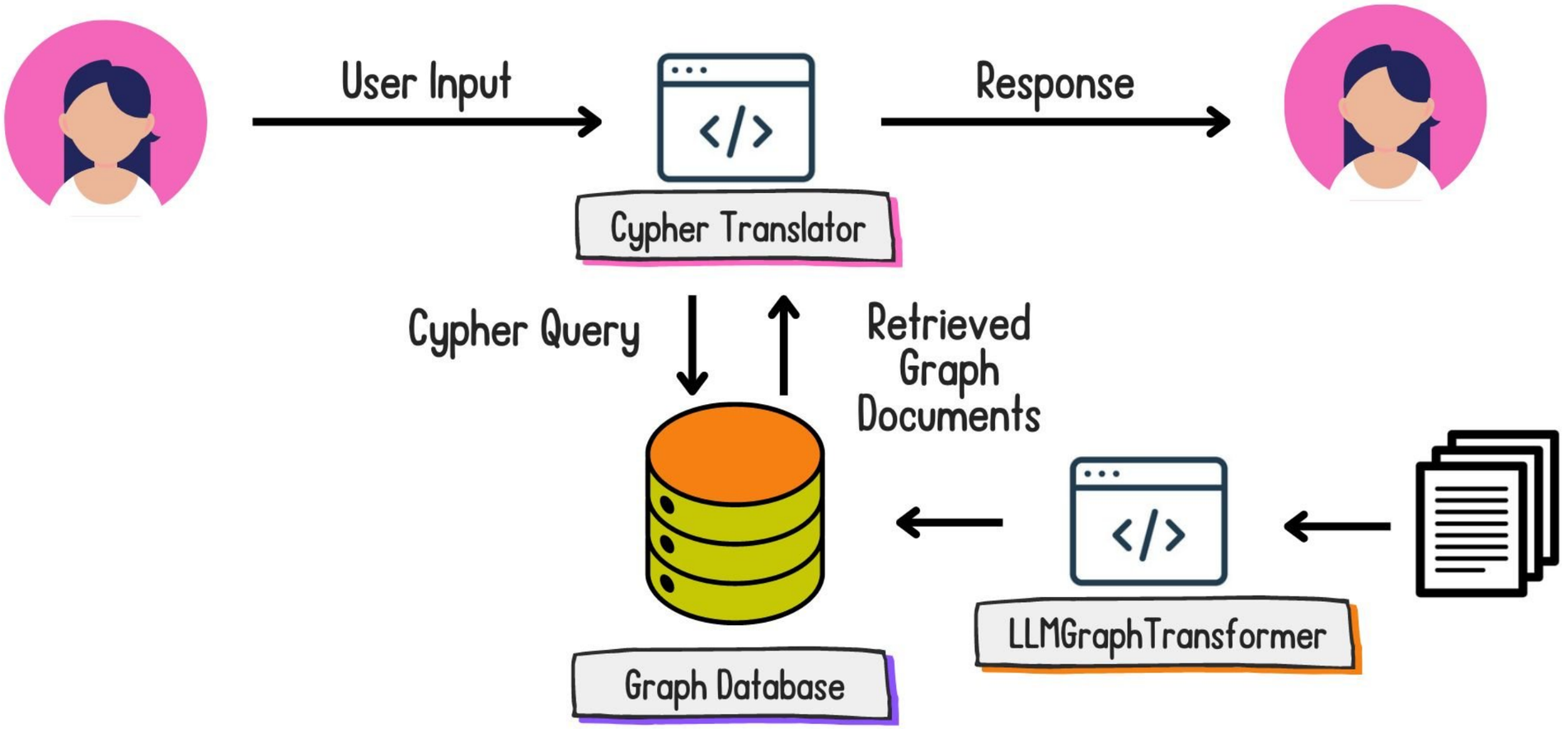
Building the Graph RAG architecture



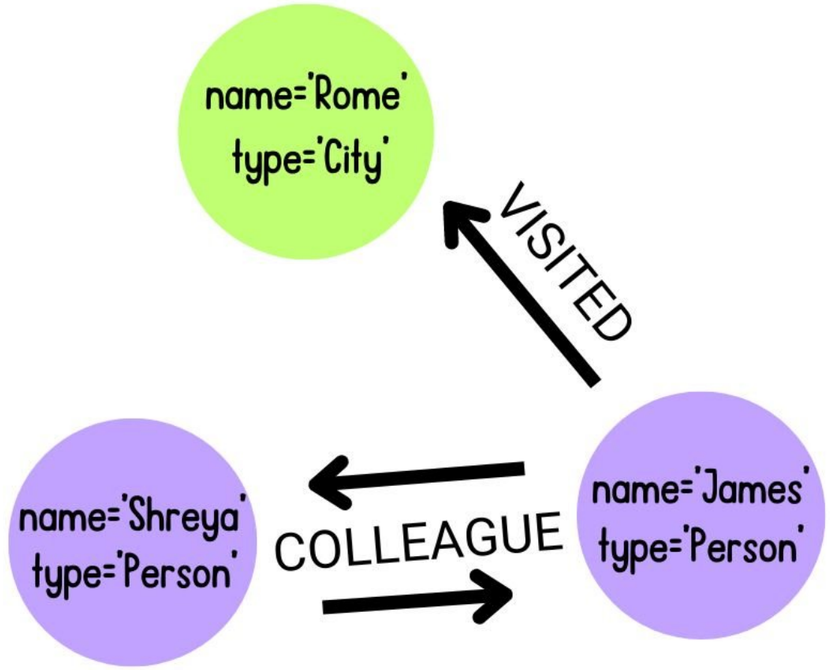
Building the Graph RAG architecture



Building the Graph RAG architecture



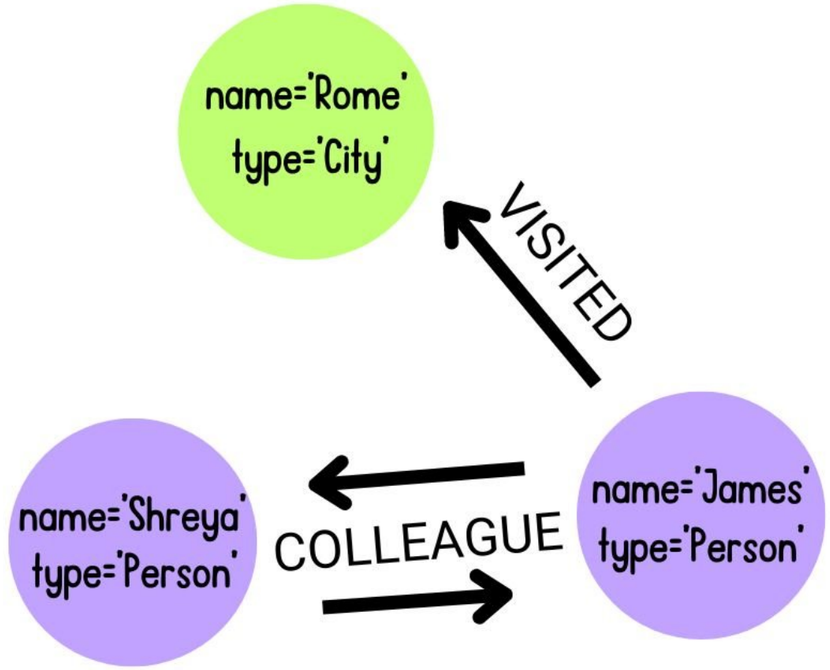
From user inputs to Cypher queries



From user inputs to Cypher queries

Where has James visited?

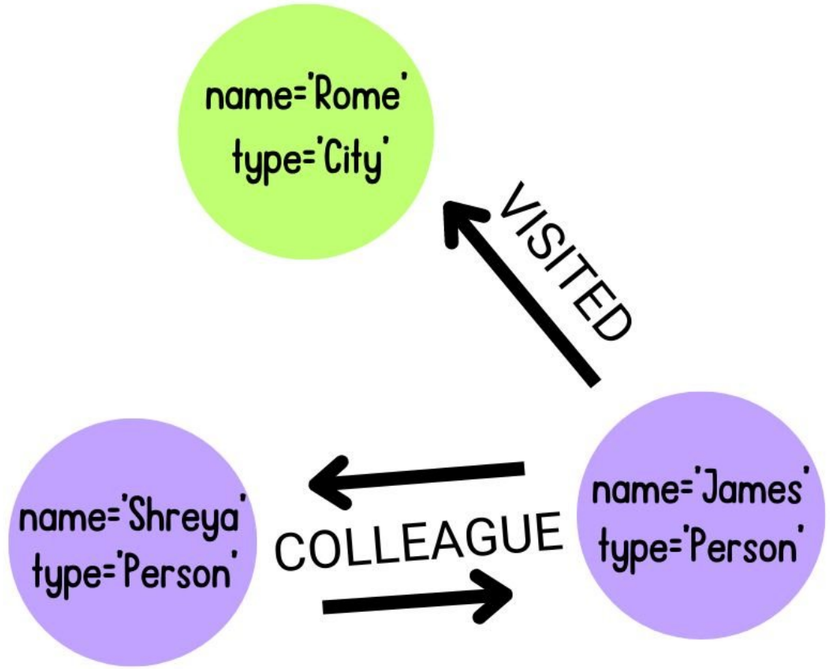
+



From user inputs to Cypher queries

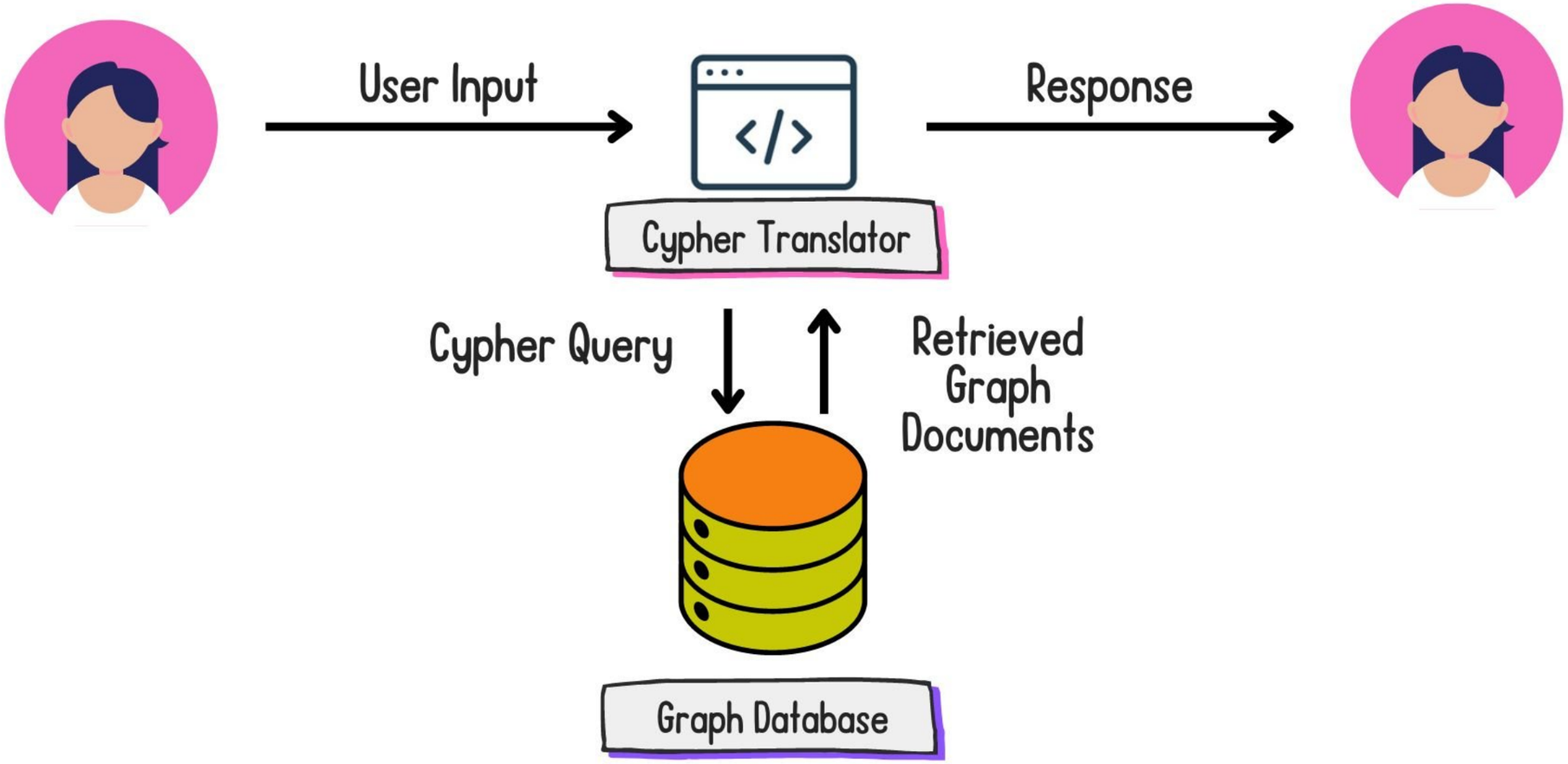
Where has James visited?

+

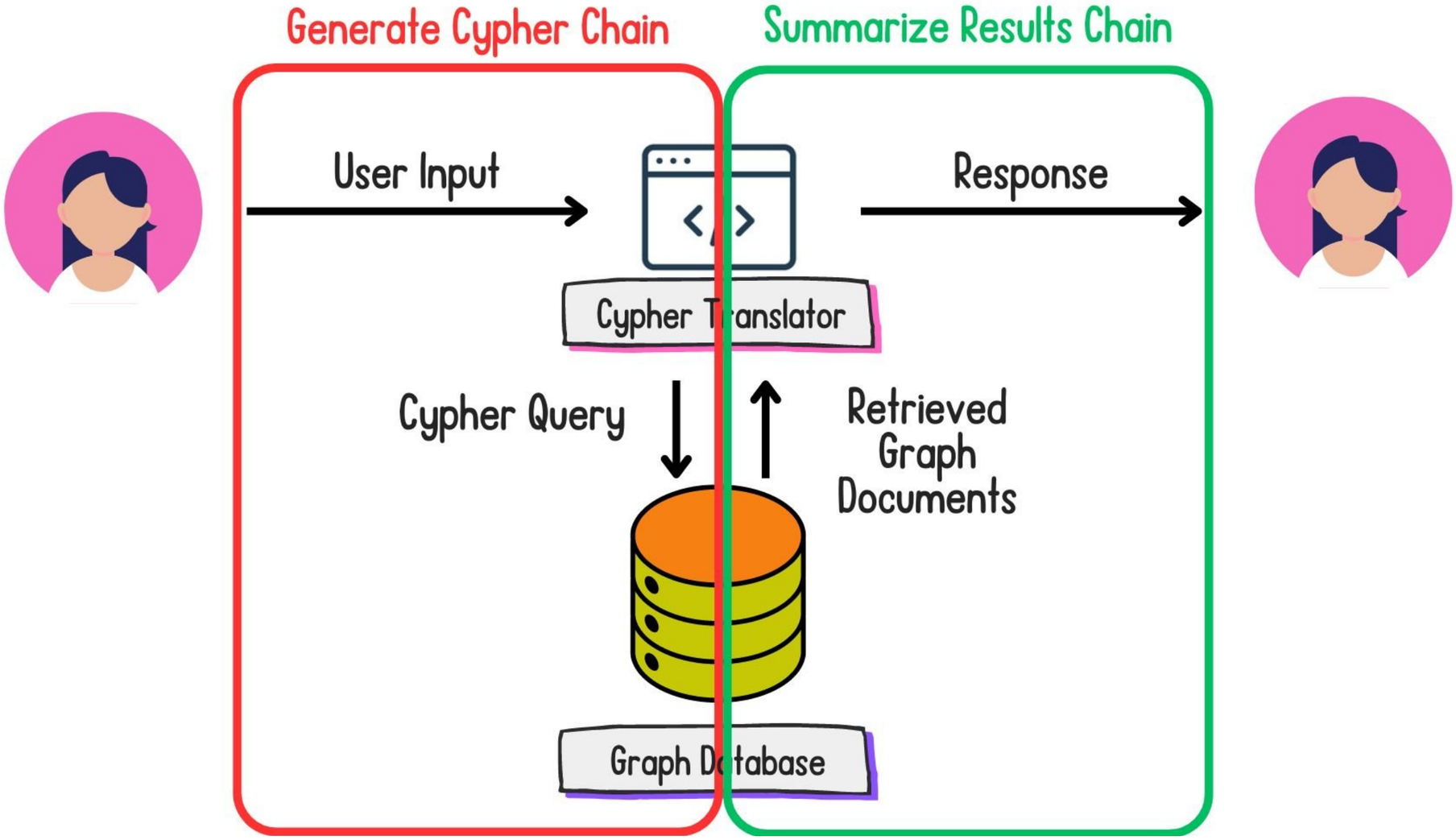



```
MATCH (james:Person{name:"James"}) -[:VISITED]> (location)
RETURN location
```

GraphCypherQAChain



GraphCypherQAChain



Refresh schema

```
graph.refresh_schema()
print(graph.get_schema)
```

```
Node properties:
Document {title: STRING, id: STRING, text: STRING, summary: STRING, source: STRING}
Concept {id: STRING}
Organization {id: STRING}
Relationship properties:

The relationships:
(:Document)-[:MENTIONS]->(:Organization)
(:Concept)-[:DEVELOPED_BY]->(:Person)
```

Querying the graph

```
from langchain_community.chains.graph_qa.cypher import GraphCypherQAChain

chain = GraphCypherQAChain.from_llm(
    llm=ChatOpenAI(api_key="...", temperature=0), graph=graph, verbose=True
)

result = chain.invoke({"query": "What is the most accurate model?"})
```

[1] https://api.python.langchain.com/en/latest/chains/langchain_community.chains.graph_qa.cypher.GraphCypherQAChain.html

Querying the graph

```
print(f"Final answer: {result['result']}")
```

```
> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (m:Model)
RETURN m
ORDER BY m.accuracy DESC
LIMIT 1;
Full Context:
[{'m': {'id': 'Artificial Neural Networks'}}]

> Finished chain.

Final answer: Artificial Neural Networks
```

Customization

```
chain = GraphCypherQAChain.from_llm(
    llm=ChatOpenAI(api_key="...", temperature=0), graph=graph, verbose=True
)
```

- `qa_prompt` : Prompt template for result generation
- `cypher_prompt` : Prompt template for Cypher generation
- `cypher_llm` : LLM for Cypher generation
- `qa_llm` : LLM for result generation

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Improving graph retrieval

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Meri Nova
Machine Learning Engineer

datacamp

Techniques

Main limitation: reliability of user → Cypher translation

Strategies to improve graph retrieval system:

- Filtering Graph Schema
- Validating the Cypher Query
- Few-shot prompting

Filtering

```
from langchain_community.chains.graph_qa.cypher import GraphCypherQAChain

llm = ChatOpenAI(api_key="...", model="gpt-4o-mini", temperature=0)

chain = GraphCypherQAChain.from_llm(
    graph=graph, llm=llm, exclude_types=["Concept"], verbose=True
)
print(graph.get_schema)
```

```
Node properties:
Document {title: STRING, id: STRING, text: STRING, summary: STRING, source: STRING}
Organization {id: STRING}
```

Validating the Cypher query

- Difficulty in interpreting the *direction* of relationships

```
chain = GraphCypherQAChain.from_llm(
    graph=graph, llm=llm, verbose=True, validate_cypher=True
)
```

1. Detects nodes and relationships
2. Determines the directions of the relationship
3. Checks the graph schema
4. Update the direction of relationships

Few-shot prompting

```
examples = [
    {
        "question": "How many notable large language models are mentioned in the article?",
        "query": "MATCH (m:Concept {id: 'Large Language Model'}) RETURN count(DISTINCT m)",
    },
    {
        "question": "Which companies or organizations have developed the large language models mentioned?",
        "query": "MATCH (o:Organization)-[:DEVELOPS]->(m:Concept {id: 'Large Language Model'}) RETURN DISTINCT o.id",
    },
    {
        "question": "What is the largest model size mentioned in the article, in terms of number of parameters?",
        "query": "MATCH (m:Concept {id: 'Large Language Model'}) RETURN max(m.parameters) AS largest_model",
    },
]
```

Implementing few-shot prompting

```
from langchain_core.prompts import FewShotPromptTemplate, PromptTemplate

example_prompt = PromptTemplate.from_template(
    "User input: {question}\nCypher query: {query}"
)

cypher_prompt = FewShotPromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
    prefix="You are a Neo4j expert. Given an input question, create a syntactically correct
    Cypher query to run.\n\nHere is the schema information\n{schema}.\n\n
    Below are a number of examples of questions and their corresponding Cypher queries.",
    suffix="User input: {question}\nCypher query: ",
    input_variables=["question"],
)
```

Complete prompt

```
You are a Neo4j expert. Given an input question, create a syntactically correct Cypher query to run.

Below are a number of examples of questions and their corresponding Cypher queries.

User input: How many notable large language models are mentioned in the article?
Cypher query: MATCH (p:Paper) RETURN count(DISTINCT p)

User input: Which companies or organizations have developed the large language models?
Cypher query: MATCH (o:Organization)-[:DEVELOPS]->(m:Concept {id: 'Large Language Model'}) RETURN DISTINCT o.id

User input: What is the largest model size mentioned in the article, in terms of number of parameters?
Cypher query: MATCH (m:Concept {id: 'Large Language Model'}) RETURN max(m.parameters) AS largest_model

User input: How many papers were published in 2016?
Cypher query:
```

Adding the few-shot examples

```
chain = GraphCypherQAChain.from_llm(
    graph=graph, llm=llm, cypher_prompt=cypher_prompt,
    verbose=True, validate_cypher=True
)
```

Let's practice!

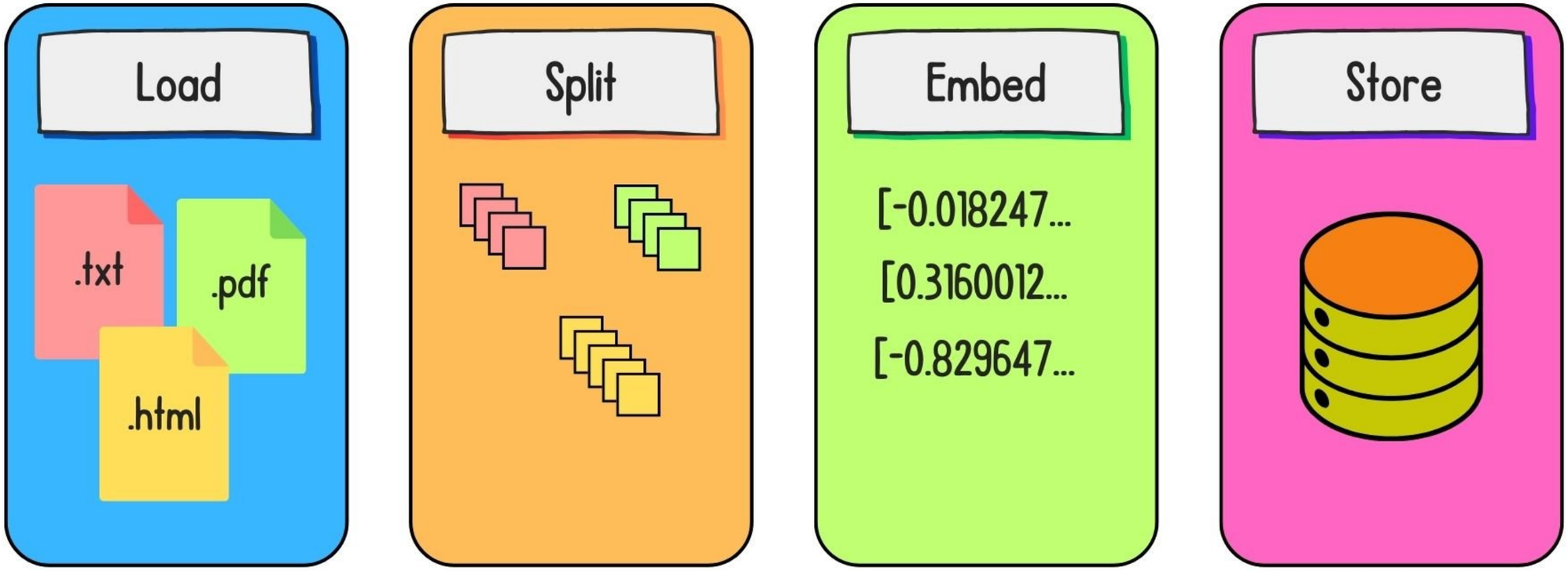
RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Congratulations!

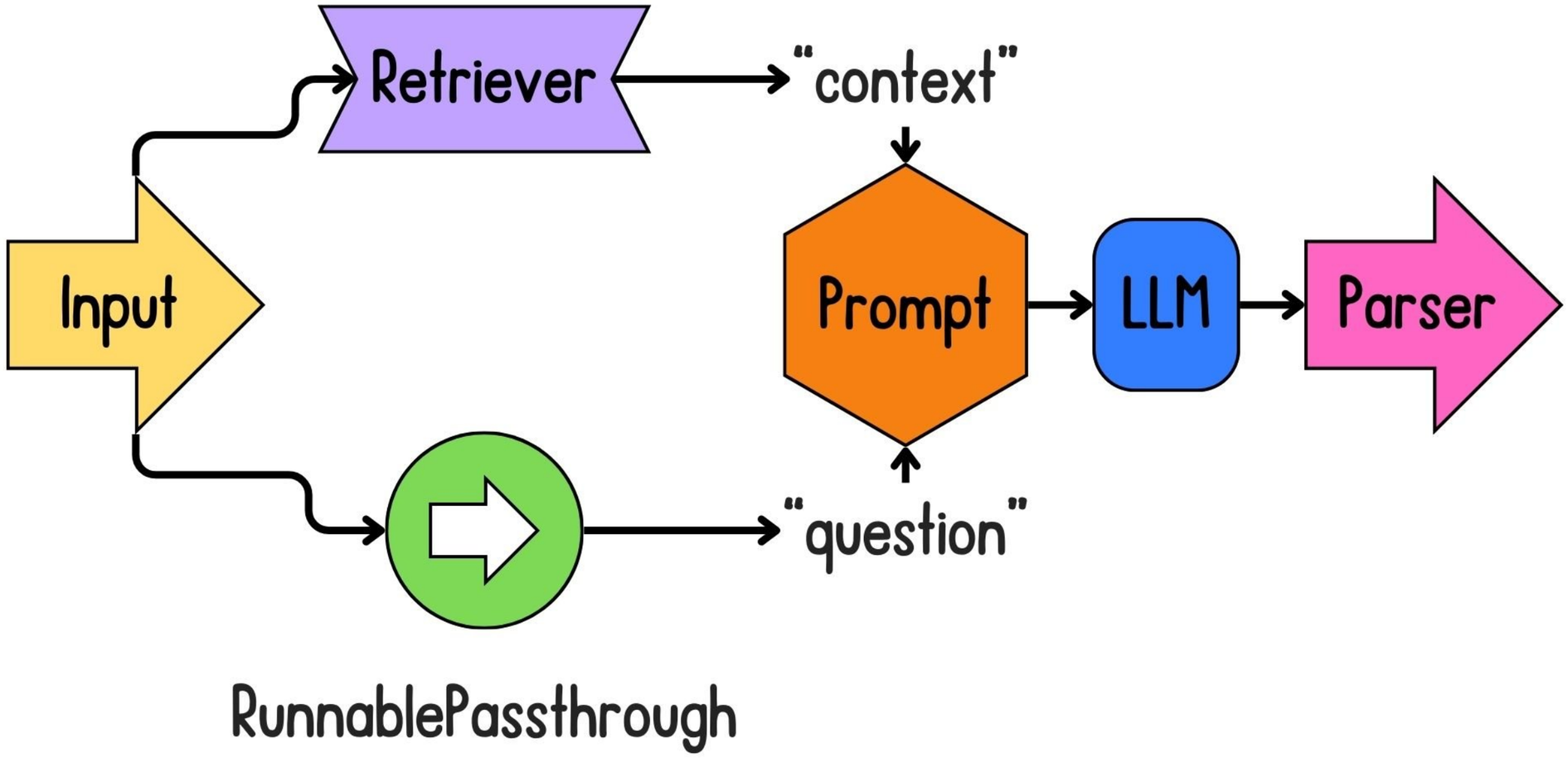
RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Meri Nova
Machine Learning Engineer

Chapter 1


Load
.txt
.pdf
.html
Split
Embed
[-0.018247...
[0.3160012...
[-0.829647...
Store

Chapter 1


Retriever
"context"
Input
Prompt
LLM
Parser
"question"
RunnablePassthrough

Chapter 2

Markdown files

- `UnstructuredMarkdownLoader`

Python files

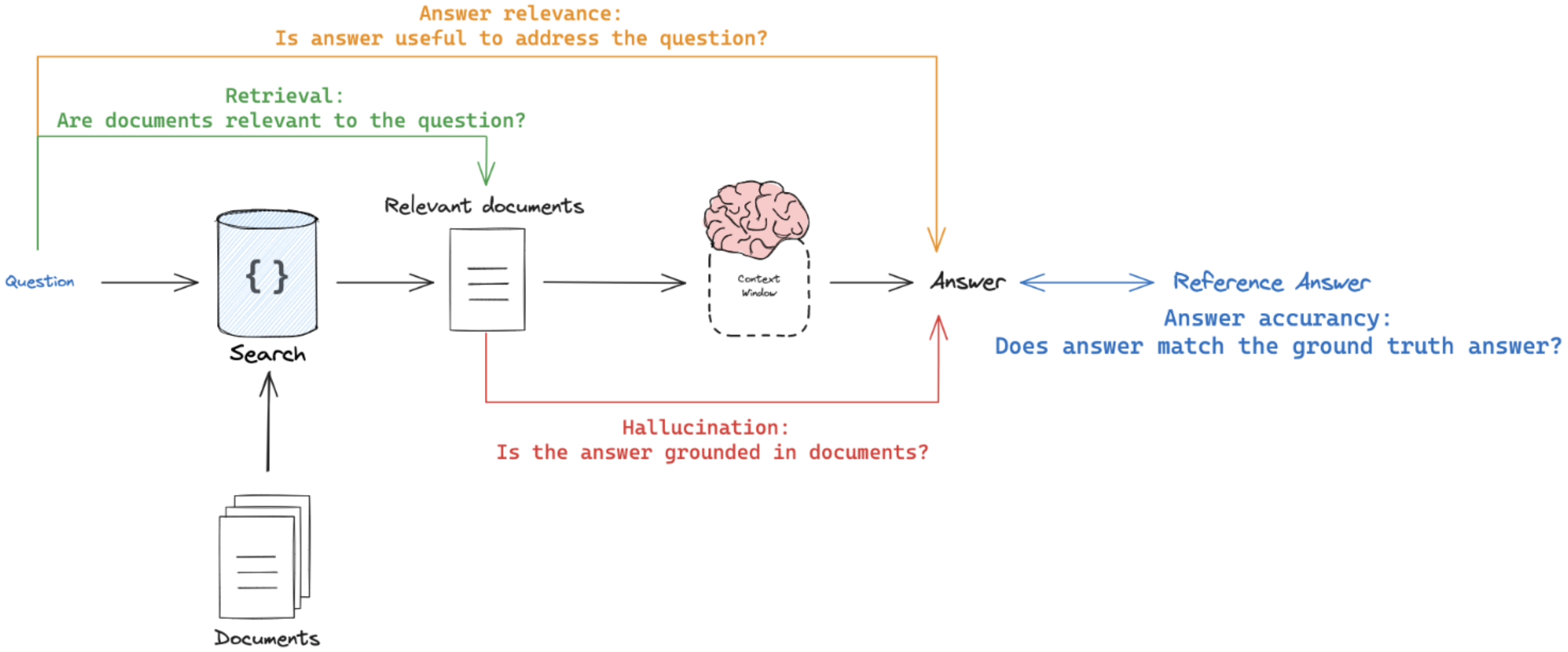
- `PythonLoader`
- `language=Language.PYTHON`

Token splitting → `TokenTextSplitter`

Semantic splitting → `SemanticChunker`

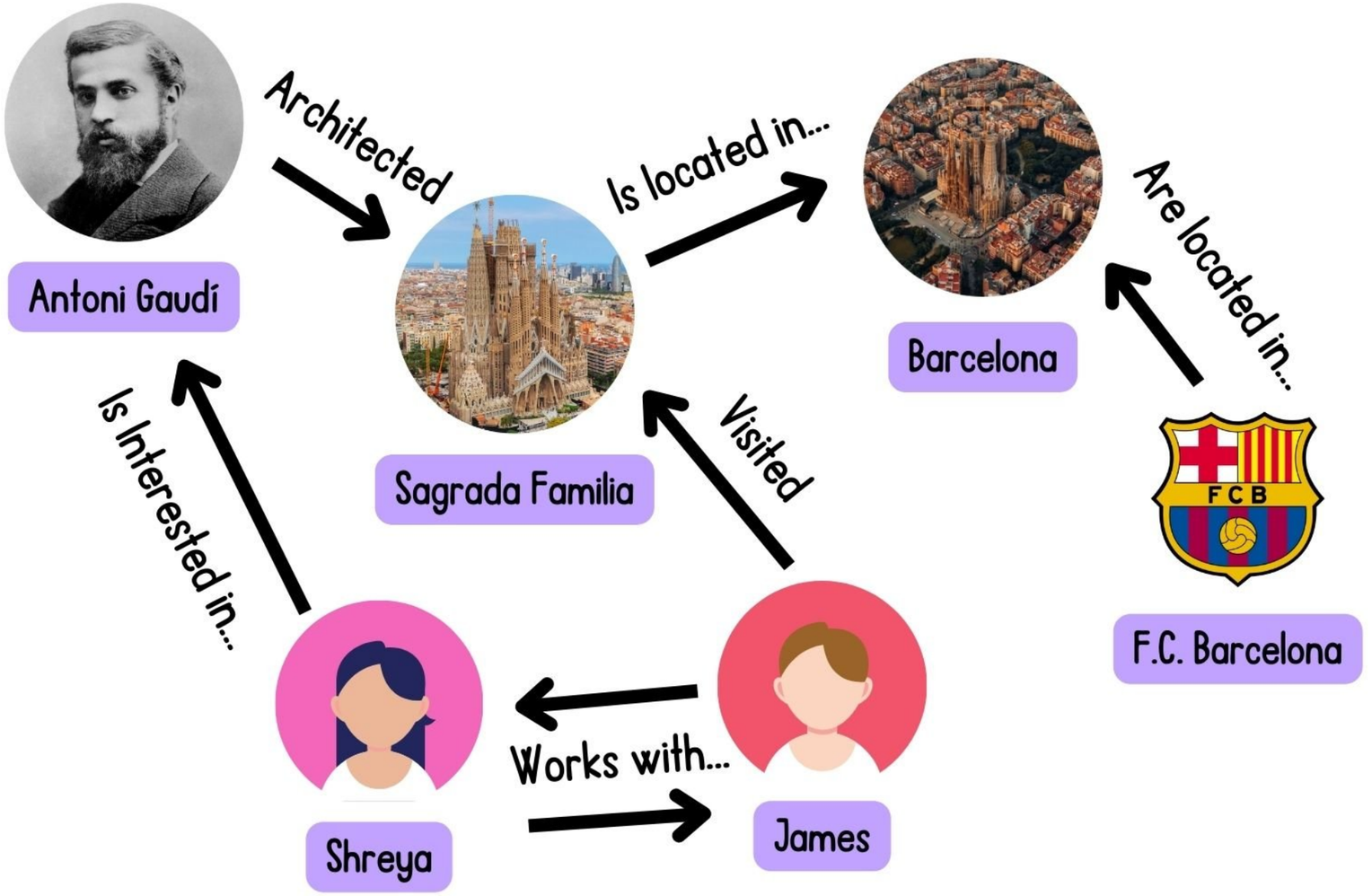
RAG applications has numerous use cases, but are most frequently deployed in chatbots. Domestic dogs are descendants from an extinct population of Pleistocene wolves over 14,000 years ago.

Chapter 2

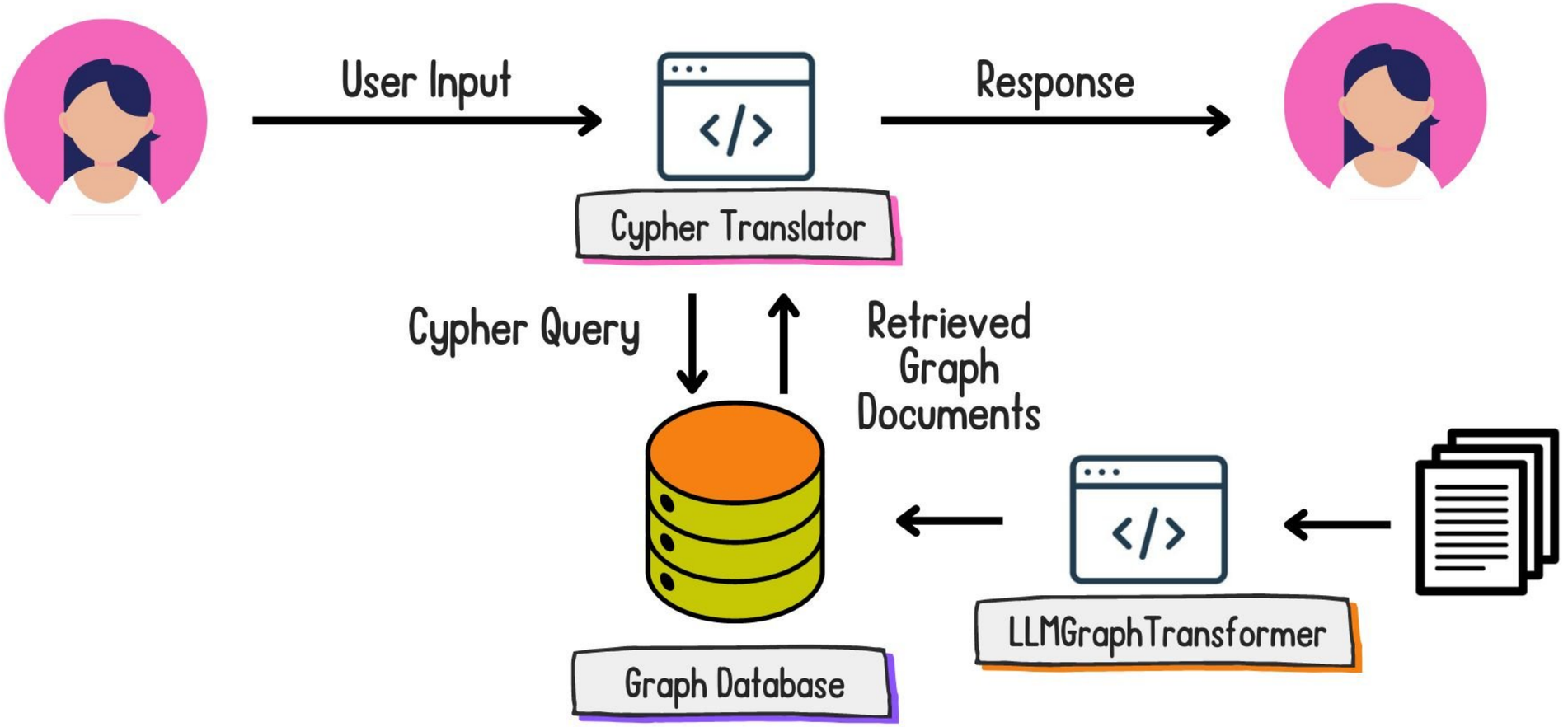


[1] Image Credit: LangSmith

Chapter 3


Antoni Gaudí
Architected
Sagrada Familia
Is located in...
Barcelona
Are located in...
F.C. Barcelona
Is Interested in...
Shreya
Visited
Works with...
James

Chapter 3


User Input
Response
Cypher Translator
Cypher Query
Retrieved Graph Documents
Graph Database
LLMGraphTransformer

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN