HOW TO BUILD A CHATBOT

Session 3 - Retrieval Augmented Generation

SESSION 3
AGENDA

1. Introduction to Retrieval Augmented Generation (RAG)
2. Vectors Embeddings
3. Vector Databases
3. Retrieval (Vector Search)

INTRODUCTION TO RAG

What?

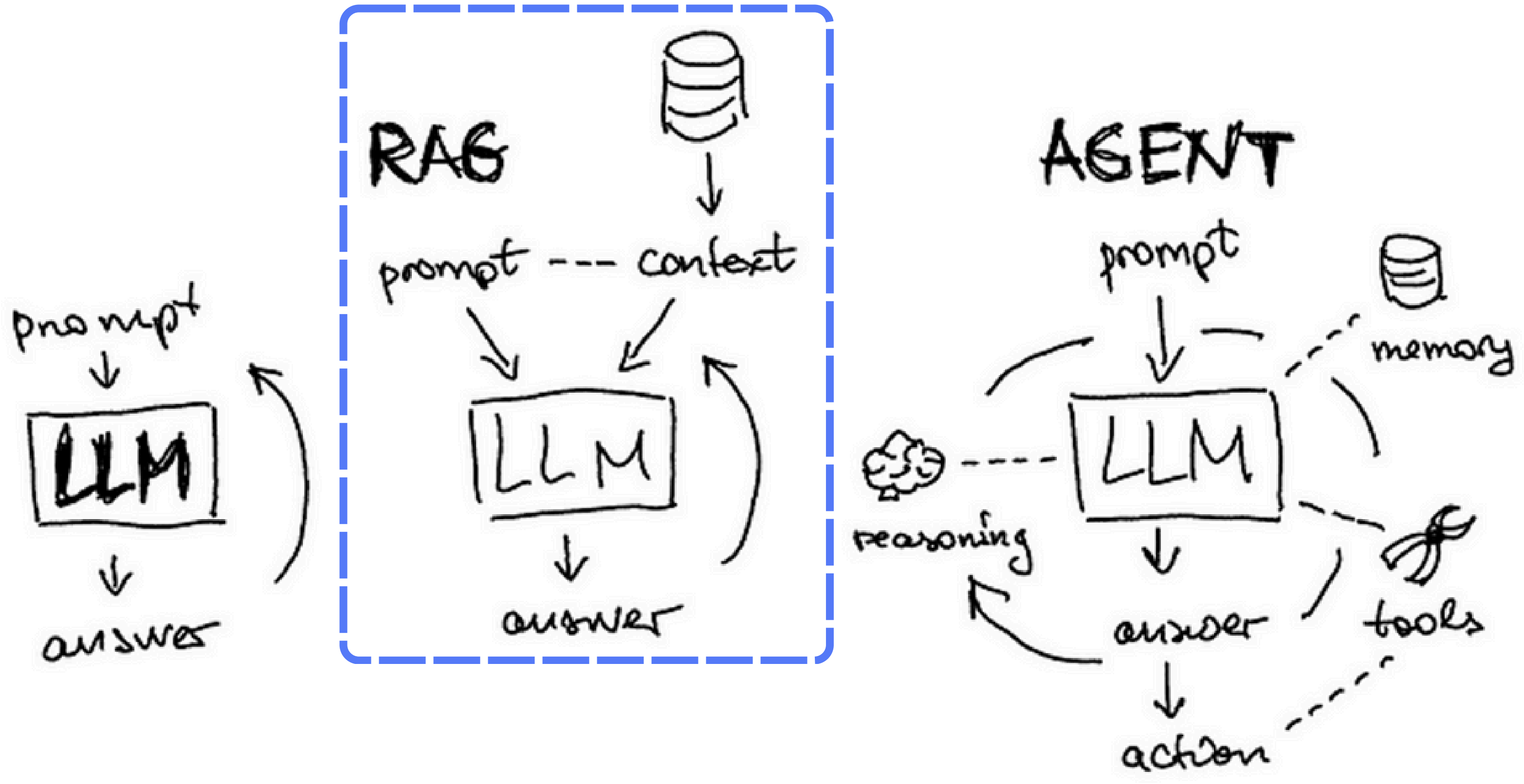
Enhances text generation by retrieving and adding external knowledge, ensuring factual output beyond LLM's internal data.

Why?

Prevents hallucinations and overcomes limited domain knowledge by incorporating relevant external information.

How?

Dynamically injects relevant external data into prompts as context.



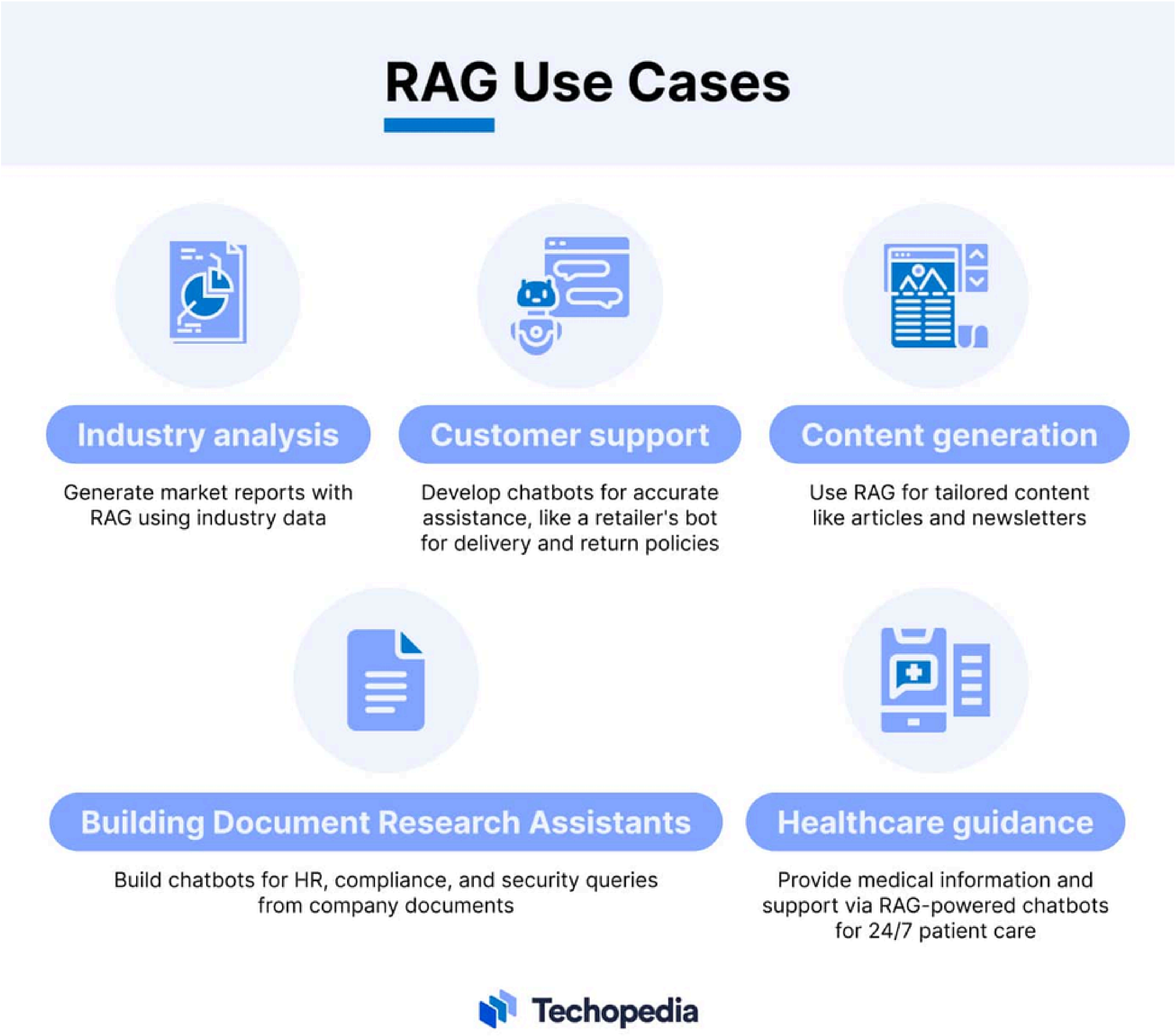
[1]

INTRODUCTION TO RAG

Example Customer Support:

Businesses can use RAG to create customer support chatbots that provide users with access to more accurate and reliable domain specific information.

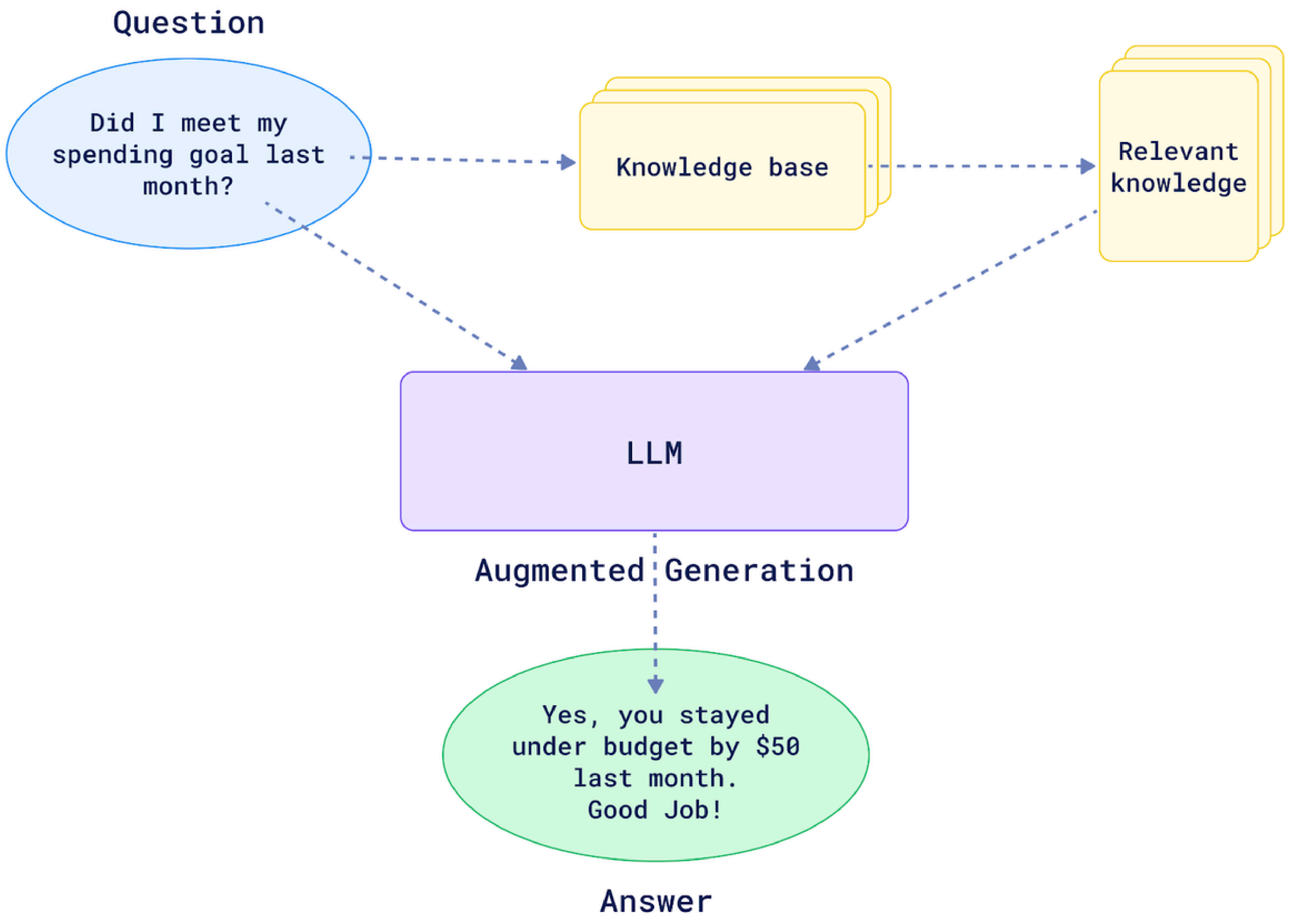
E.g.: A retailer could develop a chatbot that's prepared to answer user questions about specific products.



[2]

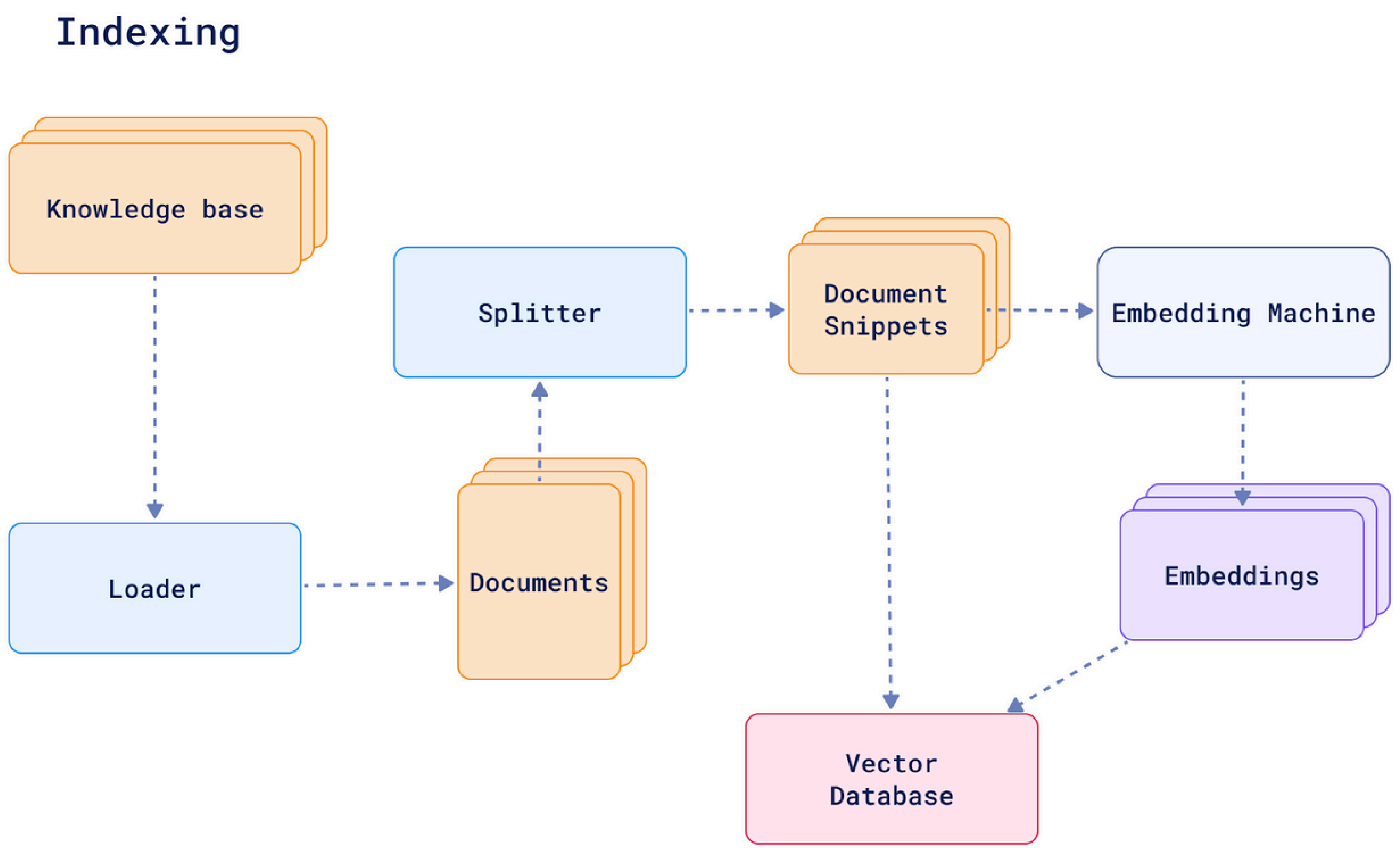
INTRODUCTION TO RAG

General workflow: Dynamic Prompt Injection



INTRODUCTION TO RAG

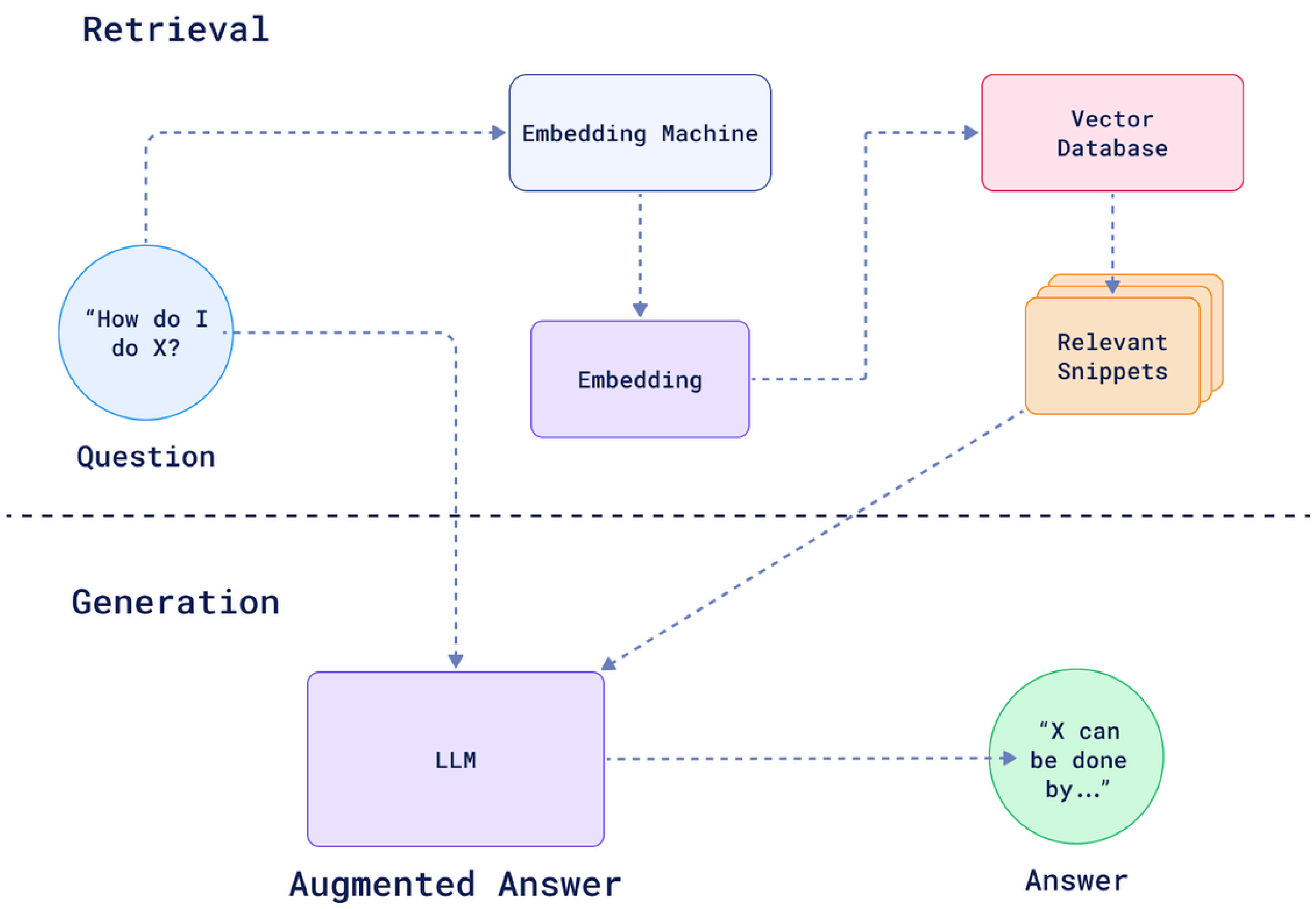
Step 1: Store vector embeddings into vector databases as knowledge



[3]

INTRODUCTION TO RAG

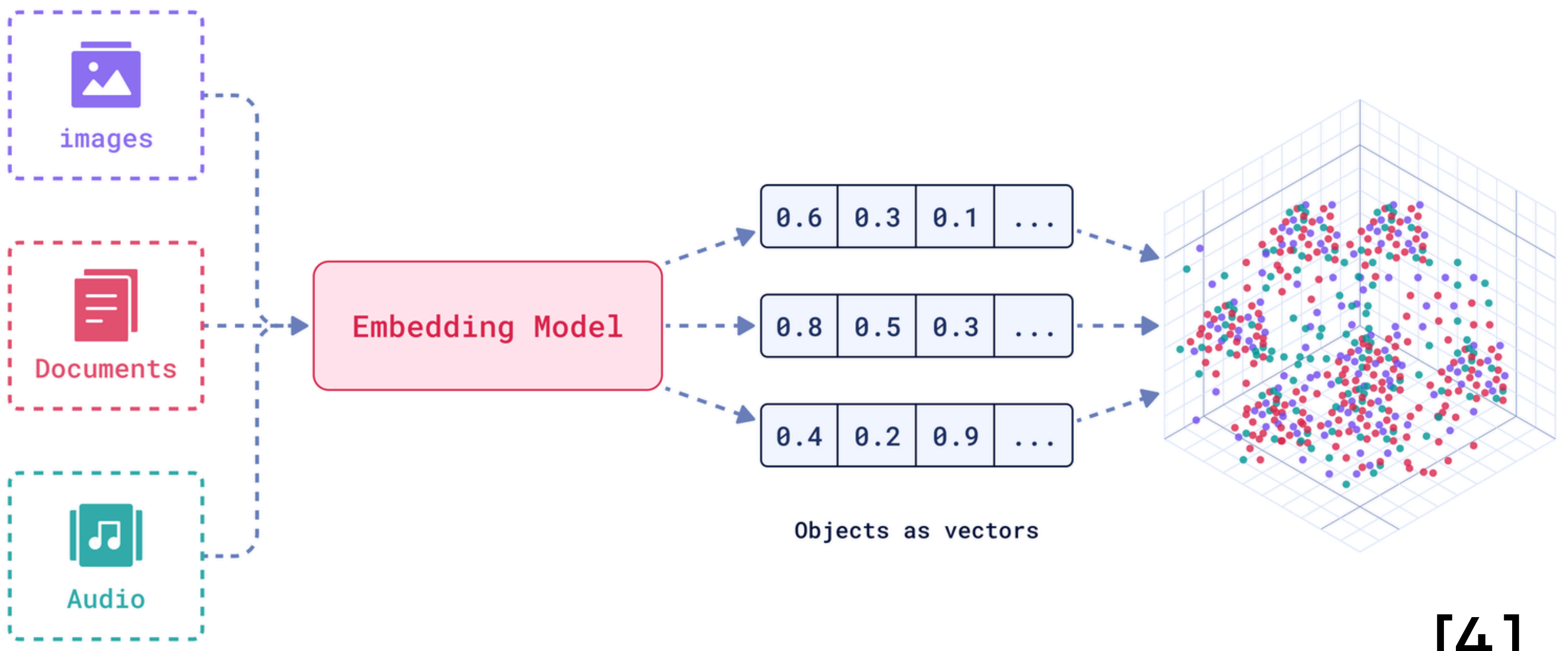
Step 2: Information retrieval and answer generation



[3]

VECTOR EMBEDDINGS

- An embedding is a numerical representation of data in vector format.
- Today embeddings are usually generated by transformer-based models.
- These models are trained on large text datasets to learn concepts and relationships from languages.
- Transformers can capture the contextual meaning, semantics and order of words.
- Embeddings enable mathematical operations and can be inputs for other ML models.



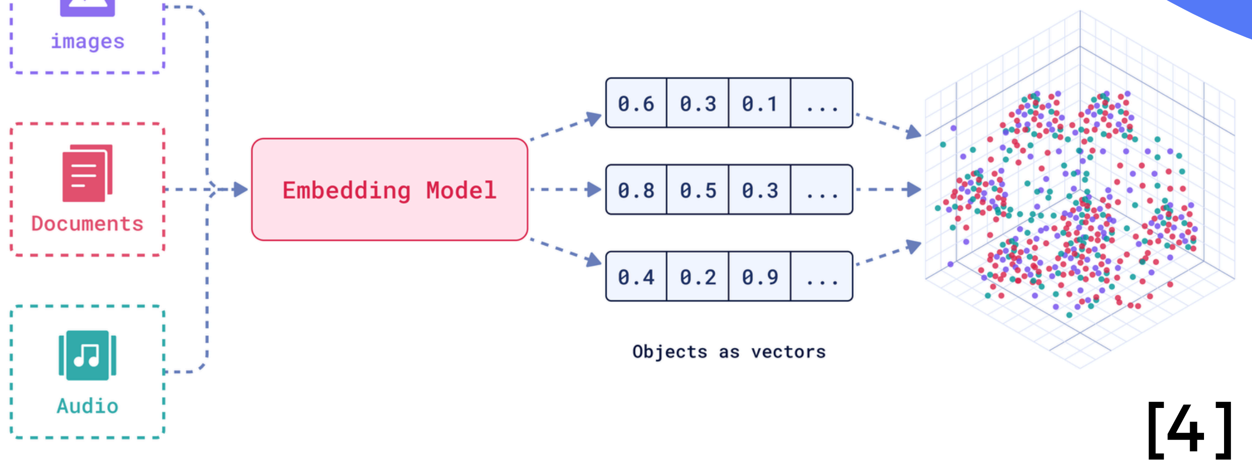
[4]

VECTOR EMBEDDINGS

Generate embedding vectors with LangChain

1. Load Embedding Model

```
from langchain_huggingface import HuggingFaceEmbeddings

embeddings_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-mpnet-base-v2")
```



[4]

2. Generate Embeddings

embed_documents

Embed list of texts

Use `.embed_documents` to embed a list of strings, recovering a list of embeddings:

```
embeddings = embeddings_model.embed_documents(
    [
        "Hi there!",
        "Oh, hello!",
        "What's your name?",
        "My friends call me World",
        "Hello World!"
    ]
)
len(embeddings), len(embeddings[0])
```

```
(5, 1536)
```

embed_query

Embed single query

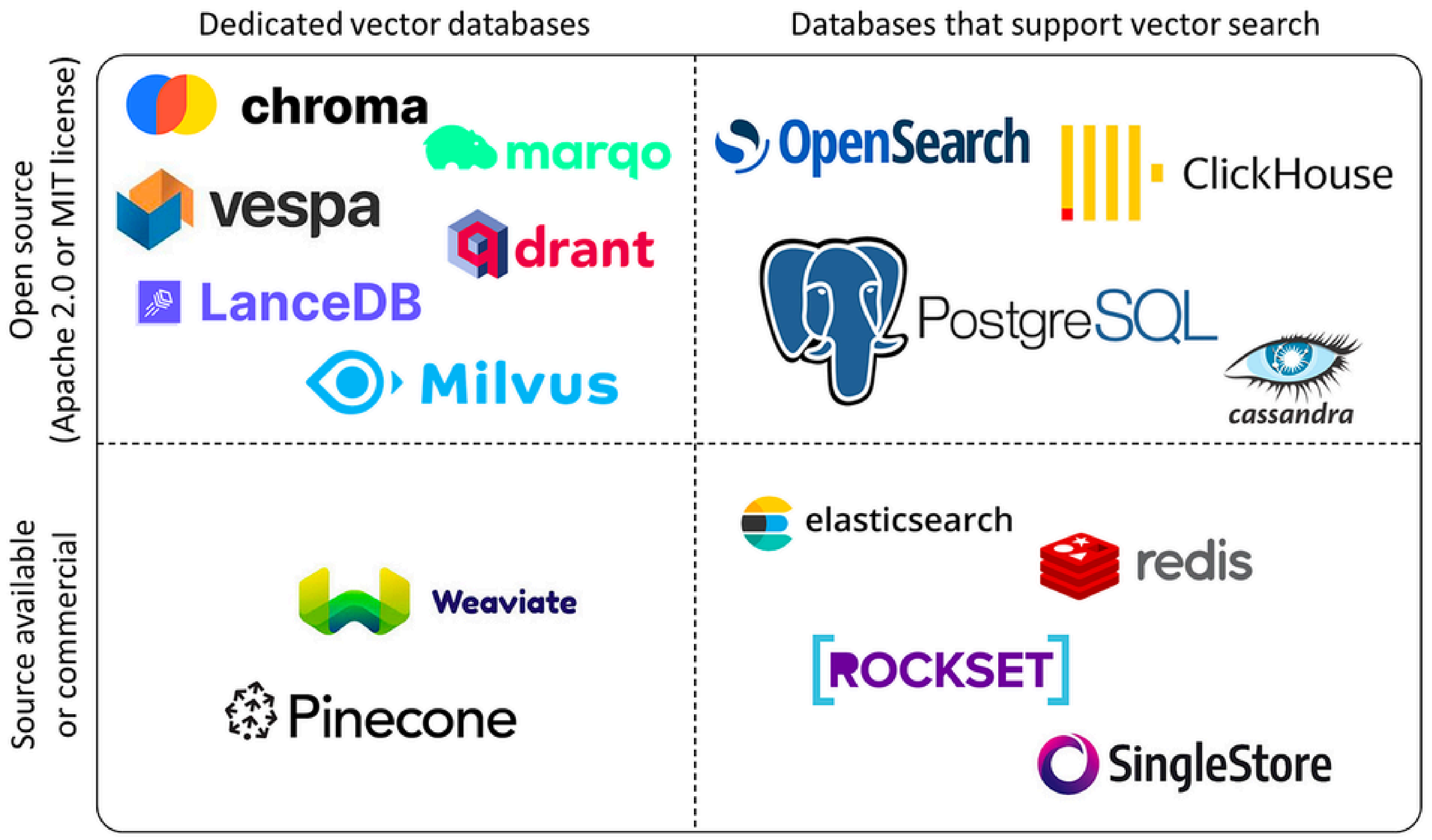
Use `.embed_query` to embed a single piece of text (e.g., for the purpose of comparing to other embedded pieces of texts).

```
embedded_query = embeddings_model.embed_query("What was the name mentioned in the conversation?")
embedded_query[:5]
```

```
[0.0053587136790156364,
 -0.0004999046213924885,
 0.038883671164512634,
 -0.0030010775662958623,
 -0.009008182212710383]
```

VECTOR DATABASES

- Vector databases refer to the mechanism for storing and retrieving vector embeddings.
- Database solutions specifically designed for efficient storage and retrieval of vectors.
- There many different solutions on the market (commercial, open source, ...)

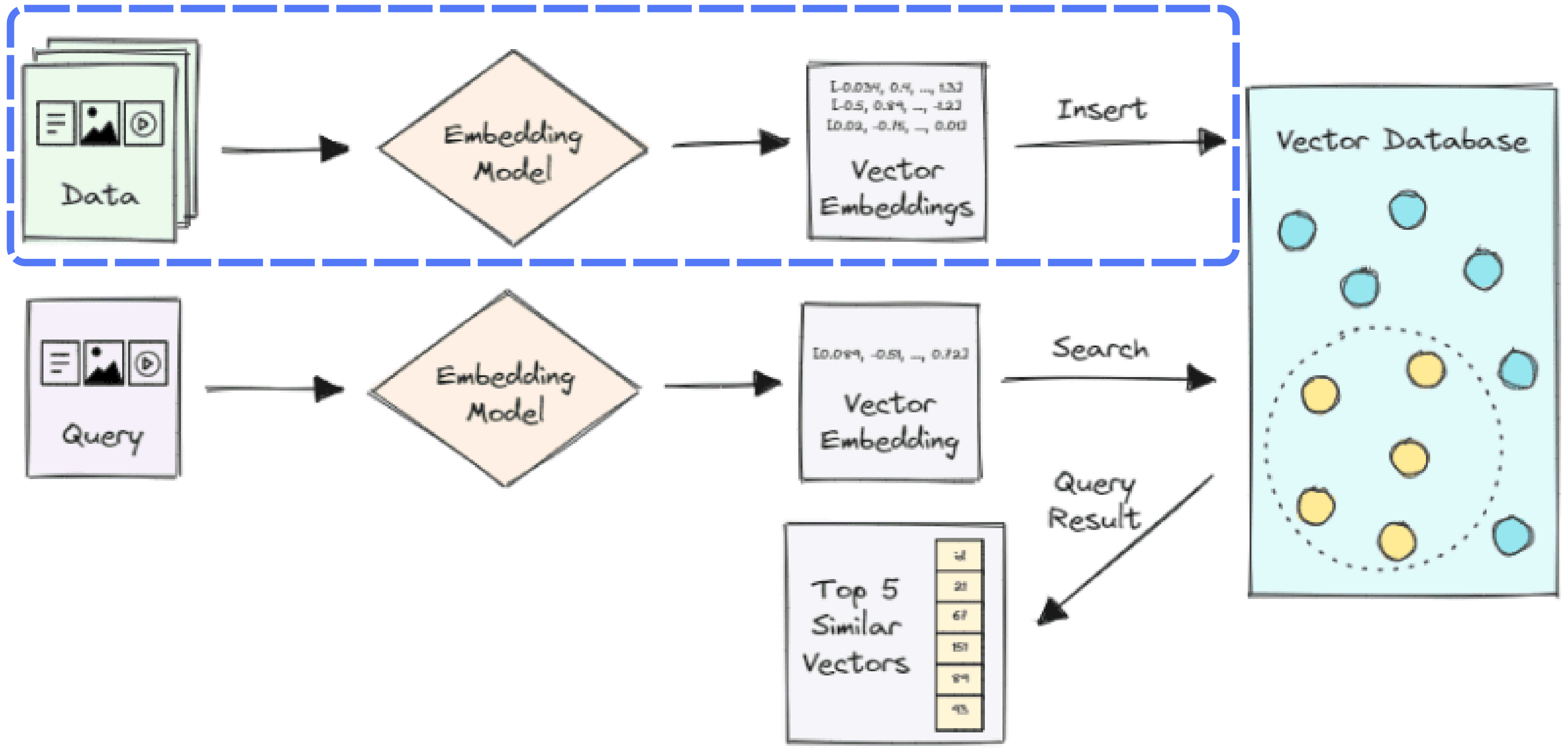


[5]

VECTOR DATABASES

General data integration workflow:

1. Collect and load unstructured data (e.g., PDFs).
2. Split data into smaller, topic-specific chunks (model input size limitations; improve search accuracy).
3. Generate embedding vectors for chunks using a embedding model.
4. Store vector embeddings and original text in the database index.



[6]

VECTOR DATABASES

Data integration workflow with LangChain

1. Load and split data in chunks

```
from langchain_text_splitters import RecursiveCharacterTextSplitter

# Load example document
with open("state_of_the_union.txt") as f:
    state_of_the_union = f.read()

text_splitter = RecursiveCharacterTextSplitter(
    # Set a really small chunk size, just to show.
    chunk_size=100,
    chunk_overlap=20,
    length_function=len,
    is_separator_regex=False,
)
texts = text_splitter.create_documents([state_of_the_union])
```

2. Load Embedding Model

```
from langchain_huggingface import HuggingFaceEmbeddings

embeddings_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-mpnet-base-v2")
```

3. Define vector database and pass embedding model

```
from langchain_chroma import Chroma

vector_store = Chroma(
    collection_name="example_collection",
    embedding_function=embeddings,
    persist_directory="./chroma_langchain_db",
)
```

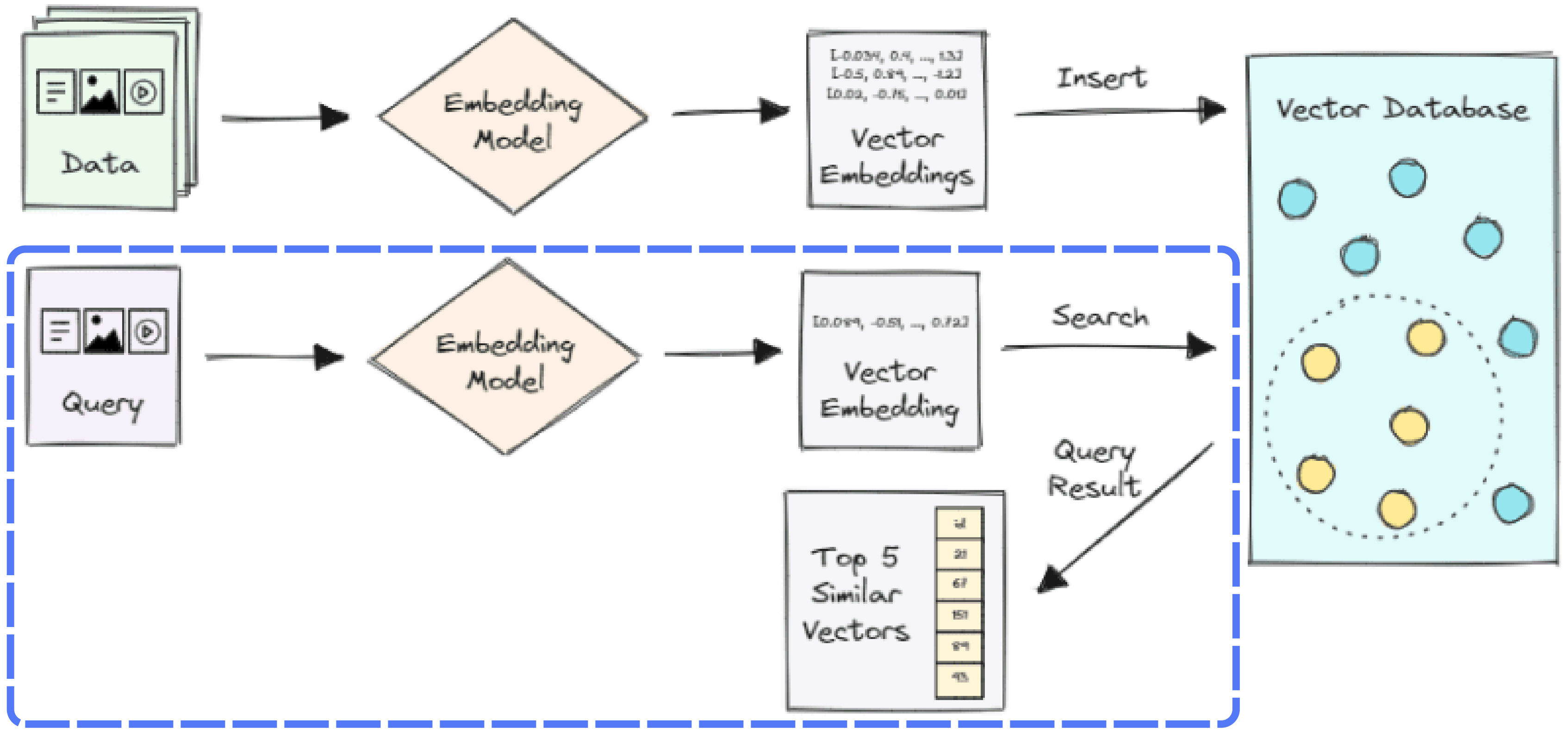
3. Generate embedding vectors from document chunks and store them into vector database index.

```
document_8 = Document(
    page_content="LangGraph is the best framework for building stateful, agentic applications!",
    metadata={"source": "tweet"},
    id=8,
)

document_9 = Document(
    page_content="The stock market is down 500 points today due to fears of a recession.",
    metadata={"source": "news"},
    id=9,
)

document_10 = Document(
    page_content="I have a bad feeling I am going to get deleted :(",
    metadata={"source": "tweet"},
    id=10,
)

documents = [
    document_1,
    document_2,
    document_3,
    document_4,
    document_5,
    document_6,
    document_7,
    document_8,
    document_9,
    document_10,
]
uuids = [str(uuid4()) for _ in range(len(documents))]

vector_store.add_documents(documents=documents, ids=uuids)
```

RETRIEVAL (VECTOR SEARCH)

General Retrieval Workflow:

1. Generate dynamically a embedding vector from the query
2. Distances between vectors are computed using metrics like cosine similarity or euclidean distance.
3. Based on the distances query vectors are compared to all vectors in the collection; smaller distances indicate higher similarity.
4. Retrieve the k-most similar documents from the vector database index.



[6]

RETRIEVAL (VECTOR SEARCH)

Perform vector search with LangChain

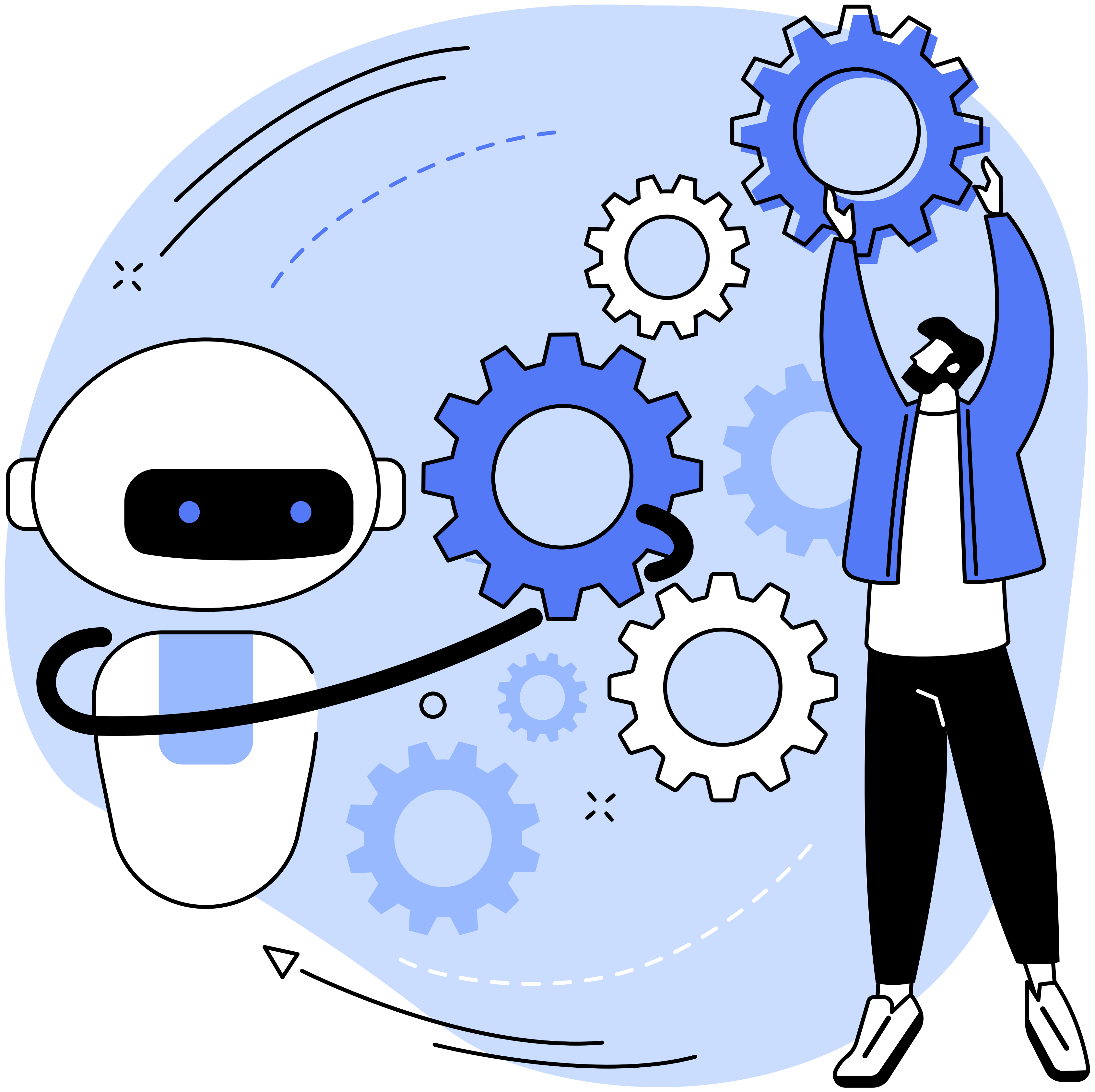
1.Similarity search

1. Generate the query's embedding vector.
2. Compute similarities using a distance metric.
3. Retrieve documents with the smallest distances.
4. Return the top-k documents.

```
results = vector_store.similarity_search(
    "LangChain provides abstractions to make working with LLMs easy",
    k=2,
    filter={"source": "tweet"},
)
for res in results:
    print(f"* {res.page_content} [{res.metadata}]")
```

2. Retrieve the k=2 most similar documents from the vector database

```
* Building an exciting new project with LangChain - come check it out! [{'source': 'tweet'}]
* LangGraph is the best framework for building stateful, agentic applications! [{'source': 'tweet'}]
```



IT’S YOUR TURN

Sources:

[1]: https://towardsdatascience.com/intro-to-llm-agents-with-langchain-when-rag-is-not-enough-7d8c08145834

[2]: https://www.techopedia.com/definition/rag

[3]: https://qdrant.tech/articles/what-is-rag-in-ai/

[4]: https://qdrant.tech/articles/what-are-embeddings/

[5]: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.datacamp.com%2Fblog%2Fthe-top-5-vector-databases&psig=AOvVaw2zepQAXBLv8MosrJV9f0nv&ust=1726393401450000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCKia-92SwogDFQAAAAAdAAAAABAE

[6]: https://medium.com/@vipra_singh/building-llm-applications-retrieval-search-part-5-c83a7004037d