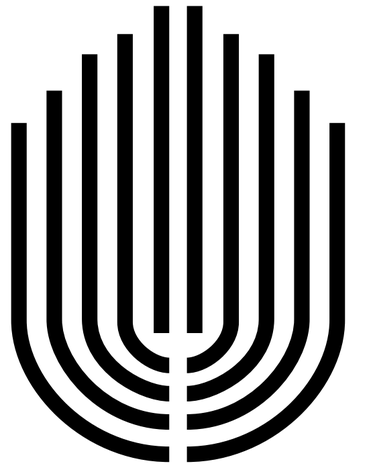
RAG
(RETRIEVAL-AUGMENTED GENERATION)

By
Dinesh Chopra
&
Yogesh Bhatt



20th Sep, 2025

Recap

- Introduction to Generative AI
- Generative AI Evolution
- Foundation Models (Text Generation, Image Generation)
- Generative AI Applications
- Prompt Engineering
  - Few Shot Learning
  - Chain of thoughts
- Best Practice of Prompt Engineering
- Fine-Tuning and Customization
- Generative AI Ethics
- Hands-on Project


Q
A

Limitations of Prompt Engineering

- **Knowledge Cutoff**
  - LLMs don’t know recent events or your private company data.
- **Token Limit**
  - You can’t paste a 200-page insurance policy or SAP manual into a single prompt.
- **Accuracy Issues**
  - Even with good prompts, LLMs may hallucinate (make up answers).
- **Domain Relevance**
  - Generic models may not use the exact policy terms, compliance rules, or invoice formats your business cares about.

Limitations of Prompt Engineering

Example

- **Scenario: Insurance Policy QnA**
  - **User Question**
    - Does Policy P123 cover fire damage?
  - **Without RAG**
    - Model guesses or says "*I don't know.*"
  - **Why?**
    - Policy P123 text isn't inside its training data.

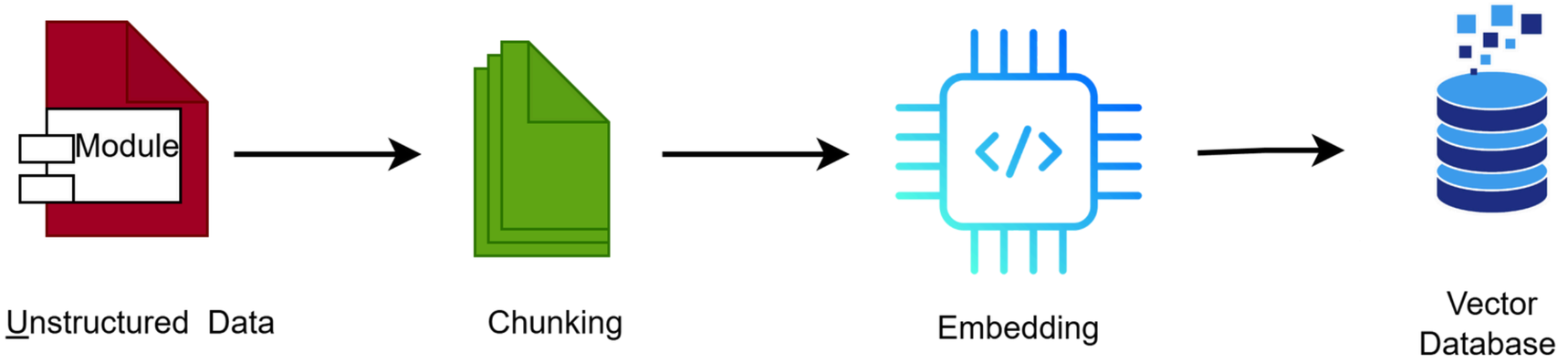
RETRIEVAL-AUGMENTED GENERATION (RAG)

- **Retrieve** → Search company knowledge base for relevant policy / invoice / circular.
- **Augment** → Add retrieved text as context for the LLM.
- **Generate** → LLM answers grounded in actual documents.

- **Benifits:**
  - Accurate
  - Up-to-date
  - Domain Specific answers with citations

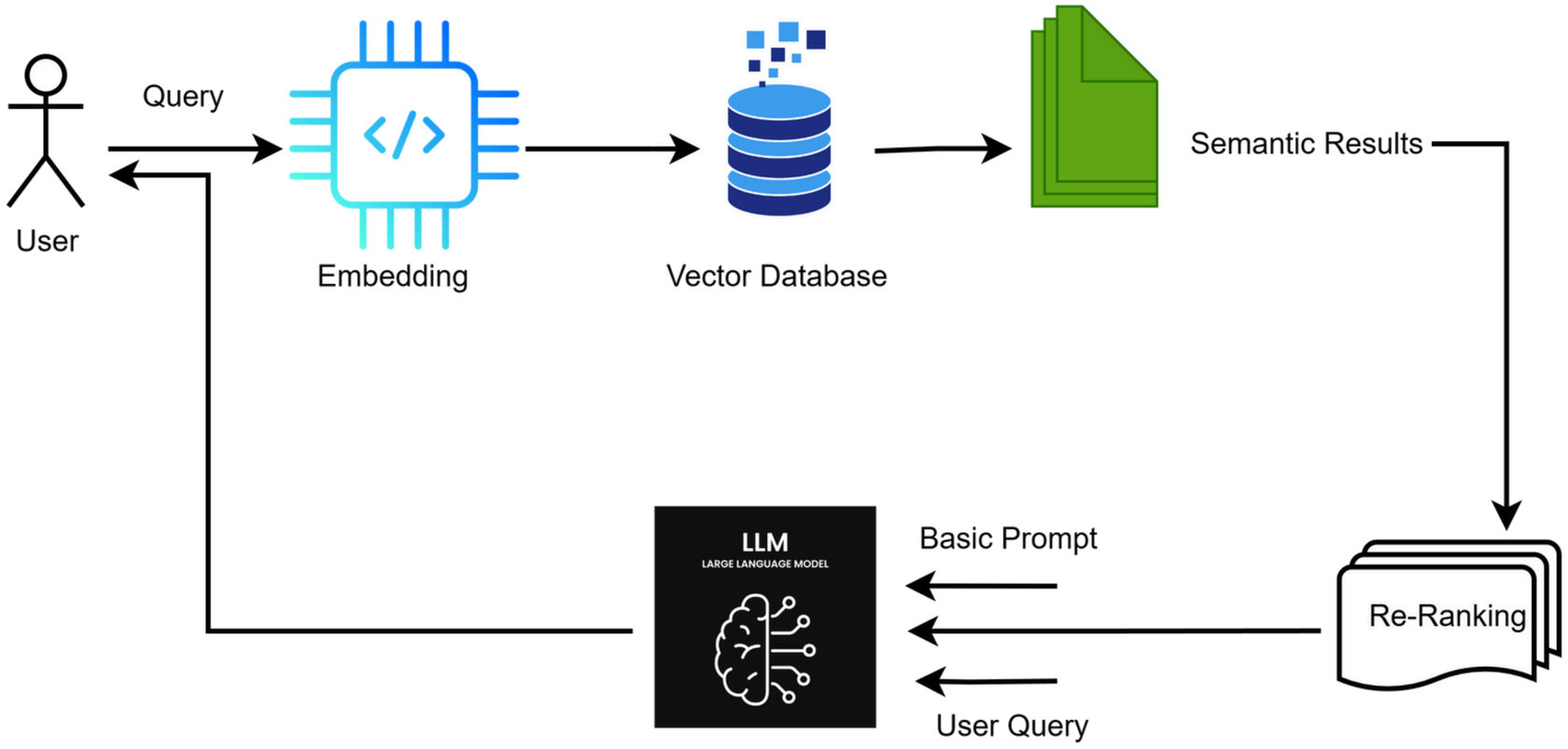
RETRIEVAL-AUGMENTED GENERATION (RAG)

Data Ingestion



RETRIEVAL-AUGMENTED GENERATION (RAG)

QnA / Response Generation

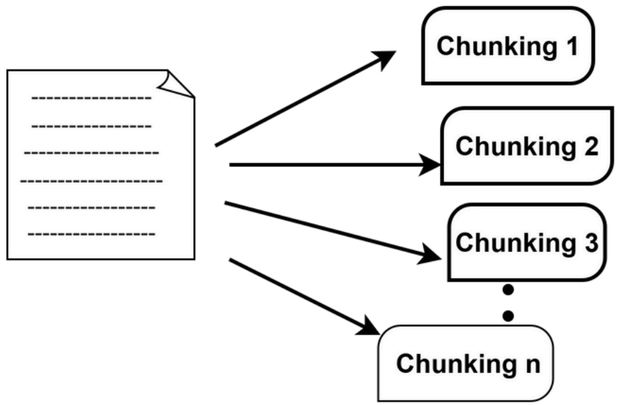


CORE COMPONENTS

- Chunking
- Embedding
- Document Ingestion in Vector Database
- Semantic Search
- Re-ranking (Optional)
- LLM + Prompt Template
- Response

Chunking

- **Chunking:** Splitting documents into smaller pieces before indexing in a vector DB.



- **Why:** LLMs can’t handle huge docs. Smaller chunks make retrieval more accurate and context manageable.

Chunking

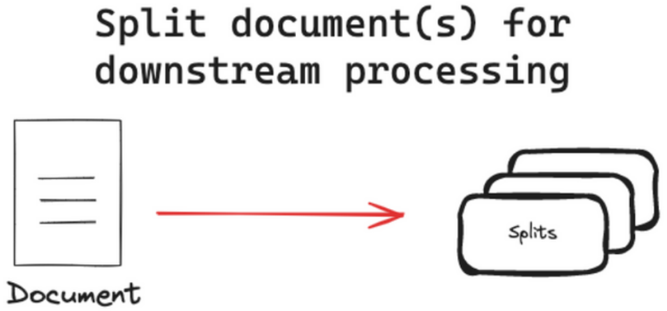
```python
from langchain_community.document_loaders import PyPDFLoader

# Load PDF
file_path = "/DOC-20250820-WA0001..pdf"
loader = PyPDFLoader(file_path)

pages = loader.load()
print(f"Total pages: {len(pages)}\n")
```

```
Total pages: 16
```

```python
print(pages[0].page_content)
```

```
PUBLISHED FROM: DELHI LUCKNOW BHOPAL RAIPUR BHUBANESWAR RANCHI CHANDIGARH DEHRADUN HYDERABAD VIJAYAWADA
LUCKNOW | WEDNESDAY AUGUST 20, 2025 | PAGES 16 | `3.00
@TheDailyPioneerfacebook.com/dailypioneer
www.dailypioneer.com
instagram.com/dailypioneer linkedin.com/in/dailypioneer
RAJESH KUMAR
n New Delhi
In a move to check rampant
online fraud, the Union
Cabinet on Tuesday
approved the much-awaited
Online Gaming Bill, which
seeks to regulate the misuse
of online gaming apps and
similar platforms. Titled ‘The
Promotion and Regulation of
Online Gaming Bill, 2025’,
the proposal law aims to
regulate the sector and curb
illegal betting. The Bill is
likely to be tabled in
Parliament on Wednesday
```

Text splitters



- **RecursiveCharacterTextSplitter** → Splits text naturally by paragraphs, sentences, and words while keeping chunks within size limits.
- **TokenTextSplitter** → Splits text based on tokens so chunks always fit within the LLM's context window.
- **CharacterTextSplitter** → Splits text into fixed-size character chunks without considering meaning.

Chunking With Langchain

```
from langchain_community.document_loaders import PyPDFLoader

file_path = "./example_data/layout-parser-paper.pdf"
loader = PyPDFLoader(file_path)

docs = loader.load()
```

```
from langchain_text_splitters import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(
    # Set a really small chunk size, just to show.
    chunk_size=100,
    chunk_overlap=20,
)
texts = text_splitter.split_documents(docs)
```

```
print(f"Total chunks: {len(texts)}\n")
```


Q
A

ASSIGNMENTS FOR CHUNKING

Task:

- **Take any PDF document (e.g., article, book chapter, company manual) and split it into meaningful chunks.**

Instructions:

- **Extract text from the PDF.**
- **Split the page into chunks.**
- **Print a few chunks to verify they are meaningful and preserve context.**

CHUNKING

Q.1. Which of these is a good real-world example of chunking?

a) Splitting video files for upload

b) Moving a file between folders

c) Dividing a book into chapters

d) Translating text into another language

CHUNKING

Q.2. Why do we cut big text into smaller chunks in RAG?

a) Save storage space on disk

b) Make each piece easier for models to handle

c) Help people read the document faster

d) Remove repeated words across the doc

CHUNKING

Q.3. Why is overlap sometimes added between chunks?

a) Preserve context at the boundaries

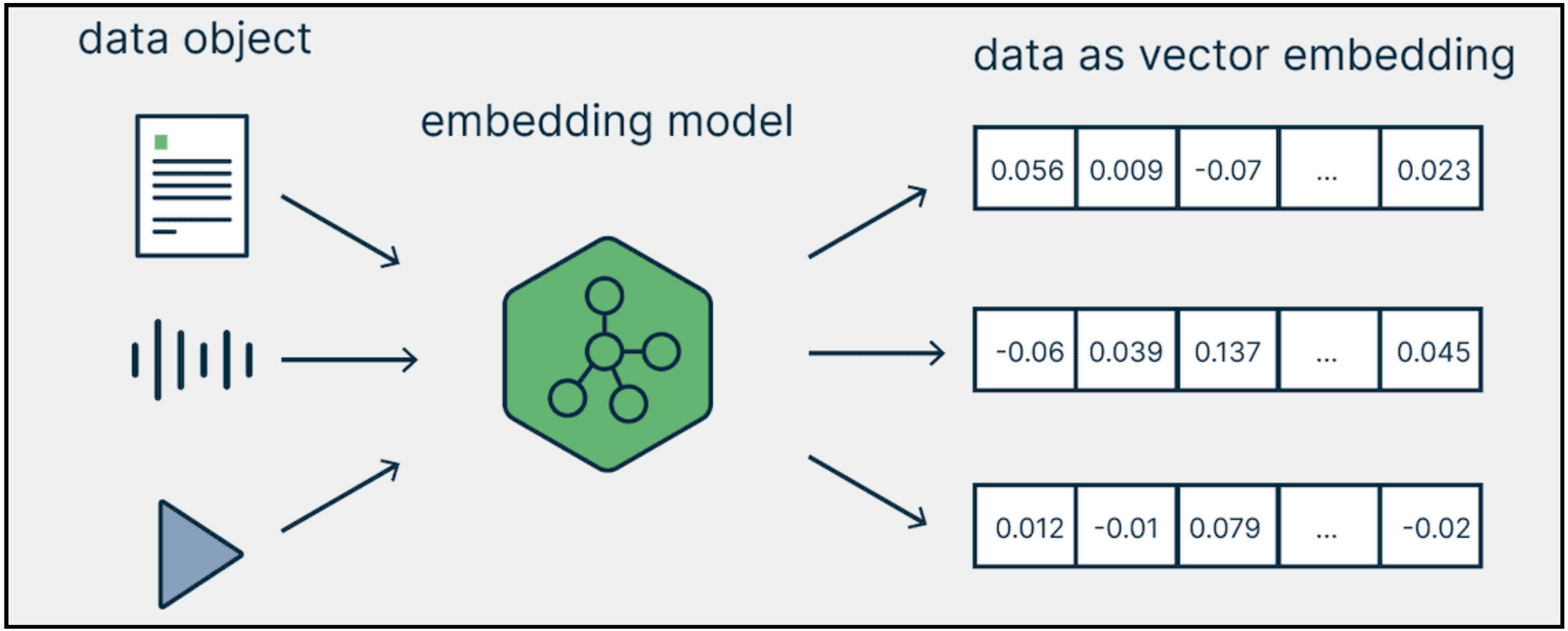
b) Shrink embedding dimensions

c) Remove duplicate phrases

d) Make chunks shorter for speed

Embedding

An embedding is a vector (list of numbers) that represents text, images, audio, or other data in a way that preserves semantic meaning.



Sources- https://weaviate.io/blog/how-to-choose-an-embedding-model

Embedding Models

Dimension	Model	Provider / Link	Notes
512	**EmbeddingGemma-300M (MRL)**	HuggingFace	Mid-range setting.
768	**all-mpnet-base-v2**	HuggingFace	Strong general semantic embeddings, widely used.
	EmbeddingGemma-300M (full)	HuggingFace	Native dimension.
1024	**Cohere embed-english-v3.0**	Cohere Docs	Supports multiple sizes (384, 512, 1024).
1536	**OpenAI text-embedding-ada-002**	Pinecone	Former default OpenAI embedding model.
	OpenAI text-embedding-3-small	Zilliz	More efficient than Ada.
	Gemini gemini-embedding-001 (MRL)	Google AI	Can request 768, 1536, or 3072 dims.
2048	**Cohere embed-multilingual-v3.0**	Cohere Docs	Supports 512, 1024, or 2048 dims.
3072	**OpenAI text-embedding-3-large**	OpenAI	Default 3072, but can truncate down to e.g. 256–3072.
	Gemini gemini-embedding-001 (full)	Google AI	Full dimension output, best quality.

Source- https://weaviate.io/blog/how-to-choose-an-embedding-model

Embedding with Hugging Face

Example

```
[ ]
%pip install -U langchain-community sentence-transformers
%pip install -U langchain-huggingface
```

```
[ ]
from langchain_huggingface import HuggingFaceEmbeddings


embedding_model = HuggingFaceEmbeddings(
    model_name="sentence-transformers/all-MiniLM-L6-v2"
)
```

```
[23]
✓ 0s
text = "Ai is the new Electricity by Andrew NG"
embedding = embedding_model.embed_query(text)
```

```
[24]
✓ 0s
# print(f"Embedding Vector Lenght {len(embedding)}")
len(embedding)
```

```
384
```

```
[25]
✓ 0s
embedding[:5]
```

```
[-0.12379784882068634,
 0.03884607553482056,
 -0.0039674160070717335,
 0.014725077897310257,
 0.06532245874404907]
```

Embedding

Embedding of PDF Chunks

```
[ ]
    embeddings_list = []

    for page in pages:
        content = page.page_content
        # embedding = embedding_model.embed_query(content)
        embedding = embedding_model.embed_documents([content])[0]
        embeddings_list.append({
            "content": content,
            "embedding": embedding
        })

        })
```

```
print(embeddings_list[0]["embedding"][:5])
```

```
[-0.007731199264526367, -0.06998325139284134, 0.05110817775130272, 0.08354076743125916, 0.10189754515886307]
```


Q
A

ASSIGNMENTS FOR EMBEDDINGS

Task:

- **Generate embeddings for all chunks from your PDF.**

Instructions:

- **Use a HuggingFace model to create embeddings.**
- **Print a few embeddings to verify.**

Embedding

Q.4.Why do we use embeddings in RAG?

a) To split large texts into bits

b) To compress documents for storage

c) To store raw text for later use

d) To convert text into numeric vectors that capture meaning

Embedding

Q.5.What happens when two texts have similar embeddings?

a) They have the same length

b) They are similar in meaning

c) They share identical grammar patterns

d) They are merged into one chunk in the DB

Embedding

Q.6.Which example is closest to embeddings?

a) A music app suggesting similar tracks

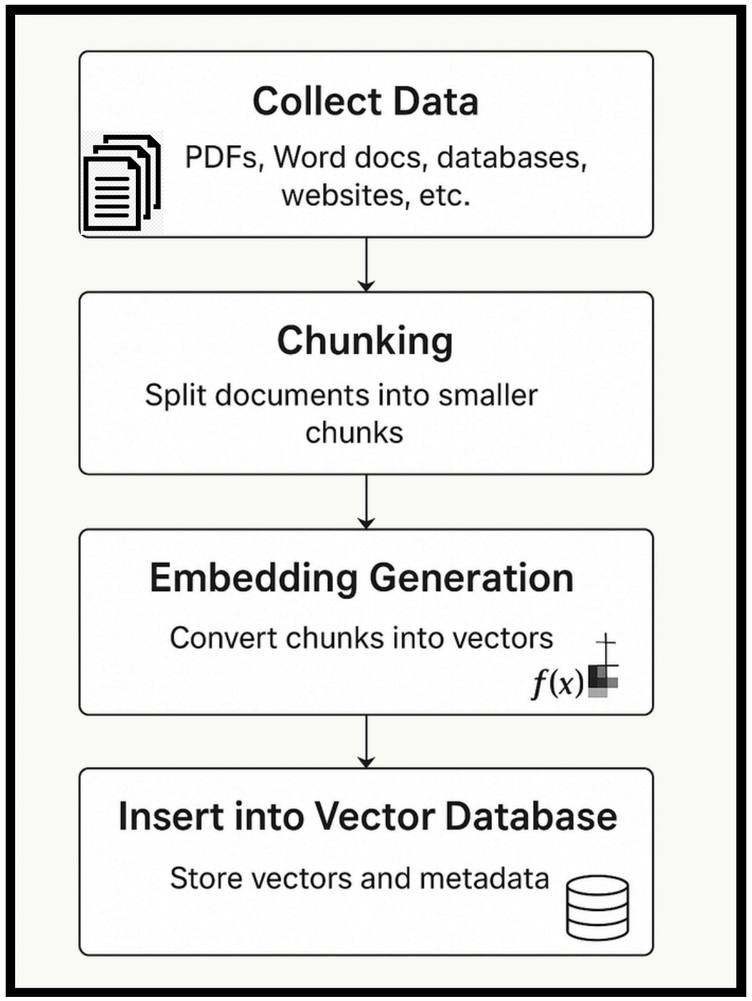
b) Cutting a movie into short clips

c) Zipping files into smaller archives

d) Copying notes into a notebook

Data insertion

The process of loading your documents into a vector database so they can be retrieved later.



Vector database

A **Vector Database** is a specialized database designed to store, index, and search high-dimensional vectors (embeddings) that represent the semantic meaning of text, images, or audio.

Chroma

redis

Pinecone Vector Database

Pinecone is a vector database optimized for fast and scalable similarity search across high-dimensional data.It powers AI applications like semantic search, recommendation engines, and real-time personalization.

[14]
✓ 12s

```
%pip install pinecone
```

Show hidden output

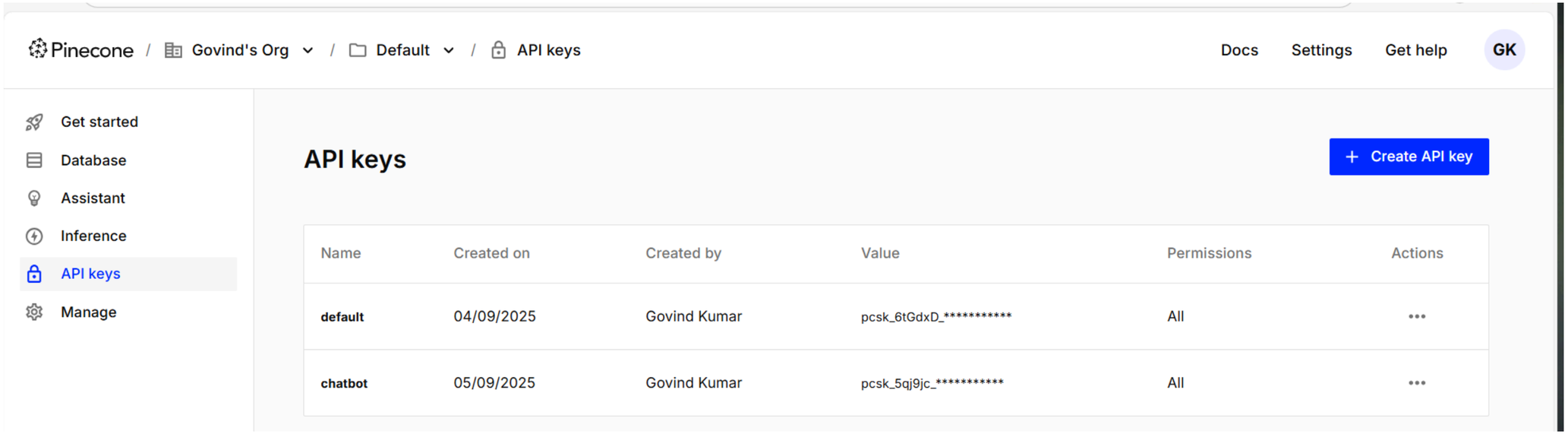
Create Pinecone Key

[15]
✓ 0s

```
from pinecone import Pinecone, ServerlessSpec

pinecone_api_key = "pcsk_5qj9jc_6WPE7Jr4rsB6GKM6nTZXL4EBpgu4oV6j6jh29TTnuEK8ihdZkqwtL9gQnh92T3b"
pc = Pinecone(api_key = pinecone_api_key)
```

Create Pinecone API Key

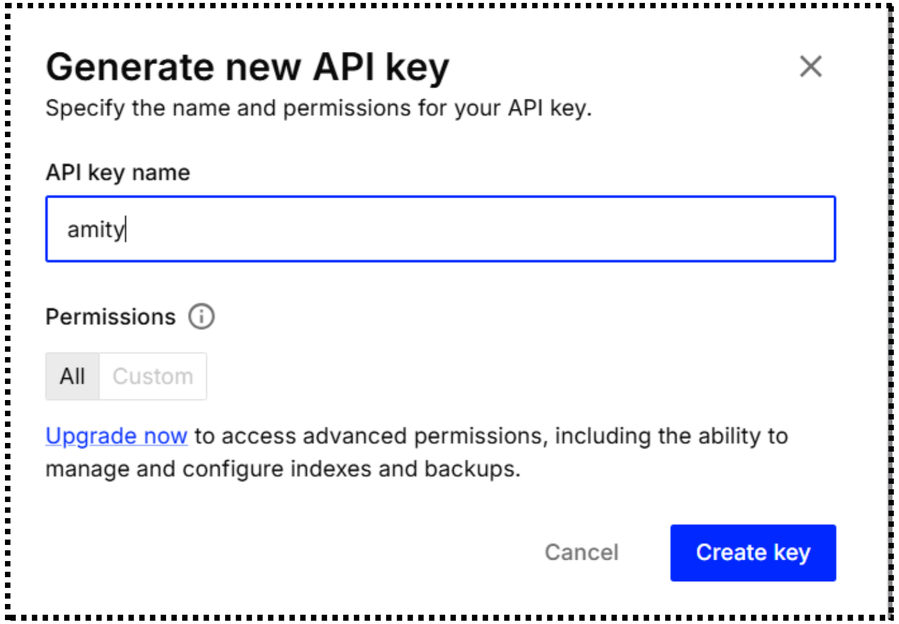
1. Open the Pinecone website and log in.
2. Navigate to API Keys In the sidebar, click on API Keys.
3. Click “Create API Key”

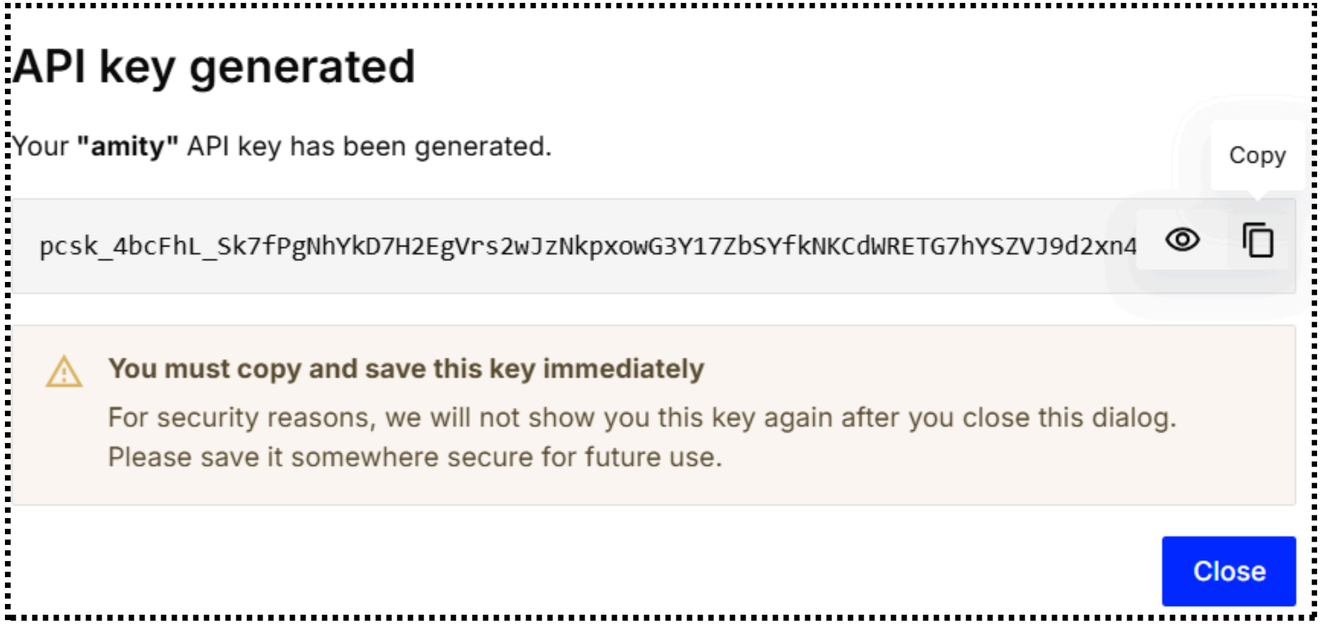


Source: https://app.pinecone.io/organizations/-OY_4Svrt9VBRxWjKq3e/projects/4a5df9f0-97c9-4e1b-8e1a-369aa5dbae4c/keys

Access Pinecone Api Key

- After click on “**Create API Key**”.
- A prompt will appear asking you to enter a name for your “**API key name**”.
- After typing the name, click the “**Create Key**” button.
- Your API key will be instantly generated and displayed.
- Use this key in your project to authenticate API calls to Pinecone.





Document Ingestion in Vector Database

Create Pinecone Key

```
from pinecone import Pinecone, ServerlessSpec

pinecone_api_key = "pcsk_5qj9jc_6WPE7Jr4rsB6GKM6nTZXL4EBpgu4oV6j6jh29TTnuEK8ihdZkqwtL9gQnh92T3b"
pc = Pinecone(api_key = pinecone_api_key)
```

Create Pinecone Index, If not created yet

```
index_name = "amity-database"

# Create index if it does not exist
if index_name not in pc.list_indexes().names():
    pc.create_index(
        name=index_name,
        dimension=384,
        metric="cosine",
        spec=ServerlessSpec(cloud="aws", region="us-east-1")  # adjust if needed
    )

index = pc.Index(index_name)
```

Data Insertion in Pinecone

Insert Data into Pinecone

```
records = []

for i, item in enumerate(embeddings_list):
    records.append({
        "id": f"chunk-{i}",
        "values": item["embedding"],
        "metadata": {"text": item["content"]}
    })

index.upsert(vectors=records, namespace="amity")
print(f"Stored {len(records)} records in Pinecone.")
```

```
Stored 16 records in Pinecone.
```

Dimension	Metadata (bytes)	Max batch size
386	0	1000
768	500	559
1536	2000	245

Request Size should be max 2MB in size


Q
A

ASSIGNMENTS FOR VECTOR DATABASE

Task:

- **Insert the embeddings into Pinecone.**

Instructions:

- **Connect to Pinecone.**
- **Store embeddings along with chunk metadata.**

Q.7.What extra information is often stored with embeddings in a vector database?

a) The exact search query used to find it

b) A short keyword list extracted from the text

c) Metadata like title, author, or source link

d) The token count used to build the embedding

Q.8.Which real-world example is closest to how a vector database works?

a) Files sorted by filename in folders

b) A paper dictionary arranged alphabetically

c) A photo app that finds visually similar faces

d) A clock showing the current time

Q.9.What is the main purpose of a vector database?

a) Quickly return vectors most similar to a query

b) Serve as long-term storage for raw documents

c) Automatically create embeddings for input text

d) Shorten text so it’s easier to read for people

Semantic Search

[54] ✓ 0s

```
query = " what is combination reactions. "
query_vector = embedding_model.embed_query(query)
```

[55] ✓ 0s

```
search_results = index.query(vector=query_vector, namespace="amity", include_metadata=True, top_k=1)
```

[56] ✓ 0s

```
len(search_results["matches"])
```

```
1
```

[57] ✓ 0s

```
for match in search_results['matches']:
    print(f"Score: {match['score']}")
    print(f"Text: {match['metadata']['text']}")
    print("-" * 50)
```

```
Score: 0.553288519
Text: Chemical Reactions and Equations 7
Let us discuss some more examples of combination reactions.
(i) Burning of coal
C(s) + O2(g)  →  CO2(g) (1.15)
(ii) Formation of water from H2(g) and O2(g)
2H2(g) + O2(g) →  2H2O(l) (1.16)
In simple language we can say that when two or more substances
(elements or compounds) combine to form a single product, the reactions
are called combination reactions.
In Activity 1.4, we also observed that a large amount of heat is evolved.
This makes the reaction mixture warm. Reactions in which heat is
released along with the formation of products are called exothermic
chemical reactions.
--------------------------------------------------
```


Q
A

ASSIGNMENTS FOR SEMANTIC SEARCH

Task:

- **Retrieve relevant chunks from the vector database.**

Instructions:

- **Use a query (e.g., a question related to the PDF content).**
- **Perform semantic search on Pinecone to retrieve the top 3 relevant chunks.**

Q.10.Why is semantic search better than keyword search?

a) It always runs faster than keyword methods

b) It finds related meaning even when words differ

c) It doesn’t need vector representations at all

d) It only works well on very small datasets

Q.11.What does the parameter top_k control?

a) How many top matches are returned

b) The dimensionality of the embedding vectors

c) Maximum length allowed for each text chunk

d) The speed or throughput of the database queries

Q.12.What might happen if top_k is set too high?

a) Only the single best match will be returned

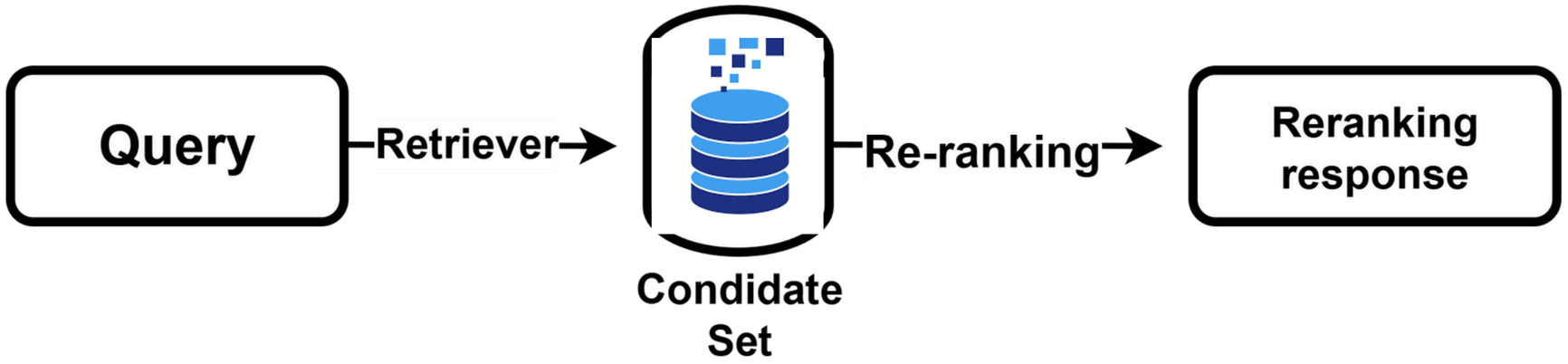
b) The LLM will become unable to generate output

c) Vectors in the database may get corrupted

d) Many low-relevance items may appear among results

Re-Ranking

Re-ranking is the process of reordering retrieved documents/chunks by their true relevance to the user query, before sending them to the LLM.



Cross Encoder model can be used for re-ranking retrieved results

Re-Ranking

Example

```
from sentence_transformers import CrossEncoder
reranker = CrossEncoder("cross-encoder/ms-marco-MiniLM-L-6-v2")
```

```
question = "What is AI?"
candidates = [
    "AI is transforming the world.",
    "Machine learning is a subset of AI.",
    "Deep learning uses neural networks.",
]

# Rerank candidates
scores = reranker.predict([(question, c) for c in candidates])

# Sort candidates by score (highest first)
reranked = sorted(zip(candidates, scores), key=lambda x: x[1], reverse=True)
```

```
# Show results
for text, score in reranked:
    print(f"Score: {score:.2f}, Text: {text}")
```

```
Score: 7.36, Text: AI is transforming the world.
Score: -0.55, Text: Machine learning is a subset of AI.
Score: -10.83, Text: Deep learning uses neural networks.
```


Q
A

ASSIGNMENTS FOR RERANKING

Task:

- **Rerank the retrieved chunks for better relevance.**

Instructions:

- **Take the top chunks from semantic search.**
- **Use a reranker model to rank them.**
- **Compare the model’s ranking with your own manual ranking.**

Q.13.Why do we rerank results?

a) To purge old vectors from the index

b) To split large results into smaller ones

c) To reduce token usage in prompts

d) To order retrieved items so the most useful come first

Q.14.Which example is similar to reranking?

a) Showing step-by-step arithmetic work

b) A search engine placing best links at the top

c) A printer arranging pages in order

d) A clock converting between time zones

Q.15.When is reranking most useful?

a) When only one chunk is returned from search

b) When many candidate chunks are retrieved at once

c) When the vector DB is offline or unreachable

d) When no embeddings exist in the system

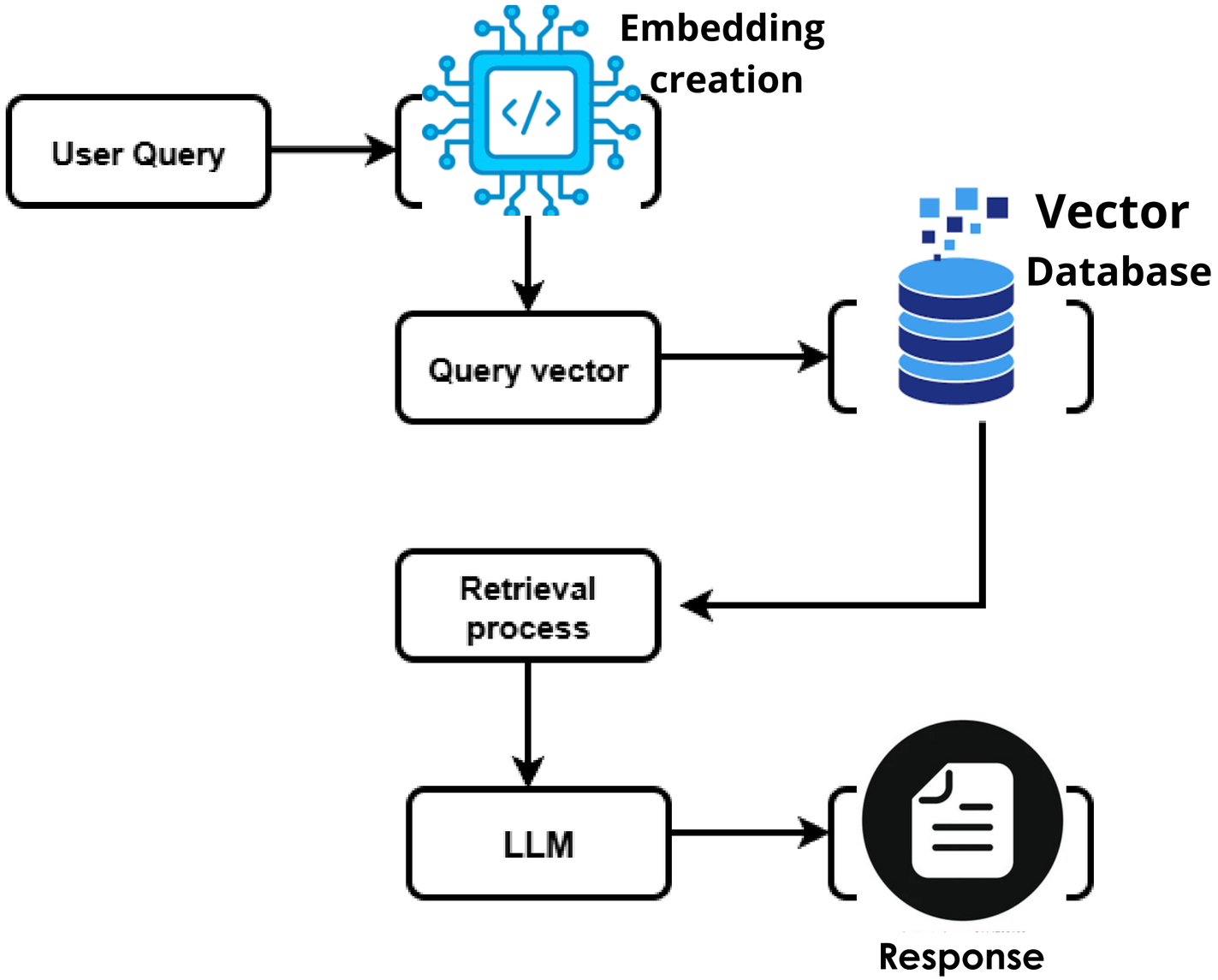
FEEDBACK
&
SUGESSIONS

edquest

LLM

A **Large Language Model** (LLM) is an AI system trained on massive amounts of text to understand, generate, and reason with natural language.

Response


Embedding
creation
User Query
Vector
Database
Query vector
Retrieval
process
LLM
Response

LLM Response

```
%pip install -U langchain-google-genai
```

Show hidden output

```
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain.prompts import PromptTemplate
```

```
api_key = "AIzaSyCk2trR6vYKnTO5RKFlx3i8dsO5NB-lJIg"

# Initialize Gemini model
model = ChatGoogleGenerativeAI(
    model="gemini-1.5-flash",
    google_api_key=api_key
)
```

```
# Build prompt template
prompt = PromptTemplate.from_template("""
You are a helful assistant. Use the provided context to answer the query
If the answer is not in the context, say you don't know.

Context:
{context}

Query : {query}

Answer:
""")
```

```
chain = prompt | model

# Invoke the chain with your context and query
result = chain.invoke({
    "context": context,
    "query": query
})
```

```
print(result.content)
```

```
print(result.content)
```

```
The iron nail becomes brownish because iron displaces copper from the copper sulphate solution, forming iron sulphate and copper.
```


Q
A

ASSIGNMENTS FOR FINAL RESPONSE

Task:

Generate a final answer using the top reranked chunks.

Instructions:

- **use Gemini for response**
- **Combine the top reranked chunks and pass them to an LLM to generate a factual answer.**
- **Compare the generated response with the content in the PDF.**
- **Check if the response is correct and matches the PDF content.**

Q.16.Why does the LLM need the retrieved chunks?

a) To reduce the size of the embeddings themselves

b) To continue training the base model online

c) To shorten the model's final reply to fit limits

d) To ground answers with factual source material

Q.17.What happens if the retrieved chunks are not relevant?

a) The vector DB will automatically restart itself

b) The stored embeddings will be deleted to free space

c) The model may hallucinate or give incorrect answers

d) The user’s query will be blocked from running further

Q.18.Which example is closest to response generation?

a) A student reading notes and writing an answer

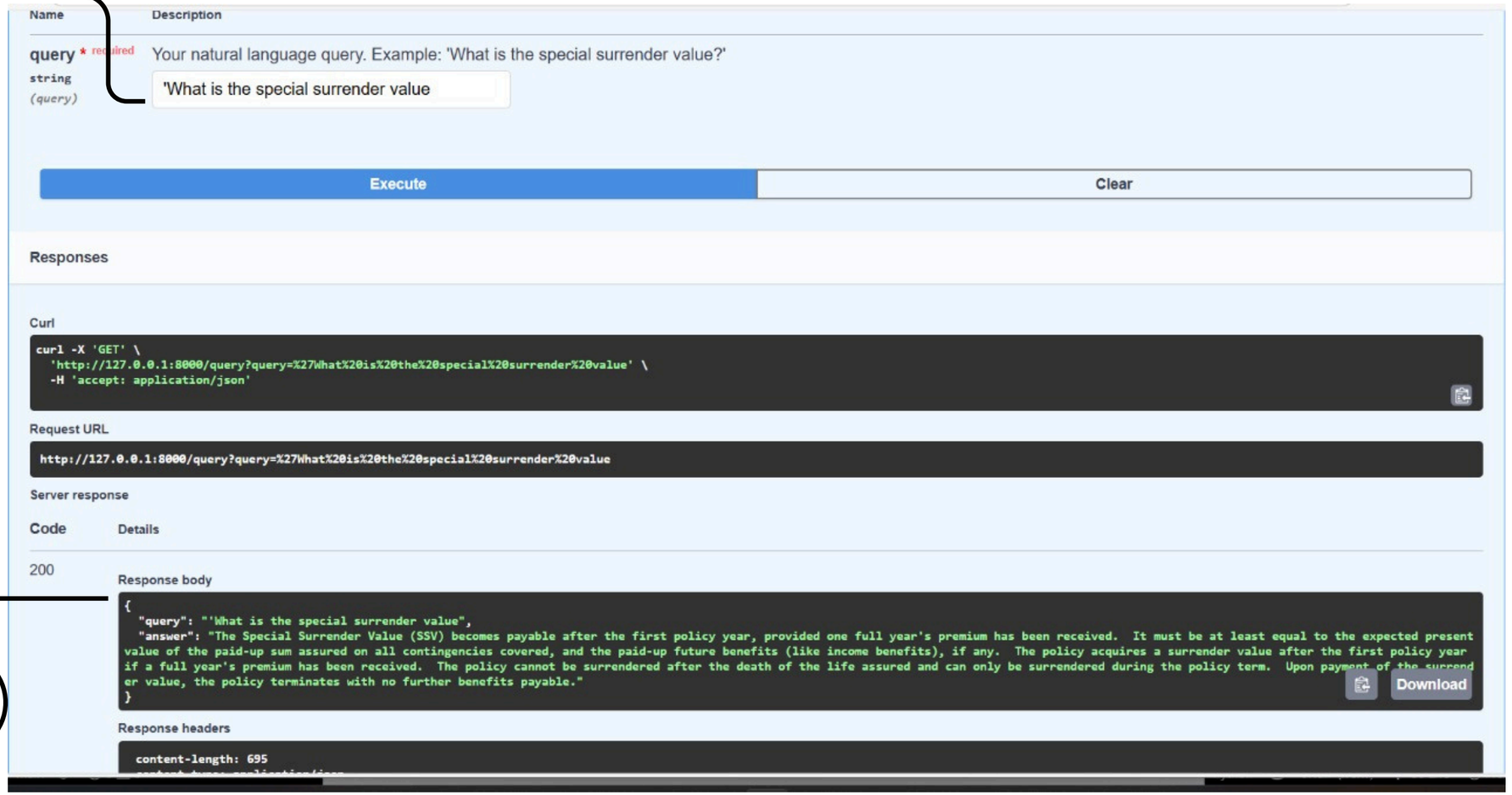
b) A library only shelving books without use

c) Taking a photo of a page to archive it

d) Looking up a short definition in a dictionary

FAST API UI

Insert your Query here



response

GITHUB URL

githubURL:_
https://github.com/edquestofficial/corporate_trainings/tree/main/Gen_AI/Amity_Software_Gen_AI_Foundations

Why RAG ?

- Reduces hallucinations.
- Keeps answers domain-specific (finance, insurance, SAP, agriculture).
- Provides up-to-date knowledge without re-training the LLM.
- Enables citations so you know where the answer came from.

Resources

📘 Beginner Guides

- **DeepLearning.AI RAG Course**
  - https://learn.deeplearning.ai/courses/retrieval-augmented-generation-rag
- **LangChain RAG Docs**
  - https://python.langchain.com/docs/use_cases/question_answering/
- **Pinecone RAG Guide**
  - https://docs.pinecone.io/docs/retrieval-augmented-generation
- **Weaviate Academy**
  - https://weaviate.io/developers/academy
- **Cohere RAG Examples**
  - https://docs.cohere.com/docs/retrieval-augmented-generation-rag

Agentic AI: From Retrieval to Strategic Thinking

edquest

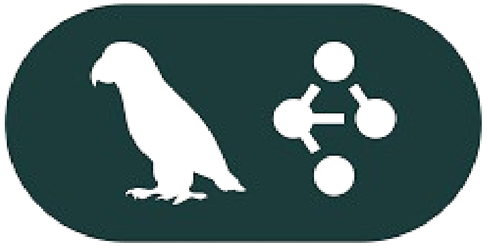
- **Chatbots** just respond to your prompts.
- Agentic AI can define its own **goals**, create a **plan**, and **execute** it using a variety of tools.
- This technology moves us beyond simple, repetitive tasks to automating complex, **multi-step workflows**.

LangGraph: Build RAG with Agents

edquest

- Not just a simple sequence of steps, but a **visual, interconnected flow chart** for your AI.
- It's how we build "**agents**" – AI systems that can **think, decide, and act**. They can correct their own mistakes, loop back, and explore new paths.
- Moving from simple RAG to **autonomous, conversational, and multi-step reasoning systems**.

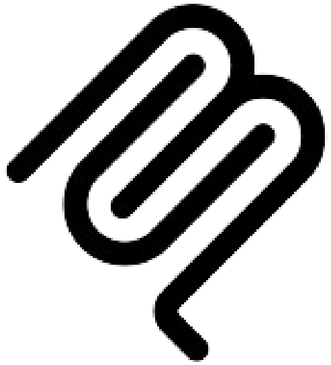
Source : https://academy.langchain.com/courses/intro-to-langgraph



MCP: Universal Language for Models

edquest

- Every AI model has its own way of **"talking" about context**. This makes it hard to mix and match different models.
- MCP is a **game-changer**. It's a universal standard that allows all **models to understand & share context** seamlessly.



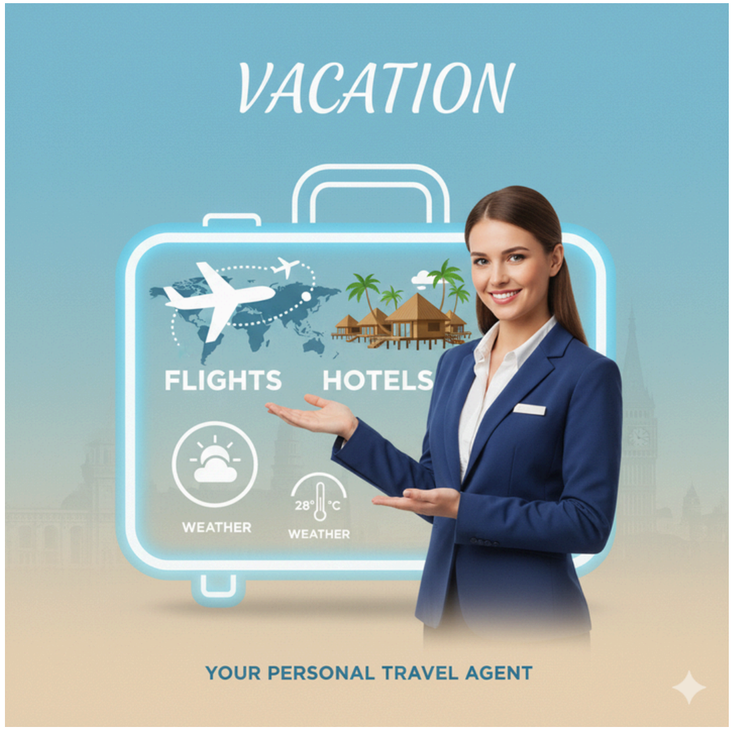
Source : https://www.deeplearning.ai/short-courses/mcp-build-rich-context-ai-apps-with-anthropic/

Agentic Workflow in Action

edquest

Task : Planning a vacation involves multiple steps and different sources of information.

- **Travel Agent**: Searches for the best flights.
- **Weather Agent**: Checks the forecast for your dates.
- **Hotel Booking Agent**: Finds top-rated hotels.
- **Itinerary Agent**: Combines all the information into a perfect, personalized plan.



edquest