

Week 4: Embedding Models and Advanced Queries

 by Alwin Lin

Class Overview

- **Goals for Today:**
 - Setting up Gemini
 - Introduction to:
 - Vector Embedding
 - Embedding Models
 - Vector Database
 - Basics of RAG
 - Usage throttling and API limits
- **Prerequisite Knowledge:**
 - Basic prompt engineering, token counting, and model parameters

Recap of Previous Lessons

- **Prompt Engineering:** Techniques to craft effective prompts.
- **Token Counting:** Understanding token limits for efficient querying.
- **Model Parameters:** Key parameters like temperature, max tokens, and top_p.
- **Importance of Understanding API Limits:** Helps avoid unnecessary costs or throttling issues.

Setting Up Gemini – Quick Review

- **Setting up API Keys:**
 - Navigate to [Gemini API Docs](#) for setup guide.
- **Environment Configuration:**
 - Install necessary packages (e.g., `gemini-client`, `openai`, etc.).
 - Set up authentication.
- **Usage Throttling:**
 - **Rate Limiting:** Requests per minute/hour.
 - **Practical Tips:** Set usage limits for small projects in the Gemini dashboard.

Solutions ▾ Code assistance ▾ Showcase ▾ More ▾

ricing Cookbook

Gemini 2.0 Flash Experimental is now available! [Learn more](#)

me > Gemini API > Models

Gemini Developer API

[Get a Gemini API Key](#)

Get a Gemini API key and make your first API request in minutes.

Python Node.js REST

```
import google.generativeai as genai

genai.configure(api_key="YOUR_API_KEY")
model = genai.GenerativeModel("gemini-1.5-flash")
response = model.generate_content("Explain how AI works")
print(response.text)
```

Select the models

[Use Gemini in Google AI Studio](#)

2.0 Flash 🧪

Our newest multimodal model,

1.5 Flash-8B

Our fastest and most cost-

1.5 Pro

Our best performing multimodal

[Made with Gamma](#)

Introduction to Embedding Models

- **What Are Embeddings?**

- A technique to represent text data as high-dimensional vectors.
- Captures semantic meaning, unlike simple keyword-based methods.
- Most LLMs (GPT, Gemini, LLAMA) have their respective embedding models

- **How Do They Work?**

- We throw a lot of text data into a model.
- The model looks at which words tend to show up together in similar contexts.
- Over time, it learns to represent words as vectors (points) in a space where similar meanings are grouped together.
- These vectors reflect the semantic relationships between words based on how they are used in real-world data.

- **Why Do We Use them?**

- Allows search without exact phrase
- Provides insight into datas

Usage Throttling Explained

- **What is Throttling?:**
 - Manage load and avoid service interruptions.
 - Key settings in Gemini's setup for managing your request frequency.
- **When to Use Throttling:**
 - Avoid high costs with excessive requests.
 - Improve system reliability for personal or production projects.
- **Do you need to be concerned?**
 - For this class, [not really](#).

Free of charge

The Gemini API "free tier" is offered through the API service with lower rate limits for testing purposes. Google AI Studio usage is completely free in all [available countries](#).

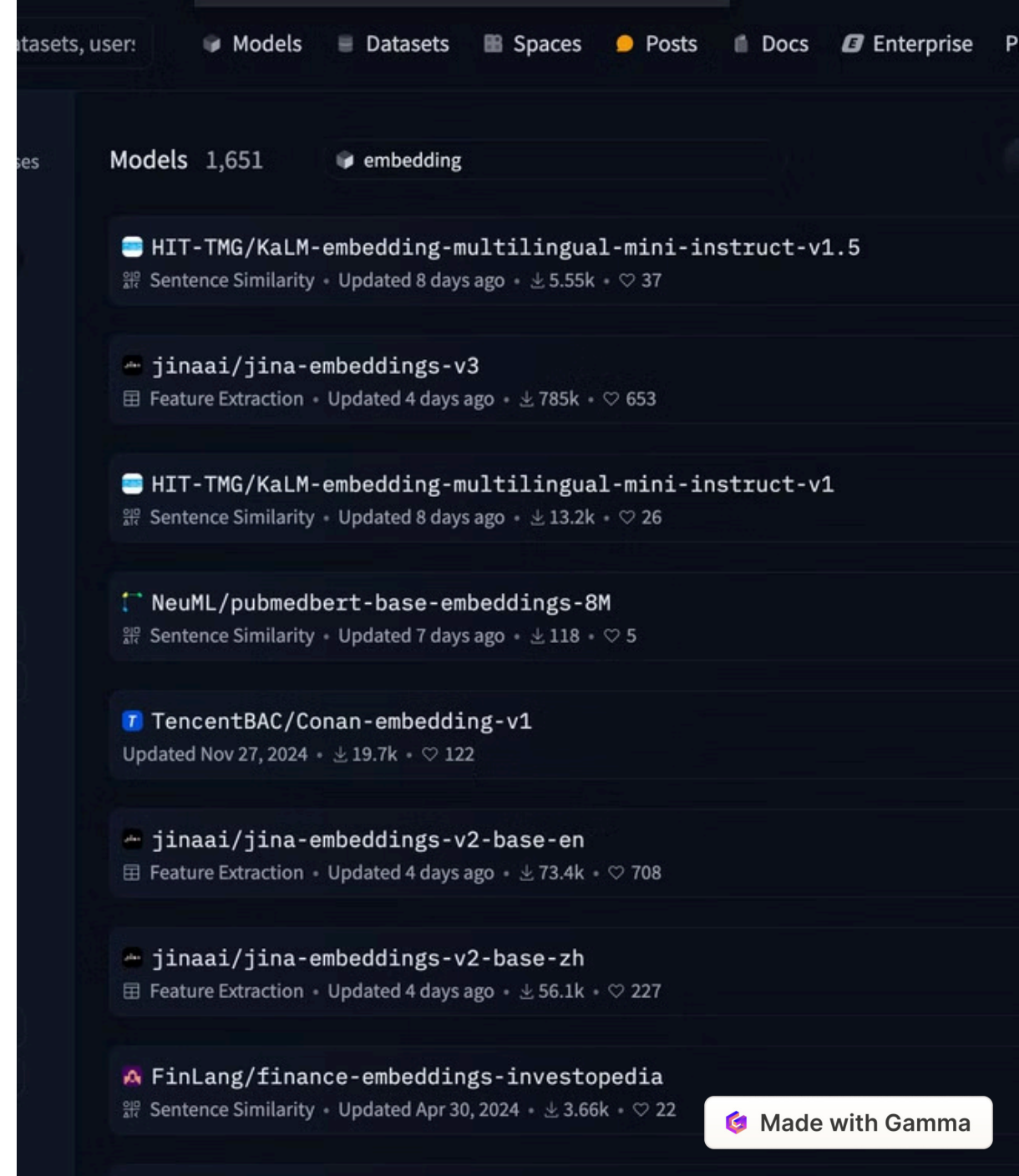
RATE LIMITS	15 RPM (requests per minute) 1 million TPM (tokens per minute) 1,500 RPD (requests per day)
INPUT PRICING	Free of charge
OUTPUT PRICING	Free of charge
CONTEXT CACHING	Free of charge, up to 1 million tokens of storage per hour
TUNING PRICE	Input/output prices are the same for tuned models. Tuning service is free of charge.
GROUNDING WITH GOOGLE SEARCH	Not available
USED TO IMPROVE OUR PRODUCTS	Yes

Embeddings vs. Text Generation

- **Text Generation Models:**
 - Generate content based on input prompts.
 - Example: ChatGPT, GPT-4.
- **Embedding Models:**
 - Encode text into vectors that can be compared for similarity.
 - Example: Sentence-BERT, OpenAI Embedding API.

Example Embedding Models:

- [OpenAI Embedding Docs](#): text-embedding-3-small/large
- [Google Embeddings](#): text-embedding-005
- [Sentence-BERT GitHub](#)
- [Many more open source options on Hugging face](#)



Why do we care which model is used?

- [Just like LLMs, Embedding models are different](#)

Spaces | mteb/leaderboard | like 4.51k | Running on CPU UPGRADE | App | Files | Community 149

Massive Text Embedding Benchmark (MTEB) Leaderboard. To submit, refer to the [MTEB GitHub repository](#). Refer to the [MTEB paper](#) for details on metrics, tasks and models. Also check out [MTEB Arena](#).

Search Bar (separate multiple queries with `;`)
Search for a model and press enter...

Model types
☒ Open ☒ Proprietary ☒ Sentence Transformers ☒ Cross-Encoders
☒ Bi-Encoders ☒ Uses Instructions ☒ No Instructions

Model sizes (in number of parameters)
☒ <100M ☒ 100M to 250M ☒ 250M to 500M
☒ 500M to 1B ☒ >1B

Overall | Bibtex Mining | Classification | Clustering | Pair Classification | Reranking | Retrieval | STS | Summarization | MultilabelClassification

Retrieval w/Instructions

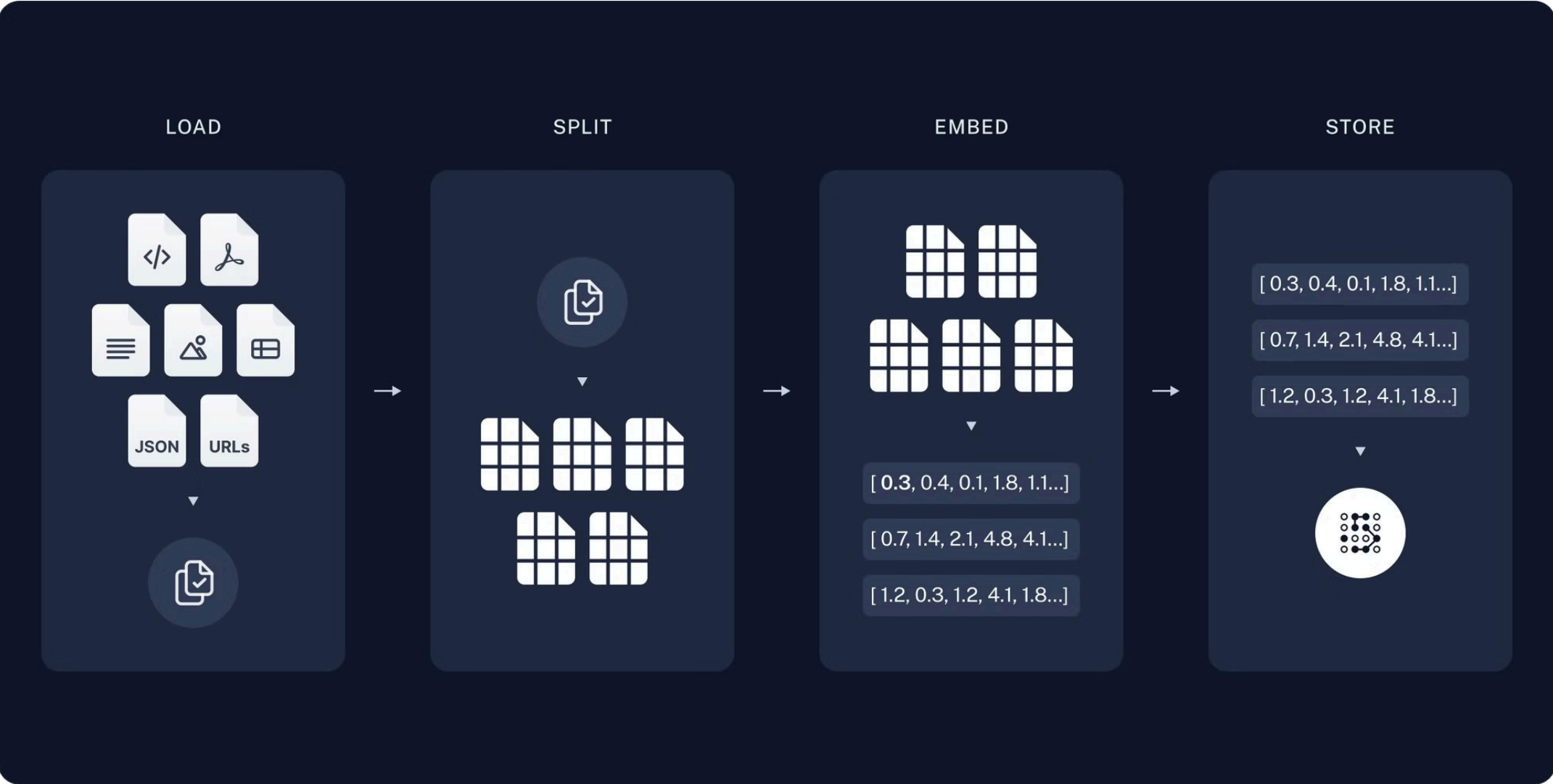
English | Chinese | French | Polish | Russian

Overall MTEB English leaderboard

- Metric: Various, refer to task tabs
- Languages: English

Rank	Model	Model Size (Million Parameters)	Memory Usage (GB, fp32)	Embedding Dimensions	Max Tokens	Average (56 datasets)	Classification Average (12 datasets)	Clustering Average (11 datasets)
1	voyage-3-m-exp					74.03	90.16	61.45
2	NV-Embed-v2	7851	29.25	4096	32768	72.31	90.37	58.46
3	jasper_en_vision_language_v1					72.02	88.49	58.04
4	bge-en-icl	7111	26.49	4096	32768	71.67	88.95	57.89
5	stella_en_1.5B_v5	1543	5.75	8192	131072	71.19	87.63	57.69
6	SFR-Embedding-2_R	7111	26.49	4096	32768	70.31	89.05	56.17
7	gte-Qwen2-7B-instruct	7613	28.36	3584	131072	70.24	86.58	56.92

Embed workflow



What does the code look like

- **Step 1: Prepare Data**
 - Collect a set of documents
- **Step 2: Generate Embeddings**
 - Embed the entire dataset.
- **Step 3: Query Processing**
 - Do something with the embedding

```
import google.generativeai as genai
import os

genai.configure(api_key=os.environ["GEMINI_API_KEY"])

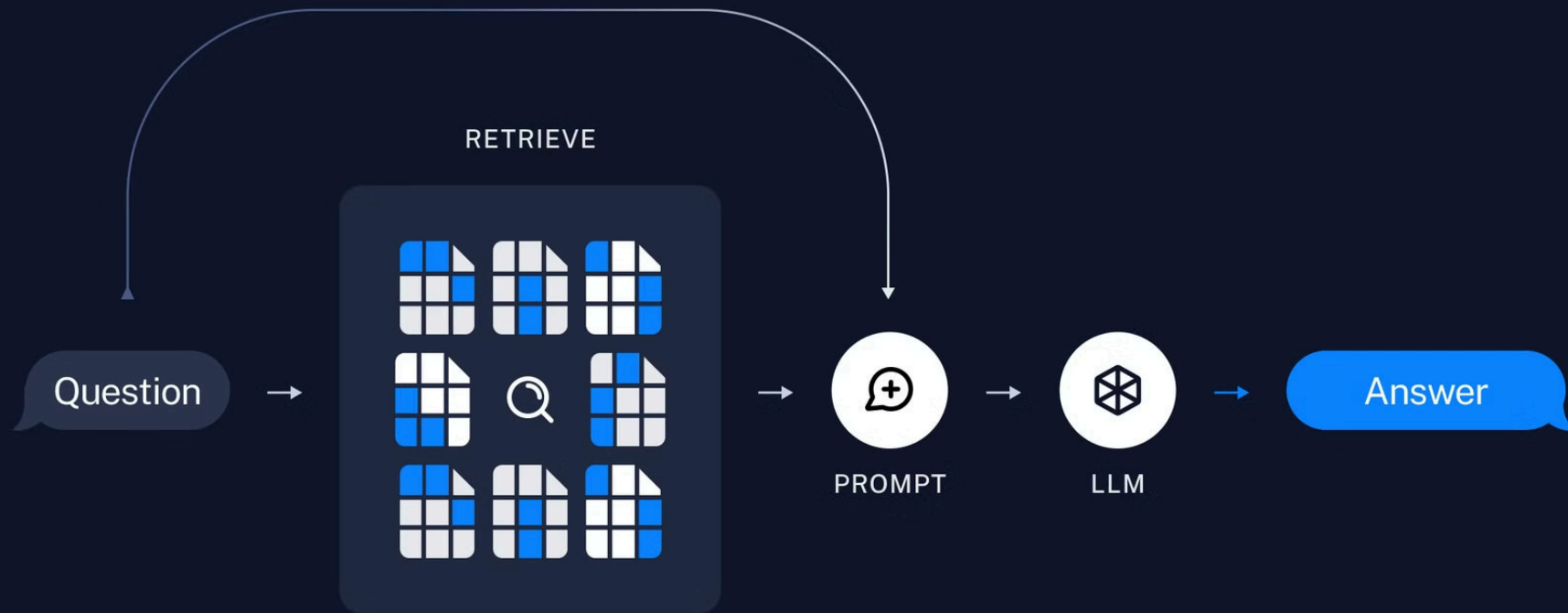
result = genai.embed_content(
    model="models/text-embedding-004",
    content="What is the meaning of life?")

print(str(result['embedding']))
```

Practical Applications of Embeddings

- **Semantic Search:**
 - Generate embeddings for documents and queries.
 - Find the most semantically similar documents.
- **Clustering:**
 - Use embeddings to cluster similar documents or texts together.
 - Visualize with 2D or 3D plots using PCA or t-SNE.
- **Recommendation Systems:**
 - Use embedding-based similarity for recommending items (e.g., products, articles).

RAG Workflow



How do I store all these vectors?

If only... There was a database

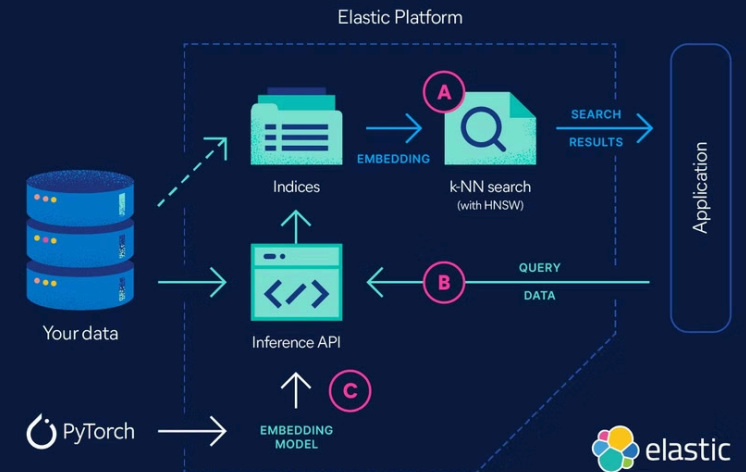
- What is it?
- Why do we use it?
- What is the significance of it?

Create vector embeddings inside Elastic

Elasticsearch provides all three key capabilities in one platform:

- A** Nearest neighbor search,
- B** Inference API, and
- C** Embedding model

— GENERATION OF EMBEDDINGS
— EXECUTION OF VECTOR SEARCH



What are my options?

Vector DB	Pros	Cons
FAISS	High performance, flexible indexing, scalable, open-source.	Complex setup, memory management, limited API support, no metadata.
Pinecone	Fully managed, easy to use, scalable, real-time updates, advanced features.	Expensive, less control, no open-source option.
Weaviate	Hybrid search, ML model integration, graph + vector search, scalable.	Complex setup, higher resource consumption, fewer indexing options.
Milvus	High performance, multi-index support, scalable, open-source.	Complex deployment, resource consumption, limited real-time updates.
Chroma	Simple, user-friendly, optimized for embeddings, open-source.	Limited scalability, fewer advanced features, not ideal for large-scale.

Chroma, How does it work?

- Install Chroma
- Setup Chroma Client
- Create and add docs to collection
- Query

```
import chromadb

client = chromadb.Client()

collection = client.create_collection("all-my-documents")
collection.add(
    documents=["This is document1", "This is document2"], # we handle tokenization, embedding, and indexing automatically. You
    can skip that and add your own embeddings as well
    metadatas=[{"source": "notion"}, {"source": "google-docs"}], # filter on these!
    ids=["doc1", "doc2"], # unique for each doc
)

results = collection.query(
    query_texts=["This is a query document"],
    n_results=2,
    # where={"metadata_field": "is_equal_to_this"}, # optional filter
    # where_document={"$contains": "search_string"} # optional filter
)
```

Key Takeaways

- **Embeddings:** Essential for advanced text analysis and search tasks.
- **Gemini Setup:** Remember to manage API keys and set usage limits for personal projects.
- **Practical Applications:** Semantic search, clustering, and recommendation systems are real-world uses of embeddings.
- **Next Steps:** Try building your own semantic search or clustering system using embeddings.