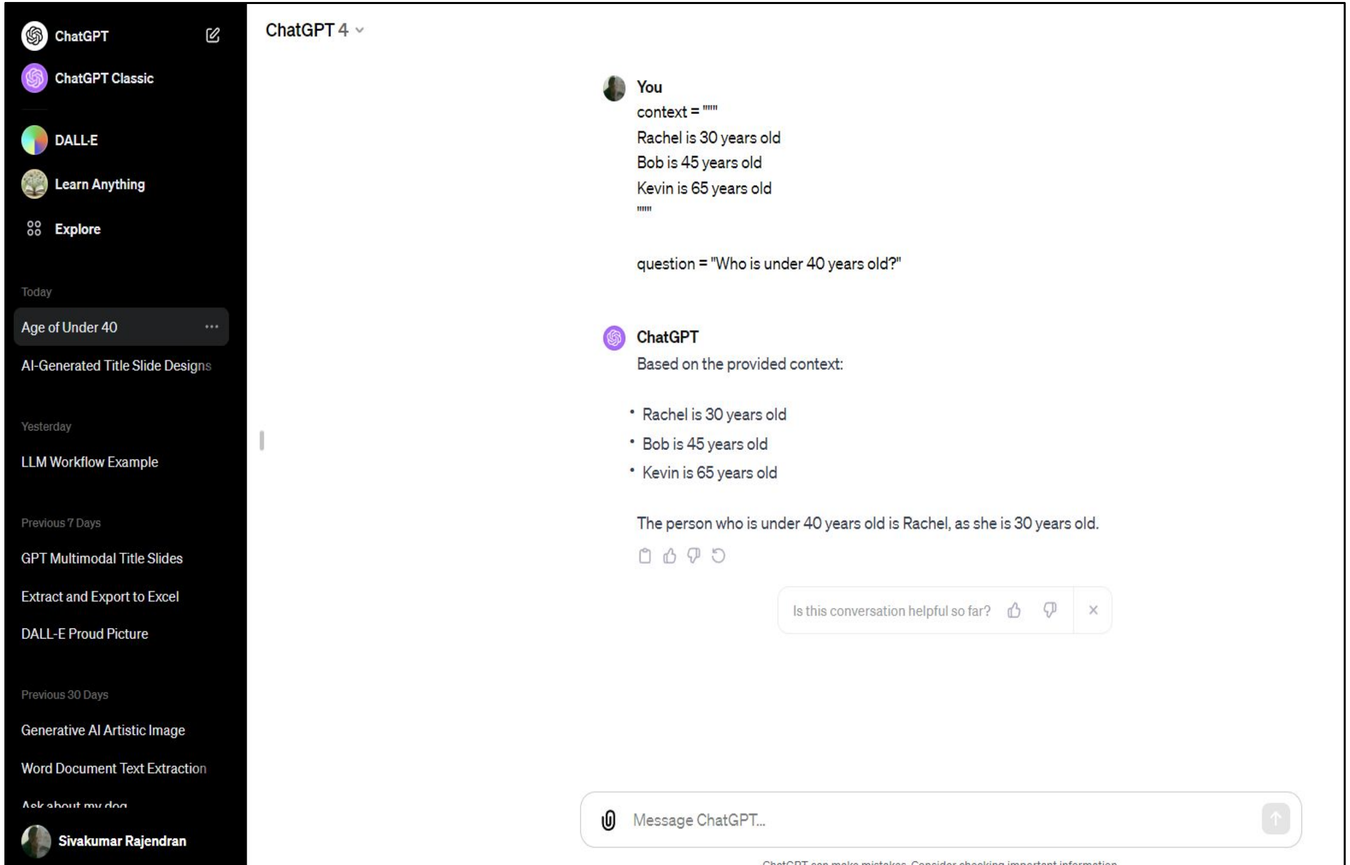

Generative AI
DAY 2

Let understand Interacting with LLM through UI vs Code



Simple Q&A Example

Here let's review the convention of `llm(your context + your question) = your answer`

In [13]:
```
from langchain.llms import OpenAI

llm = OpenAI(temperature=0, openai_api_key=openai_api_key)
```

In [14]:
```
context = """
Rachel is 30 years old
Bob is 45 years old
Kevin is 65 years old
"""

question = "Who is under 40 years old?"
```

Then combine them.

In [15]:
```
output = llm(context + question)

# I strip the text to remove the leading and trailing whitespace
print (output.strip())
```

```
Rachel is under 40 years old.
```

"https://github.com/gkamradt/langchain-tutorials/blob/main/LangChain%20Cookbook%20Part%202%20-%20Use%20Cases.ipynb"

Hence, accessing the LLM through Code is not as easy as accessing through Web UI.

It requires a different mindset, different workflow and different skillset.

Can I build a new LLM and the application?

How much did it cost to develop GPT-3?

According to OpenAI, the research organization responsible for developing GPT-3, the project's total cost is estimated to be around $4.6 million. This includes not only the cost of developing the model itself but also the cost of training it using massive amounts of data.

Technext
https://technext24.com › 2023/08/14 › chatgpt-costs-7...

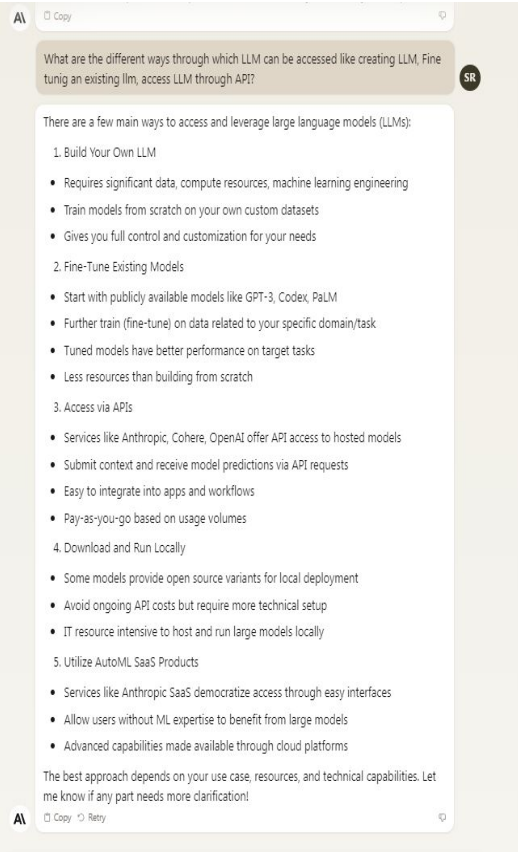
ChatGPT costs $700000 to run daily, OpenAI may go ...

Challenges

Cost Infrastructure Skillset ROI Success/ Guaranteed Result?!

So, what are the options?

Different ways of accessing LLM



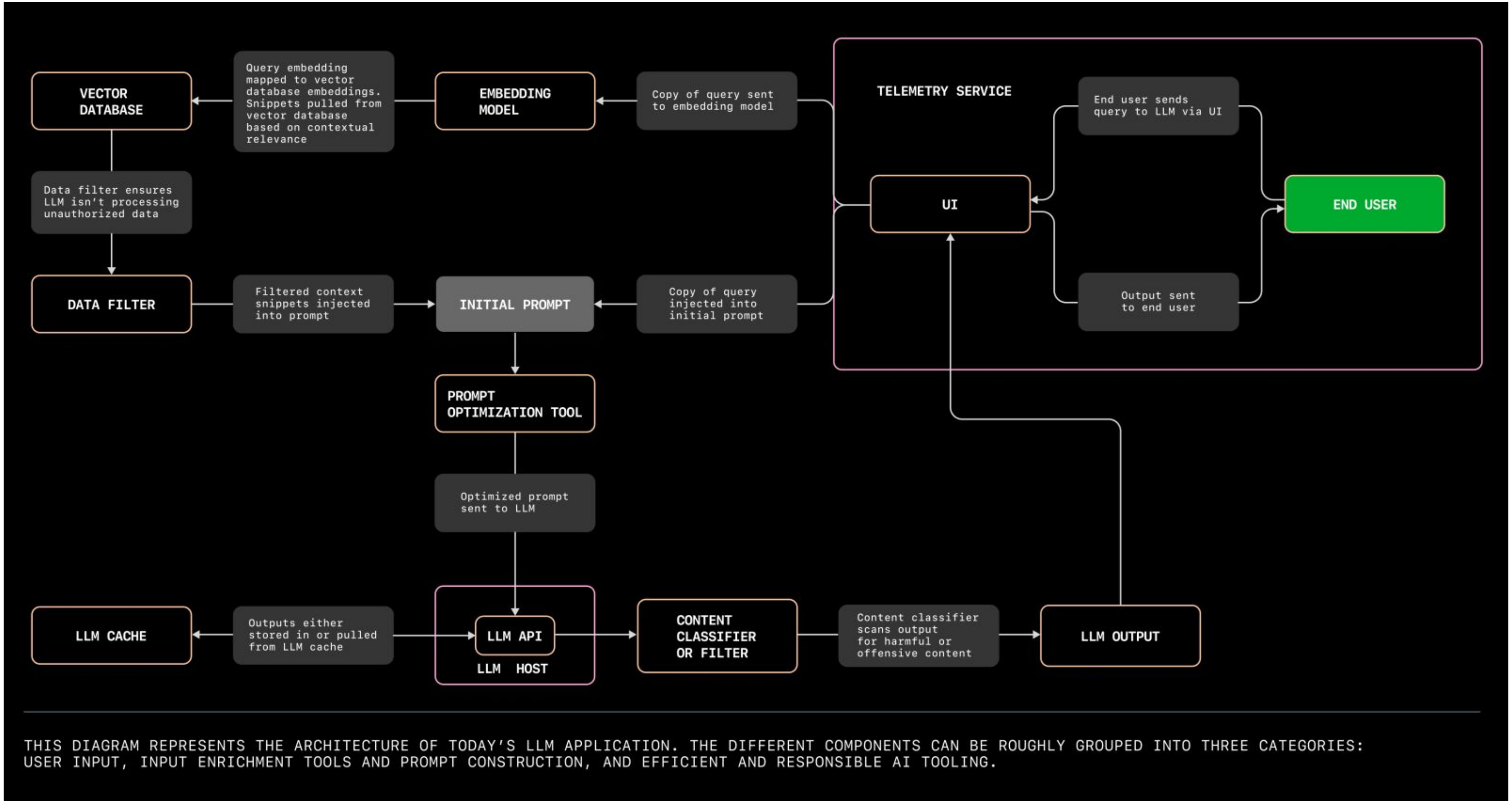
There are a few main ways to access and leverage large language models (LLMs):

1. **Build Your Own LLM**
   - Requires significant data, compute resources, machine learning engineering
   - Train models from scratch on your own custom datasets
   - Gives you full control and customization for your needs
2. **Fine-Tune Existing Models**
   - Start with publicly available models like GPT-3, Codex, PaLM
   - Further train (fine-tune) on data related to your specific domain/task
   - Tuned models have better performance on target tasks
   - Less resources than building from scratch
3. **Access via APIs**
   - Services like Anthropic, Cohere, OpenAI offer API access to hosted models
   - Submit context and receive model predictions via API requests
   - Easy to integrate into apps and workflows
   - Pay-as-you-go based on usage volumes
4. **Download and Run Locally**
   - Some models provide open source variants for local deployment
   - Avoid ongoing API costs but require more technical setup
   - IT resource intensive to host and run large models locally
5. **Utilize AutoML SaaS Products**
   - Services like Anthropic SaaS democratize access through easy interfaces
   - Allow users without ML expertise to benefit from large models
   - Advanced capabilities made available through cloud platforms

The best approach depends on your use case, resources, and technical capabilities. Let me know if any part ne
more clarification!

How to build a LLM Application?

Architecture of LLM Applications



https://github.blog/2023-10-30-the-architecture-of-todays-llm-applications/

If the architecture is Complex to get start with, let’s do a reverse engineering.

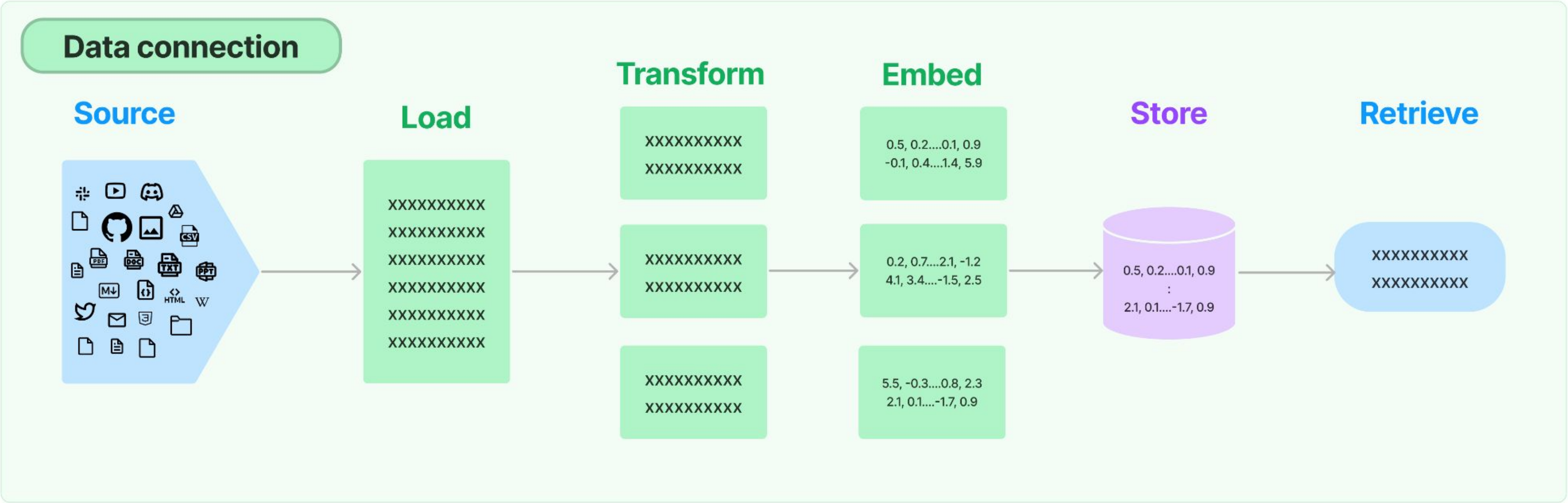
Start with a LLM code and modularize that

Let observe few Code snippets

```
# Sample code to illustrate the stages in building an LLM
application

# 1. Document Loader
def load_documents(file_paths):
    documents = []
    for path in file_paths:
        with open(path, 'r') as file:
            documents.append(file.read())
    return documents

# 2. Embedding
def embed_documents(documents):
    # Placeholder for a complex embedding process
    embeddings = ["embedding" + str(i) for i, _ in
enumerate(documents)]
    return embeddings

# 3. Indexing
def index_embeddings(embeddings):
    # Placeholder for an indexing mechanism
    indexed_embeddings = {i: embeddings[i] for i in
range(len(embeddings))}
    return indexed_embeddings

# 4. Querying
def process_query(query):
    # Placeholder for query processing
    query_embedding = "query_embedding"
    return query_embedding

# 5. Retrieval
def retrieve_documents(query_embedding,
indexed_embeddings):
    # Placeholder for document retrieval logic
    relevant_documents = ["doc1", "doc2"]  # Example of
```


```
# Sample code to illustrate the stages in building an LLM
application

# 1. Document Loader
def load_documents(file_paths):
    documents = []
    for path in file_paths:
        with open(path, 'r') as file:
            documents.append(file.read())
    return documents

# 2. Embedding
def embed_documents(documents):
    # Placeholder for a complex embedding process
    embeddings = ["embedding" + str(i) for i, _ in
enumerate(documents)]
    return embeddings

# 3. Indexing
def index_embeddings(embeddings):
    # Placeholder for an indexing mechanism
    indexed_embeddings = {i: embeddings[i] for i in
range(len(embeddings))}
    return indexed_embeddings

# 4. Querying
def process_query(query):
    # Placeholder for query processing
    query_embedding = "query_embedding"
    return query_embedding

# 5. Retrieval
def retrieve_documents(query_embedding,
indexed_embeddings):
    # Placeholder for document retrieval logic
    relevant_documents = ["doc1", "doc2"]  # Example of
```

Building Block of LLM



https://js.langchain.com/docs/modules/data_connection/

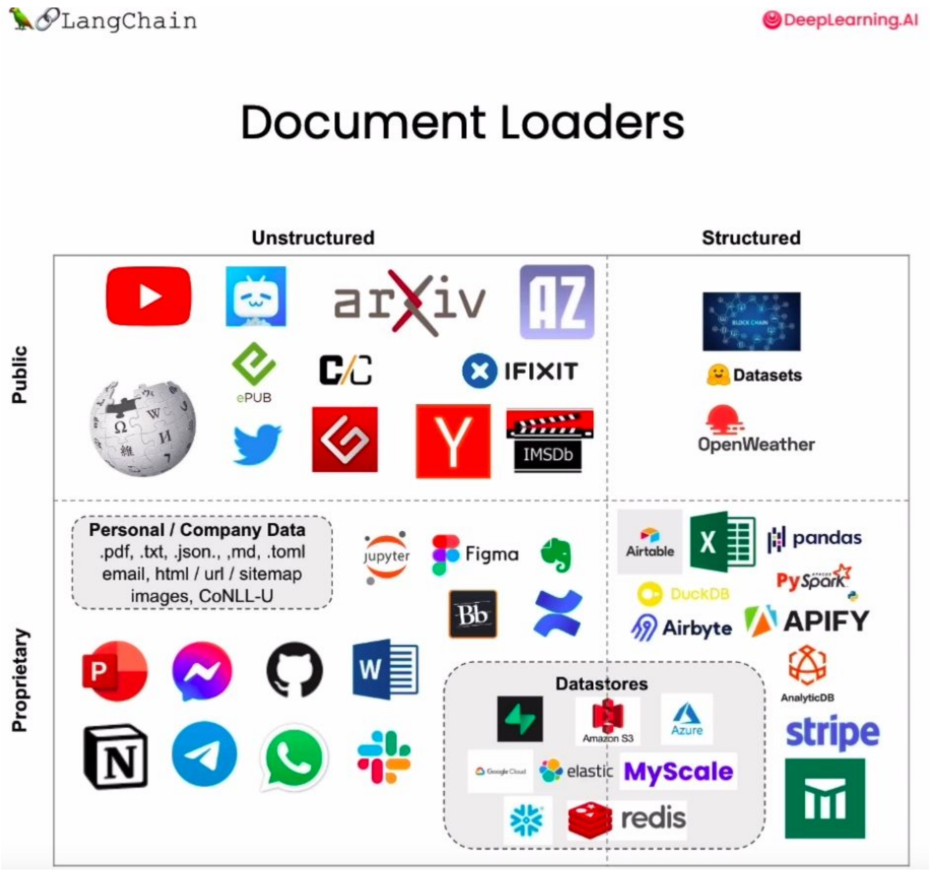
Before get into Building blocks, let see what is Embedding

Like Computers understands only **0 and 1,**
Algorithms can understand only **NUMBERS**

Hence, whatever the data source, that have to be converted to NUMBERS.
When store the numbers based on the relationship, retrieval becomes effective.

For more details, “https://jalammar.github.io/illustrated-word2vec/”

Document Loaders



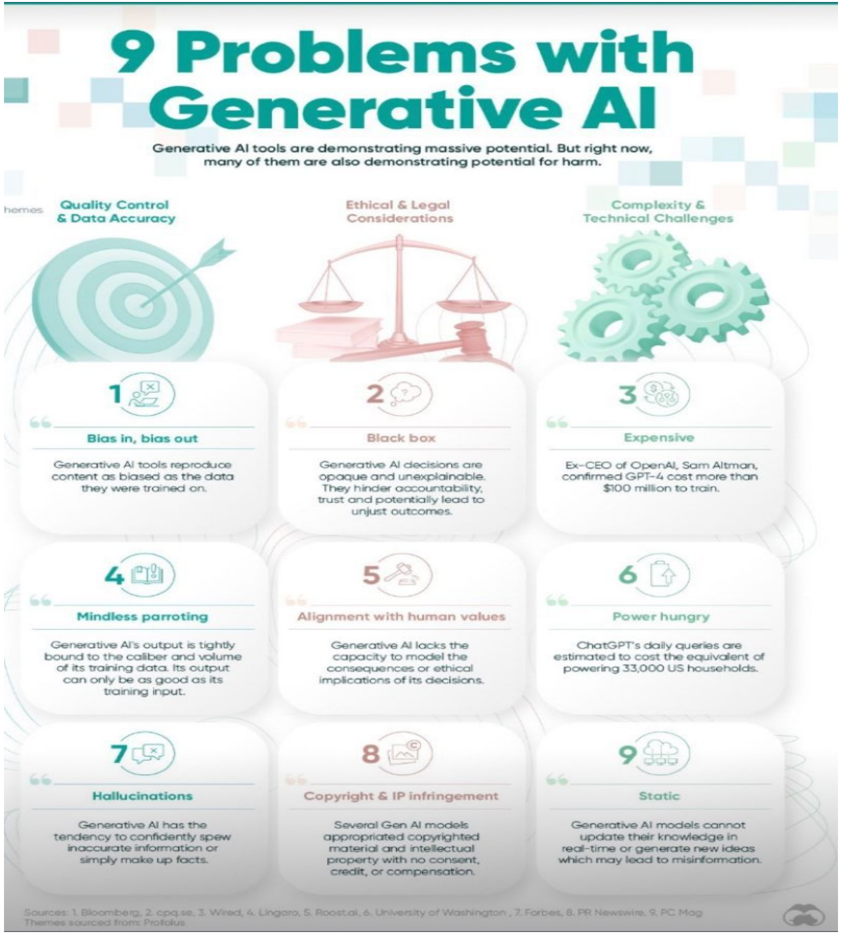
Transformers

https://bbycroft.net/llm

Vector Database - A Comparison

#	DB | Attributes	Open-source/self-host	Managed Cloud Offering	In-built Embeddings creation	Hybrid Search	Metadata Filtering	BM25 support	Full-text Search Engine	Multiple vectors per point	Tensor Search	Streaming Index	Langchain integration	Llama index integration	Metadata/Doc s	Ma
1	Pinecone	❌	✅	❌	❌	✅	❌	❌	❌	❌	❌	✅	✅	40kb	20
2	Qdrant	✅	✅	❌	❌	✅	❌	❌	✅	❌	❌	✅	✅		Unk
3	Weaviate	✅	✅	✅	✅ RR	✅	✅	❌	✅	❌	✅	✅	✅		65
4	PG Vector	✅	✅ (s	❌		✅		❌		❌	❌	❌	✅		2
5	Vespa	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅	❌	None	
6	Milvus	✅	✅	❌	❌	✅	❌	❌		❌		✅	✅		34
7	MongoDB Atlas	❌	✅	❌	❌	✅	✅	✅				✅	✅	16MB	
8	Marqo	✅	✅	✅	✅ via	✅	✅	❌		❌		✅	❌		
9	Vectara	❌	✅	❌	✅ On	✅	❌	❌		❌		✅	❌		
10	Elastic	❌	✅	✅	✅		✅	✅		❌		✅	✅	100MB	2
11	OpenSearch	✅	✅	✅	✅ Only line		✅	✅		❌		✅	✅		
12	Chroma	✅	❌	✅	❌	✅	❌	❌	❌	❌		✅	✅		
13	Vald	✅	❌					❌		❌		✅	❌		
14	Redis	❌	✅	❌				❌		❌		✅	✅		

“https://www.linkedin.com/posts/dhruv-anand-ainorthstartech_vectordatabases-vectorsearch-vectorembeddings-activity-7137357363879026688-m41R?utm_source=share&utm_medium=member_desktop”



"https://www.linkedin.com/posts/martin-ciupa-76418b17_title-9-problems-with-generative-ai-in-activity-7132823231152943105-I7f1?utm_source=share&utm_medium=member_desktop"