



# Generative AI

DAY 2

# Let understand Interacting with LLM through UI vs Code

The screenshot displays the ChatGPT web interface. On the left is a dark sidebar with navigation options: ChatGPT, ChatGPT Classic, DALL-E, Learn Anything, and Explore. Below these are sections for 'Today' (containing 'Age of Under 40'), 'Yesterday' (containing 'LLM Workflow Example'), and 'Previous 7 Days' (containing 'GPT Multimodal Title Slides', 'Extract and Export to Excel', and 'DALL-E Proud Picture'). At the bottom of the sidebar is the user's profile, 'Sivakumar Rajendran'.

The main chat area is titled 'ChatGPT 4'. It shows a conversation where the user provides a context of ages for Rachel (30), Bob (45), and Kevin (65) and asks 'Who is under 40 years old?'. ChatGPT responds with a bulleted list of the ages and then concludes that Rachel is the person under 40 years old. At the bottom of the chat area is a feedback prompt: 'Is this conversation helpful so far?' with thumbs up/down icons and a close button.

**ChatGPT 4**

**You**  
context = ""  
Rachel is 30 years old  
Bob is 45 years old  
Kevin is 65 years old  
""  
  
question = "Who is under 40 years old?"

**ChatGPT**  
Based on the provided context:

- Rachel is 30 years old
- Bob is 45 years old
- Kevin is 65 years old

The person who is under 40 years old is Rachel, as she is 30 years old.

Is this conversation helpful so far?

Message ChatGPT...

## 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.

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

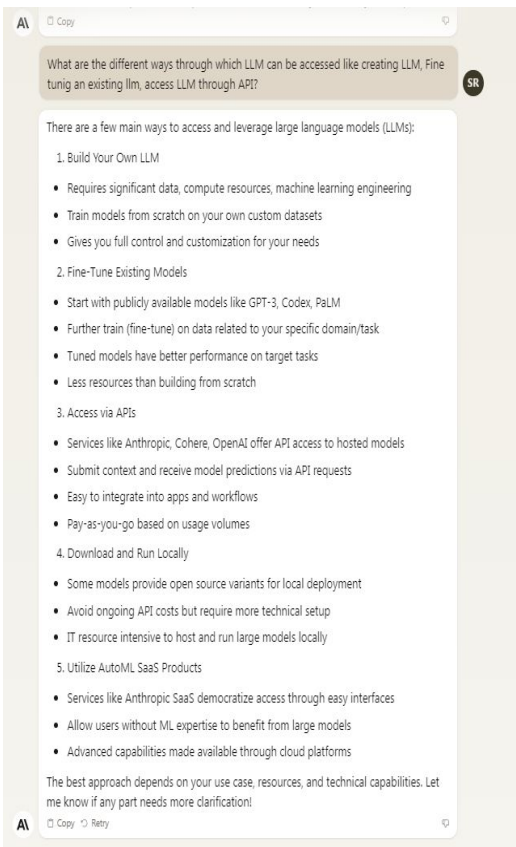
Skillset

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**

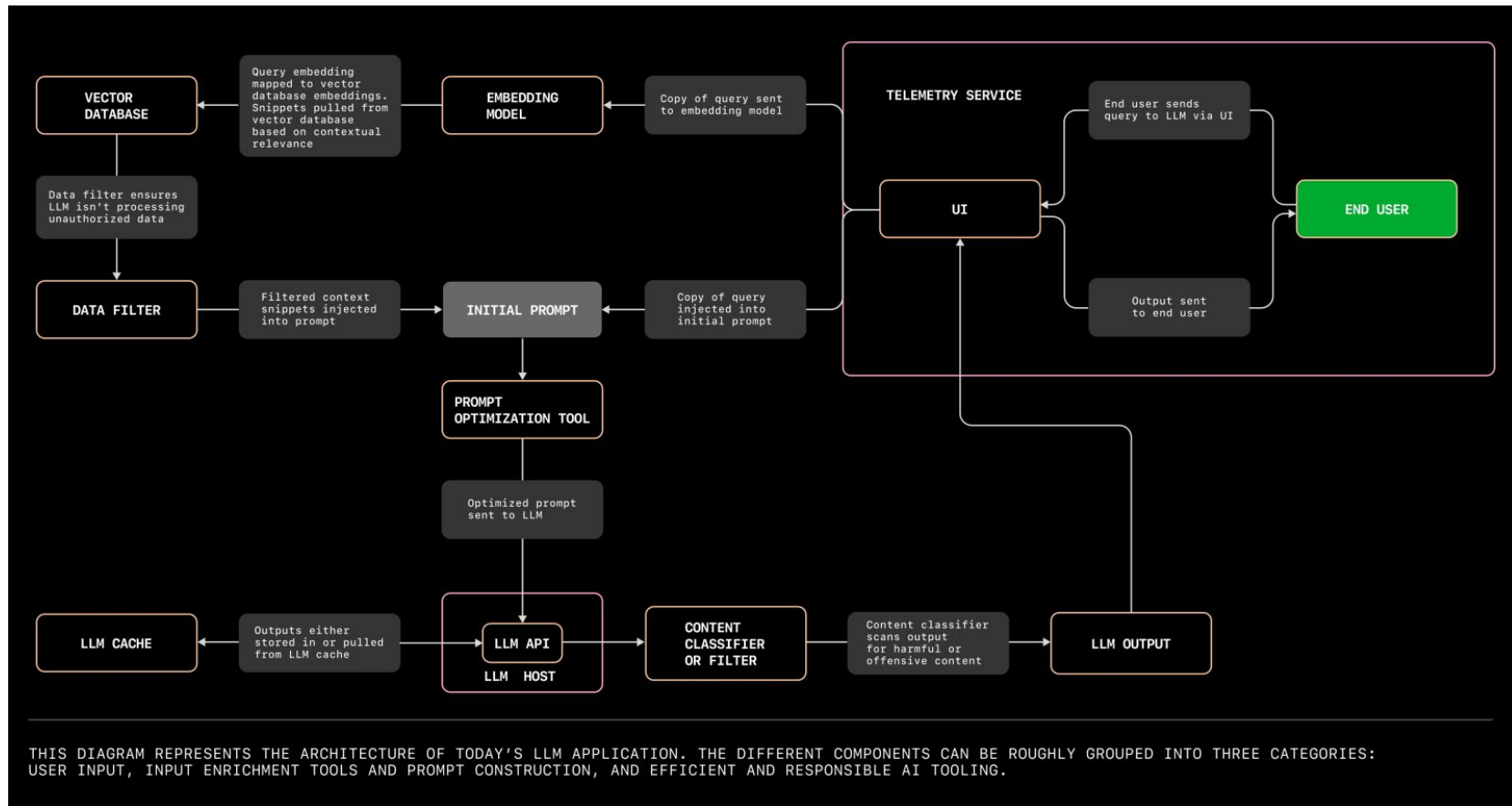
- Services like Anthropic SaaS democratize access through easy interfaces
- Allow users without ML expertise to benefit from large models
- Advanced capabilities made available through cloud platforms

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



How to build a LLM Application?

# Architecture of 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
```

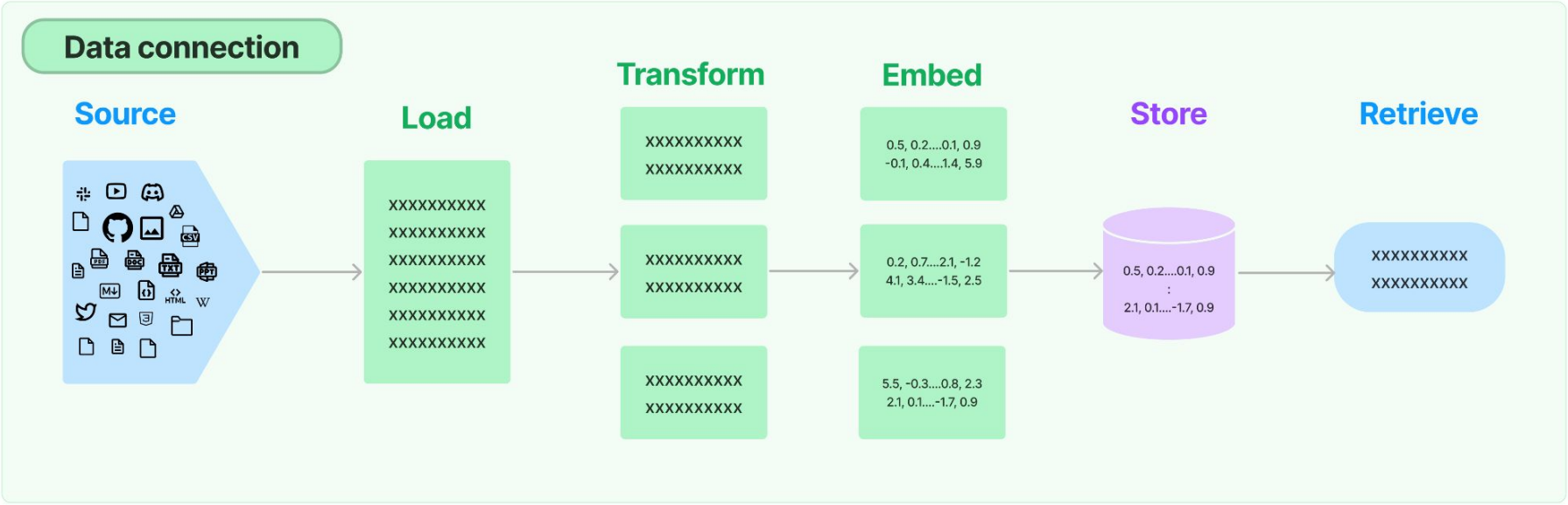
# 4. Querying

```
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



## Before get into Building blocks, let see what is Embedding

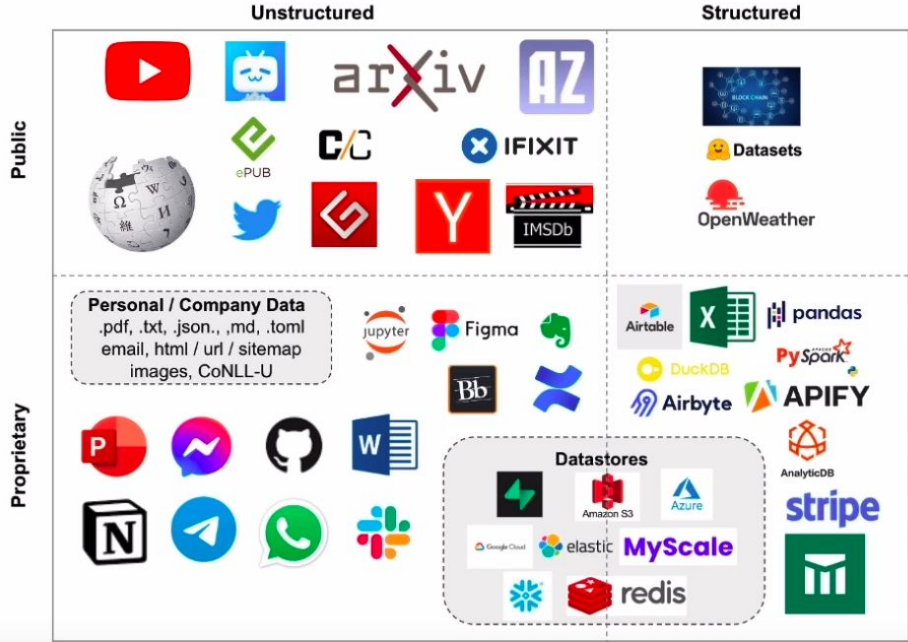
---

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	Hybrid Search	Metadata Filtering	BM25 support	Full-text Search	Multiple Search Engine	Tensor vectors per point	Streaming Index	Langchain integration	Llama index integration	Metadata/Doc e	Max
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	✗	✓	✗				✗		✗		✓	✓		

# 9 Problems with Generative AI

Generative AI tools are demonstrating massive potential. But right now, many of them are also demonstrating potential for harm.

Quality Control  
& Data Accuracy



1

## Bias in, bias out

Generative AI tools reproduce content as biased as the data they were trained on.

Ethical & Legal  
Considerations



2

## Black box

Generative AI decisions are opaque and unexplainable. They hinder accountability, trust and potentially lead to unjust outcomes.

Complexity &  
Technical Challenges



3

## Expensive

Ex-CEO of OpenAI, Sam Altman, confirmed GPT-4 cost more than \$100 million to train.

4

## Mindless parrot

Generative AI's output is tightly bound to the caliber and volume of its training data. Its output can only be as good as its training input.

5

## Alignment with human values

Generative AI lacks the capacity to model the consequences or ethical implications of its decisions.

6

## Power hungry

ChatGPT's daily queries are estimated to cost the equivalent of powering 33,000 US households.

7

## Hallucinations

Generative AI has the tendency to confidently spew inaccurate information or simply make up facts.

8

## Copyright & IP infringement

Several Gen AI models appropriated copyrighted material and intellectual property with no consent, credit, or compensation.

9

## Static

Generative AI models cannot update their knowledge in real-time or generate new ideas which may lead to misinformation.

Sources: 1. Bloomberg, 2. cpg.se, 3. Wired, 4. Lingora, 5. Roostai, 6. University of Washington, 7. Forbes, 8. PR Newswire, 9. PC Mag  
Themes sourced from Profibus