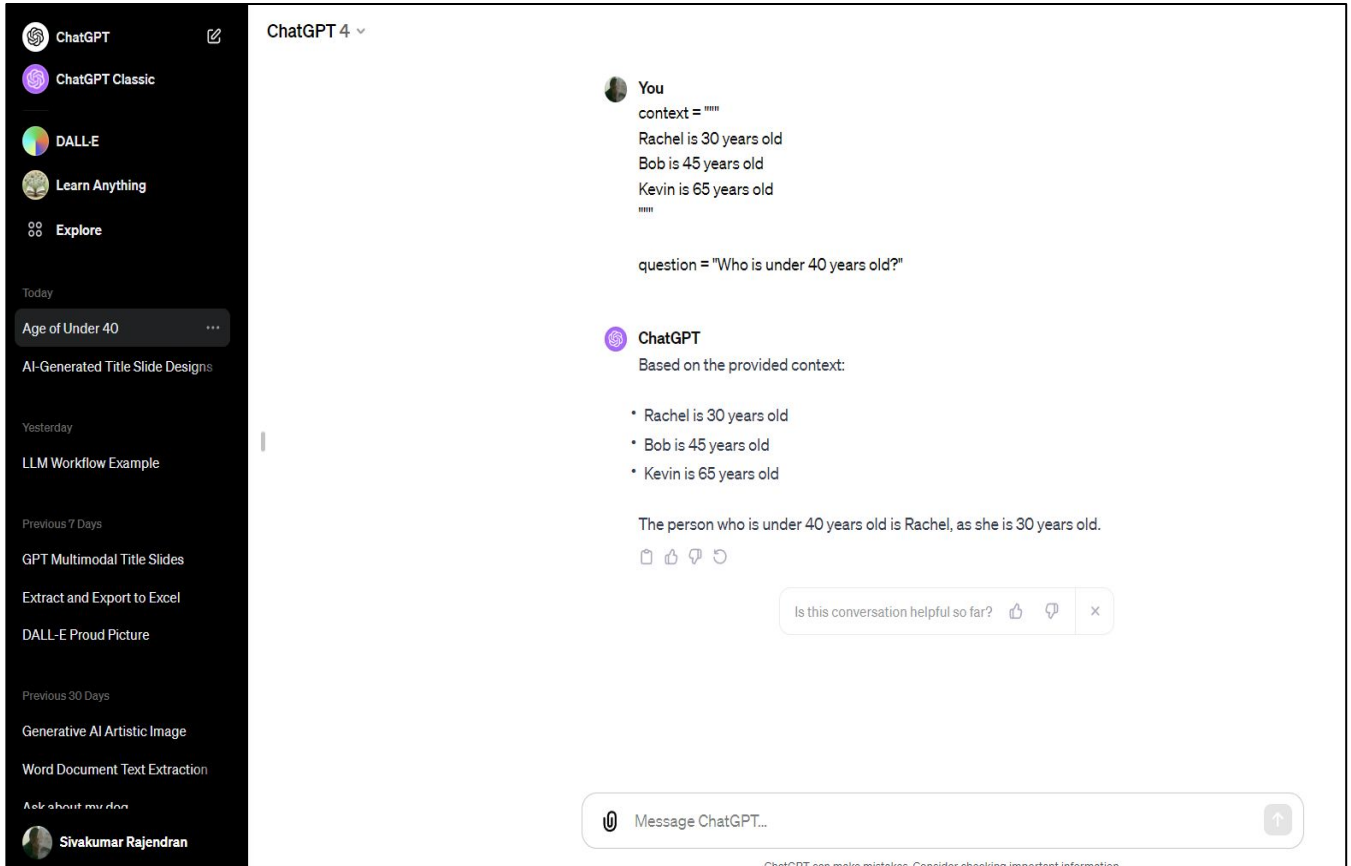

Generative AI
DAY 2

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"

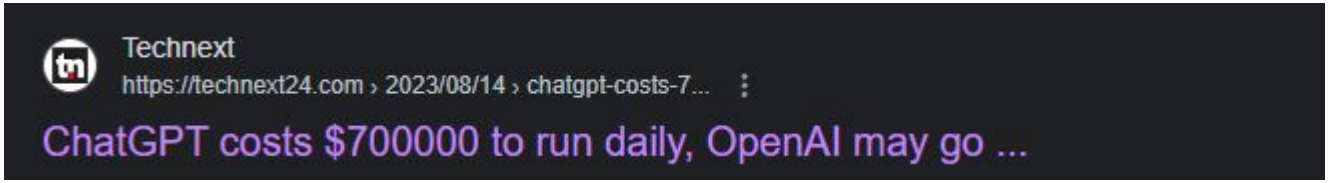
Hence, accessing the LLM through Code is not as easy as accessing through Web UI.

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.

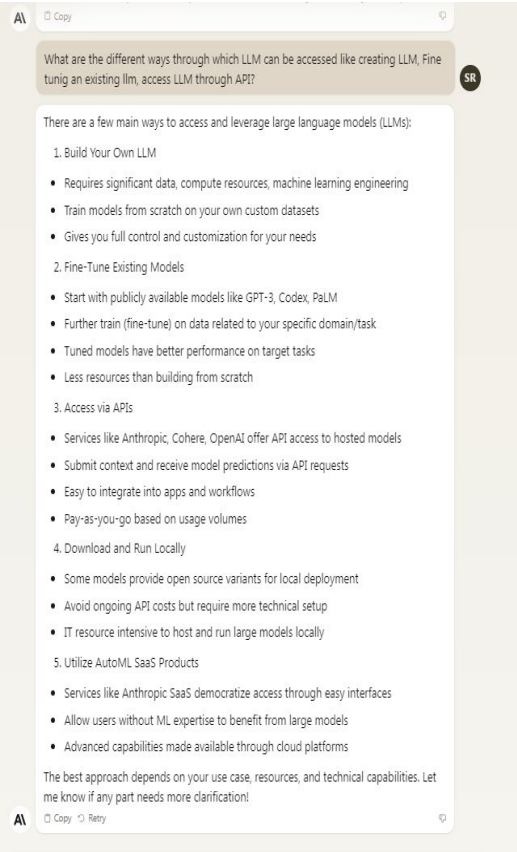


Challenges

Cost Infrastructure Skillset ROI Success/ Guaranteed Result?!

So, what are the options?

Different ways of accessing LLM



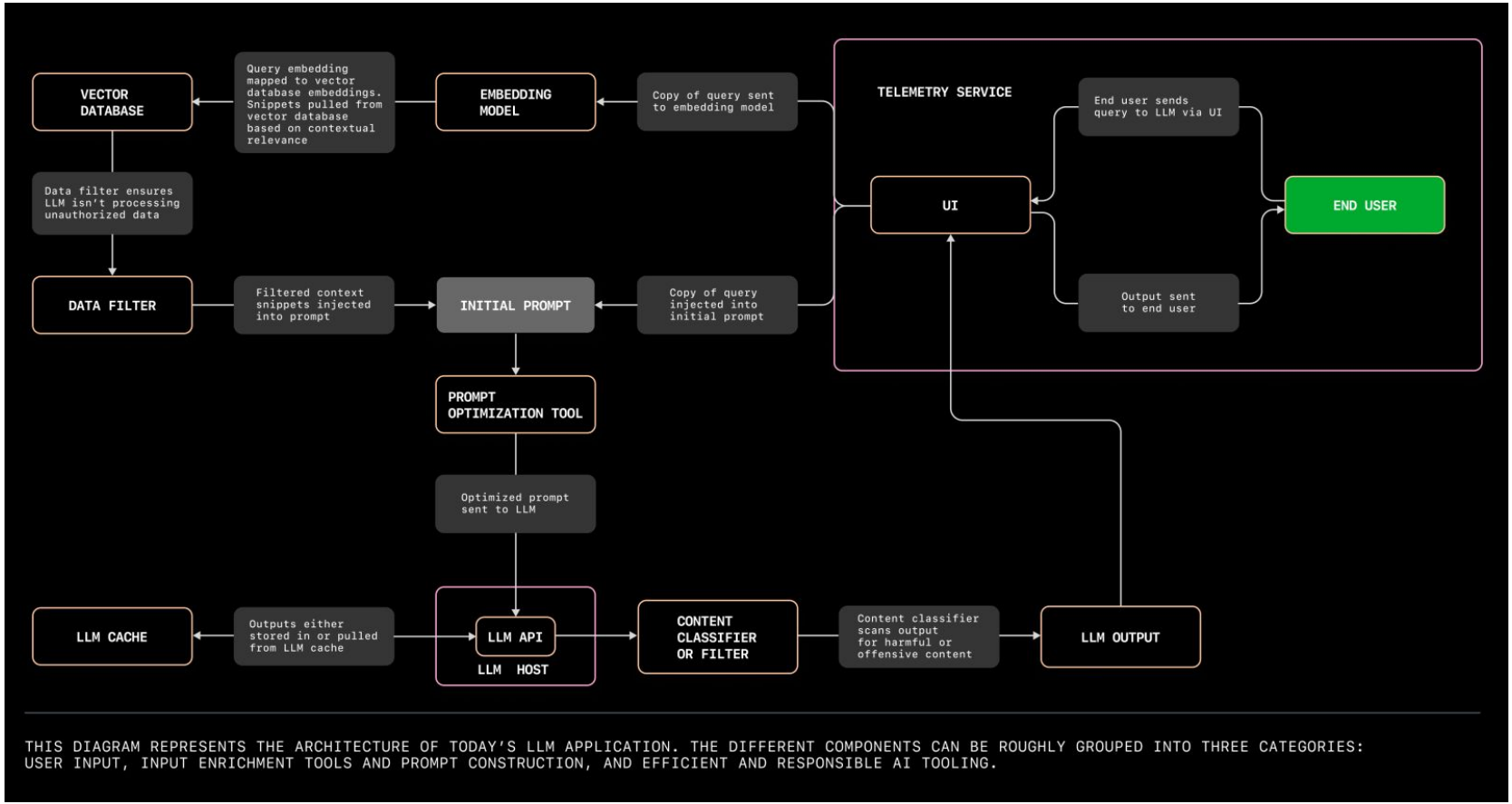
There are a few main ways to access and leverage large language models (LLMs):

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.

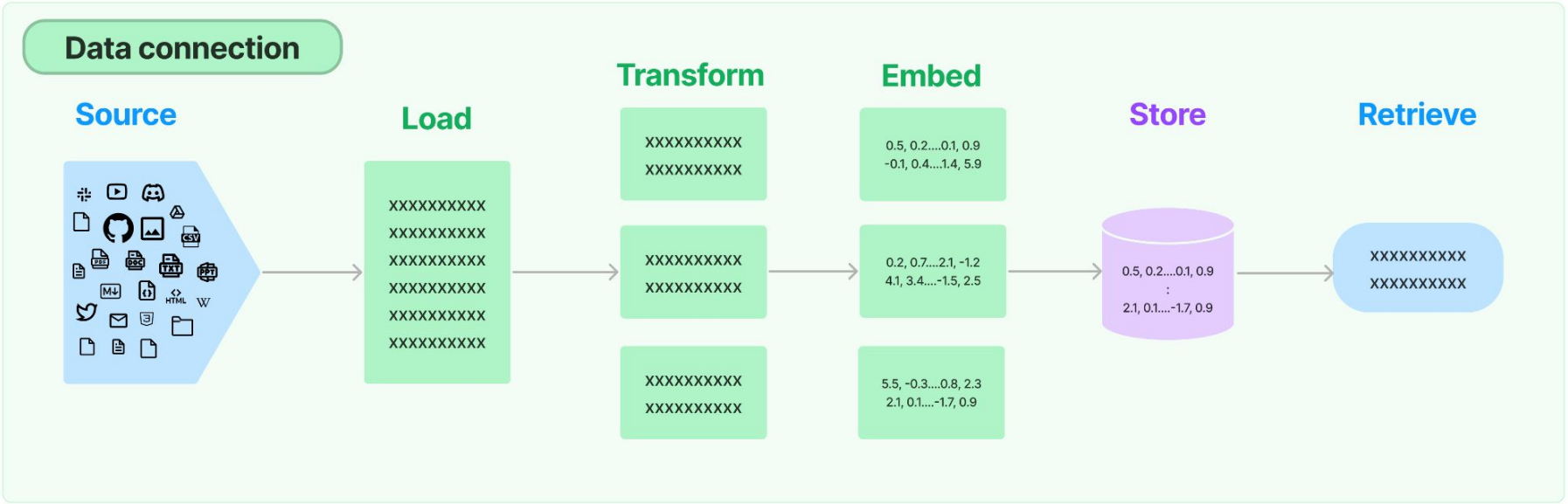
Start with a LLM code and modularize that

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/

LLM: At the heart of the system is the LLM, the core AI model responsible for generating human-like text responses.

Document Loader: With vast amounts of data to process, the Document Loader is essential. It imports and reads documents, preparing them for chunking and embedding.

Document Chunker: To make the data more manageable and efficient for retrieval, the Document Chunker breaks documents into smaller, more digestible pieces.

Embedder: Before storing or retrieving data, we need to convert textual information into a format the system can understand. The Embedder takes on this role, transforming text into vector representations.

Vector Store: This is where the magic happens. The Vector Store is a dedicated storage system that houses embeddings and their corresponding textual data, ensuring quick and efficient retrieval.

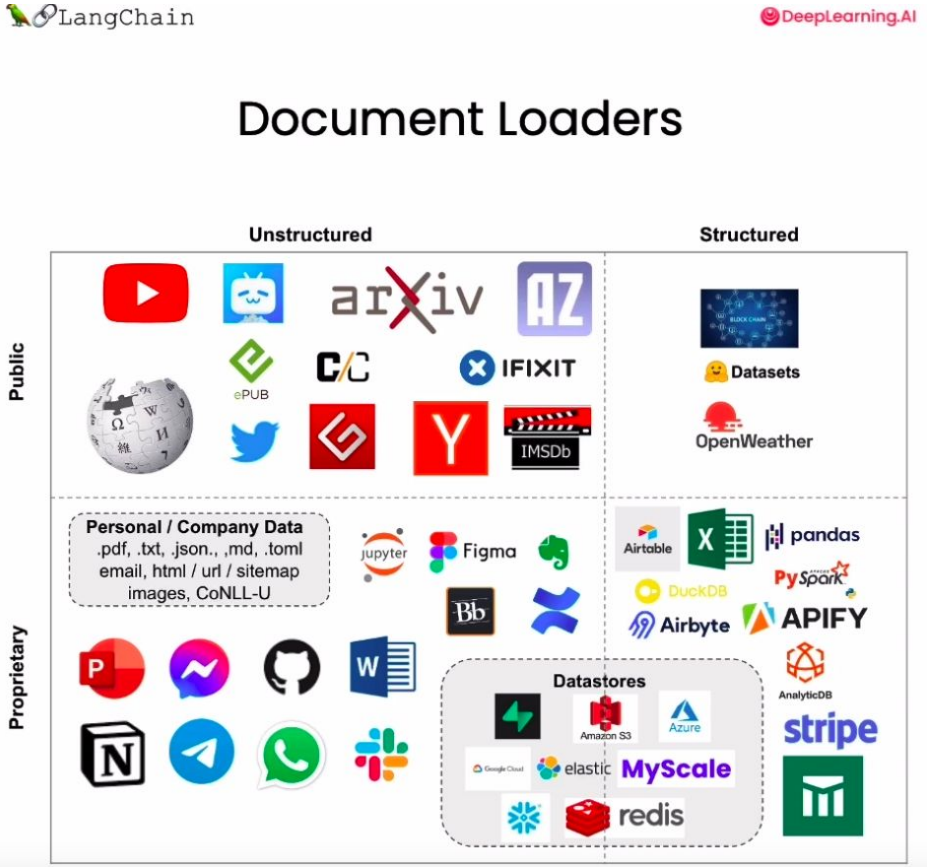
Vector Store Retriever: Think of this as the search engine of the system. The Vector Store Retriever fetches relevant documents by comparing vector similarities, ensuring that the most pertinent information is always at hand.

Prompt: Every interaction starts with a user's query or statement. The Prompt captures this initial input, setting the stage for the retrieval and generation processes.

User Input: Last but not least, the User Input tool captures the query or statement provided by the end-user, initiating the entire RAG process.

“https://github.com/amutharun/GenAIWorkshop/blob/main/RAG_A_Complete_Guide.ipynb”

Document Loaders



Word 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/"

Transformers

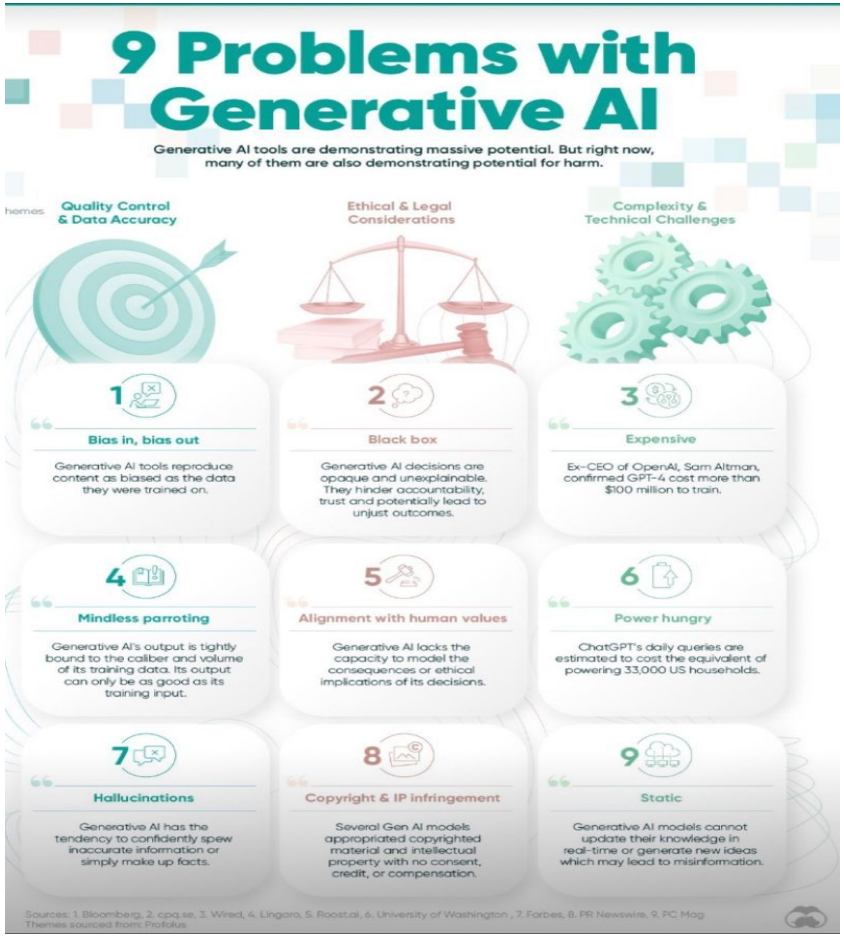
https://bbycroft.net/llm

https://www.youtube.com/playlist?list=PLoROMvodv4rNiJRchCzutFw5ItR_Z27CM

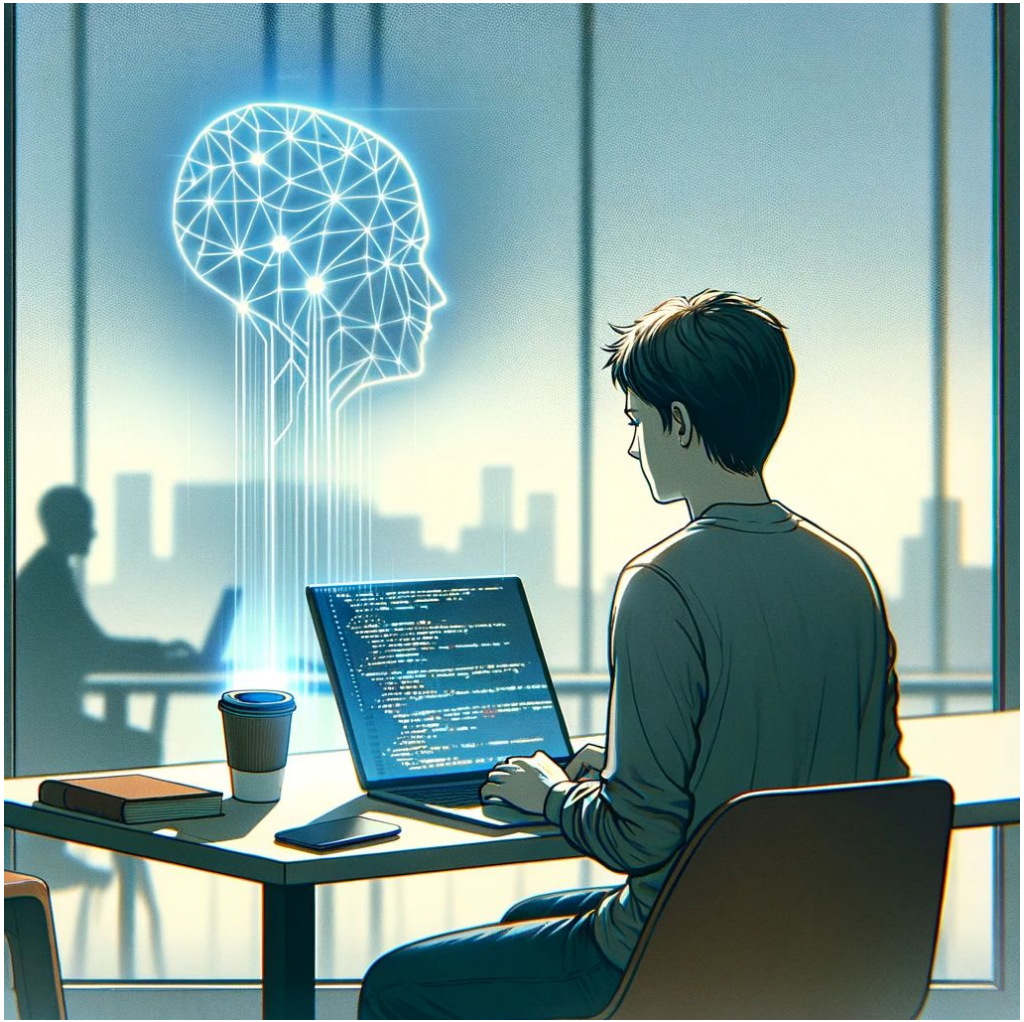
Vector Database - A Comparison

	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	✗		✓		✗		✗	✗	✓		
5	Vespa	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	None	
6	Milvus	✓	✓	✗	✗	✓	✗	✗		✗		✓	✓		34
7	MongoDB Atlas	✗	✓	✗	✗	✓	✓	✓				✓	✓	16MB	
8	Marqo	✓	✓	✓	✓via	✓	✓	✗		✗		✓	✗		
9	Vectara	✗	✓	✗	✓On	✓	✗	✗		✗		✓	✗		
10	Elastic	✗	✓	✓	✓		✓	✓		✗		✓	✓	100MB	
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-l7f1?utm_source=share&utm_medium=member_desktop"



Recommended Curriculum to become

GenAI -
Developers

&

Citizen Developers

AI & MACHINE LEARNING

ILLUSTRATOR BLAKE CALE

We're All Programmers Now

With generative AI, anyone can code. Here's

IDEA IN BRIEF

THE PROBLEM

Employees with no coding background are increasingly using generative AI and other easy-to-use software tools to build business process applications.

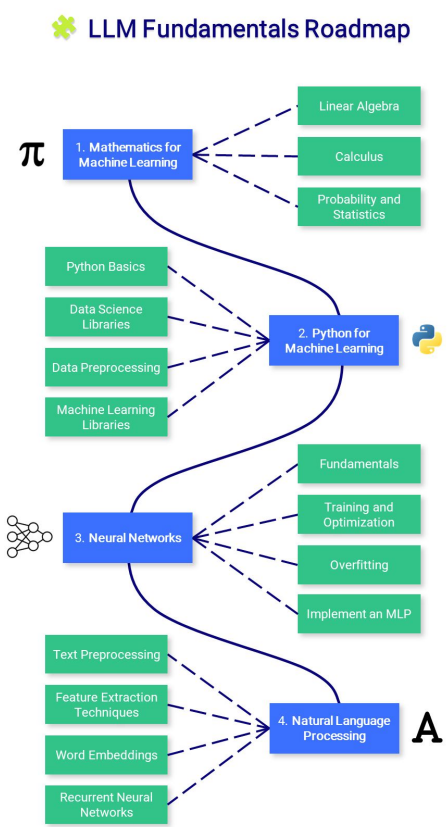
THE CONCERN

Technology experts fear that citizen developers will introduce poor-quality systems that IT teams will have to repair, or that generative AI tools will replace IT entirely.

THE SOLUTION

Businesses should proactively recruit, train, and empower citizen developers so that they are working alongside, and not against, IT departments and other lines of business.

Learning Curriculum for Beginners



Start with,
Programming Language, Libraries - Python, PyTorch

LLM - LangChain/ LlamaIndex framework, Up-to-date on Industry advancements

Statistics - Concept understanding is required to understand, improve, analyze interpret and explain the Metrics

Cloud - Working knowledge on AWS/ Azure/ GCP

DevOps - Working knowledge on Git, Docker

Want to say
Which is required - Proficient on the skill (or) High level knowledge?
Depends on the use case. There are jobs for both Applied scientist as well as Principal Scientist

How to Study - Enrolling for Masters/ PG Diploma/ Self-study?
Based on my experience, most of the Paid certifications are shallow and not equating the real-time skills. Look for freely available study resources from Stanford, edX and Industry practioners

https://github.com/mlabonne/llm-course/tree/main

Few hand-picked course if you have hands-on on Machine Learning

- Generative AI for Everyone: How generative AI works, and how to use it in your life and at work. https://lnkd.in/gfpBdEEV

- Understanding and Applying Text Embeddings: Use text embeddings to capture the meaning of sentences and paragraphs and apply them for tasks like text clustering, classification, and outlier detection using VertexAI. https://lnkd.in/gQU7fbJ4

- Large Language Models with Semantic Search: Enhance keyword search using Cohere Rerank, use embeddings to leverage dense retrieval, a powerful NLP tool, and evaluate effectiveness for further optimization. https://lnkd.in/g5RK-kYU

- LangChain for LLM Application Development: Apply LLMs to your proprietary data to build personal assistants and specialized chatbots, and Use agents, chained calls, and memories to expand your use of LLMs. https://lnkd.in/gh8EWaNa

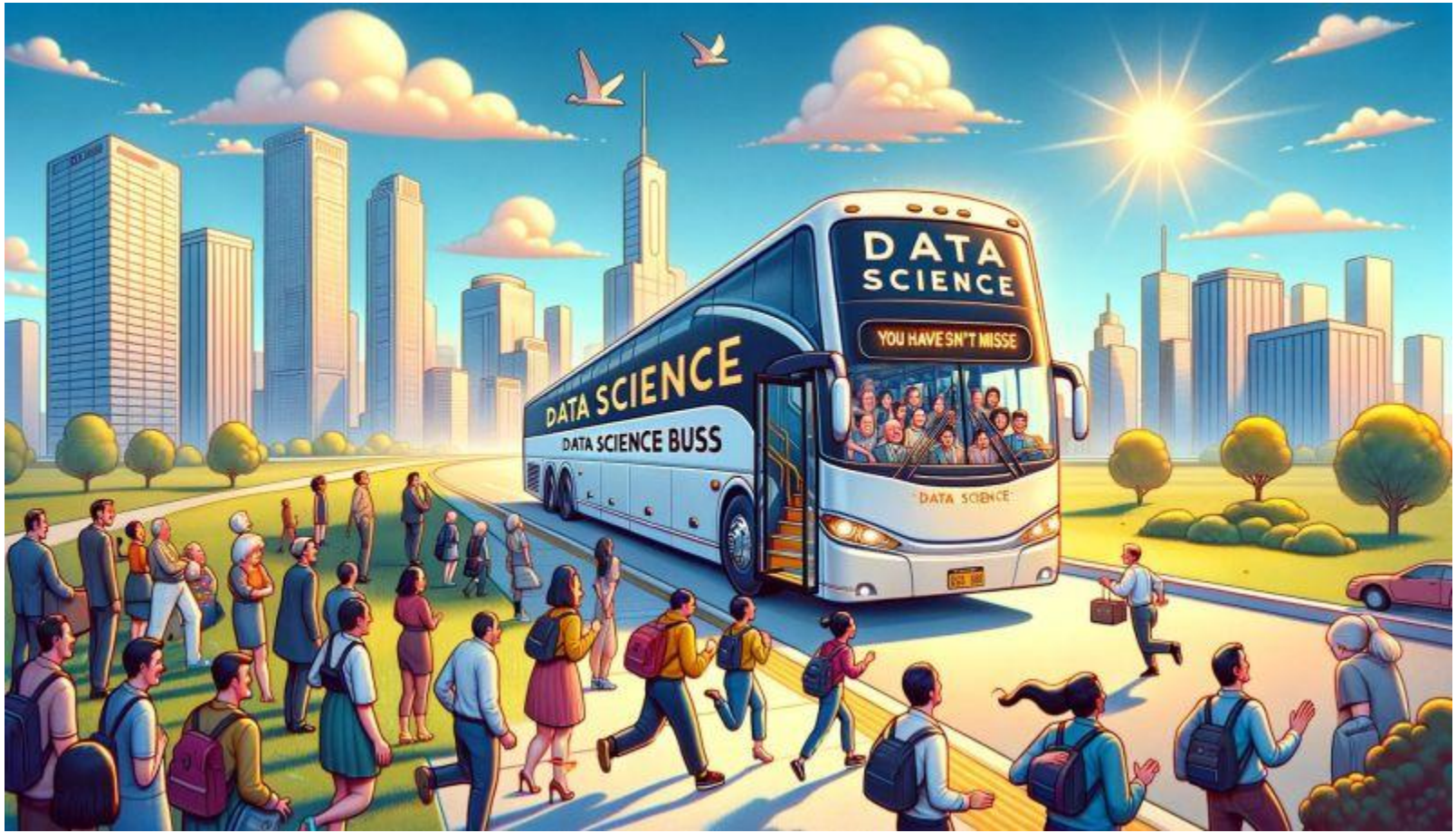
- LangChain: Chat with Your Data: Access over 80 unique loaders to handle accessing various data sources using LangChain, and build your own chatbot to chat directly with information from your own documents and data. https://lnkd.in/g4XSsmBM

- Functions, Tools & Agents with LangChain: Use LangChain Expression Language (LCEL). https://lnkd.in/gZzbeREv

- Finetuning Large Language Models: Finetune LLM, understand how finetuning differs from prompt engineering, when to use both, and get practical experience with real data sets. https://lnkd.in/gnzwKnM3

- Building and Evaluating Advanced RAG Applications: Methods like sentence-window retrieval and auto-merging retrieval, improving your RAG pipeline's performance beyond the baseline, and evaluation and best practices to streamline your process. https://lnkd.in/gwmm_HxG

https://www.linkedin.com/posts/ebrahim-alareqi-1b570048_ten-weeks-ago-a-friend-asked-me-if-i-could-activity-7140473659772534784-kEVW?utm_source=share&utm_medium=member_desktop


DATA SCIENCE
YOU HAVE SN'T MISSE
DATA SCIENCE
DATA SCIENCE BUSS
DATA SCIENCE