
AI BOOTCAMP
FOR DEVELOPERS

EXPERIENCE BUILDING AND LECTURING AT

hp accenture DELL Technologies ماجد الفطيم MAJID AL FUTTAIM Microsoft

Software Engineer
2005 - 2010

Data Scientist
2012 - 2012

2010

Data Scientist
2012 - 2014

Lead Data Scientist
2014 - 2021

Data Science Manager
2021 - 2022

Founder
2022 - Present

HARVARD BUSINESS REVIEW Yale GAP

Agenda

- AI Application Architecture
- Understanding APIs and REST
- Calling/using APIs
- Building Gen AI Apps Steps
- LLM
- Gen AI Configurations like Temperature
- Running Telecom Custom Support LLM Application with groq
- - with no prompt Template and default Gen AI Configuration
- - with prompt template and proper Gen AI Configuration
- Benefits of Prompt Template

Agenda

- Limitation of LLM
- When to use RAG
- 5 stages of RAG
- RAG Architecture
- Vector Embedding
- Demo of Vector embedding 3D
- Building your First RAG using llamaindex
- Vector Stores Type
- Pinecone
- Project – Social Media Regulation UAE - gradio

Agenda

- Vector Stores Type
- Pinecone
- Project – Social Media Regulation UAE - gradio
- Introduction to llama Parse
- Langchain
- Components of Langchain
- Message & Chain
- Prompt Template
- Conversation History
- Introduction of Tools/AI Agents

Projects

Building AI Application with groq/ Prompt template

Building Application Basic RAG using open AI

Building Application Advanced RAG using llamaindex

Building and Deploying AI Applications with Hugging Face

AI Application Architecture

Google chrome

Web UI

https://chat.openai.com/chat

curl

Server

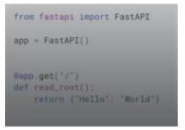
API

UI

API

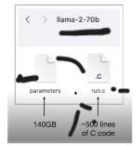
llama
gpt-4o
gpt-3.5-turbo

Backend/Server

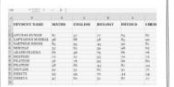


API

AI Models



Data Stores



Custon UI

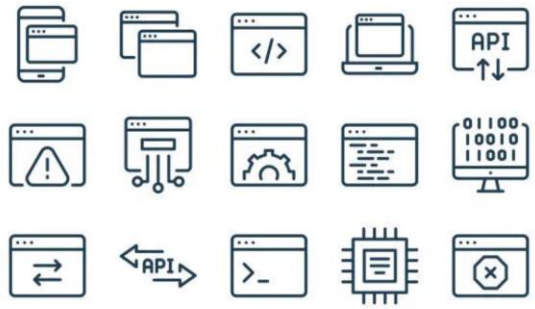
User Interface

API
agents
gradio
streamlit

gradio playground

Calling/using APIs

- Calling API using Postman (API Testing)
- Calling API using python (API Integration)
- Calling API using SDK (API Integration)
- Building chatbot in 5 mins.
- Integrate gradio chat UI with LLM
- Deployed it on HF spaces



Understanding APIs and REST

Your Gateway to Web & App Integration

HTTP

Client

Request

Response

Server

Add new Employee

HTTP POST Request

RequestMethod = POST

Employees API

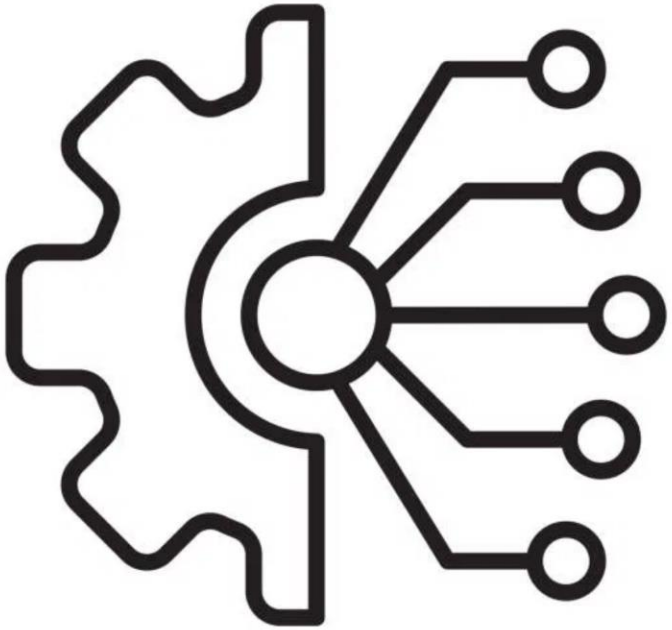
- Modify the underlying data -

- Create new resources -

What is an API?

Application Programming Interface is a set of rules that allows different software applications to communicate with each other.

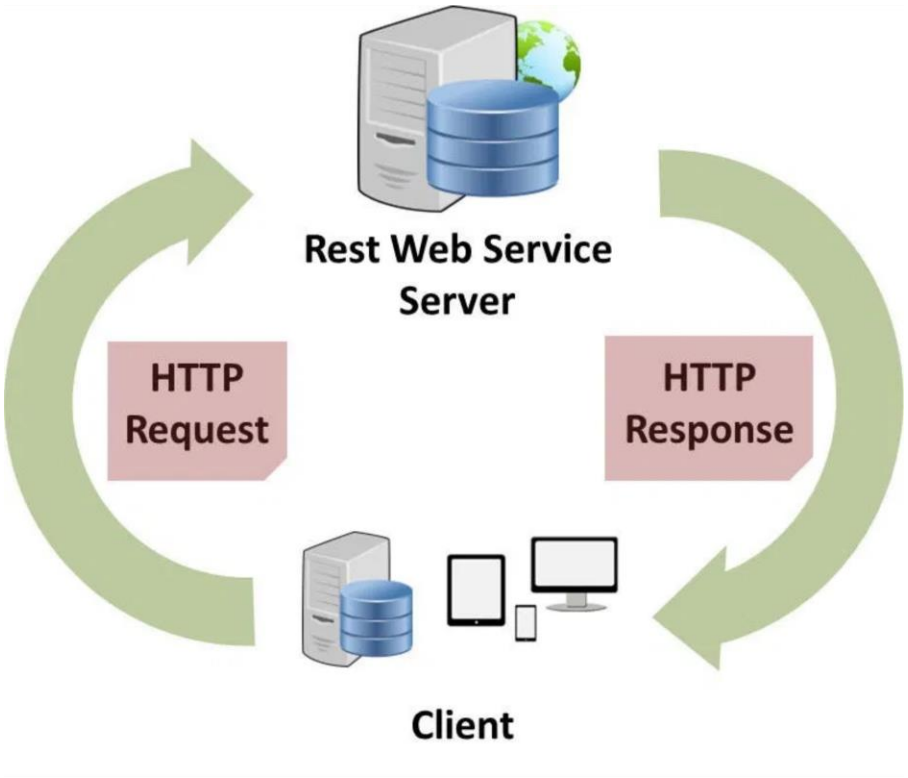
Think of an API as a **restaurant menu** : it presents what's available, you order (request), and receive your food (response) without needing to know how it's prepared.



Why REST?

REST (Representational State Transfer) is an architectural style that uses standard HTTP methods.

It's the default for web integrations because it's **stateless** , **scalable** , and **simple** to implement with existing web infrastructure.

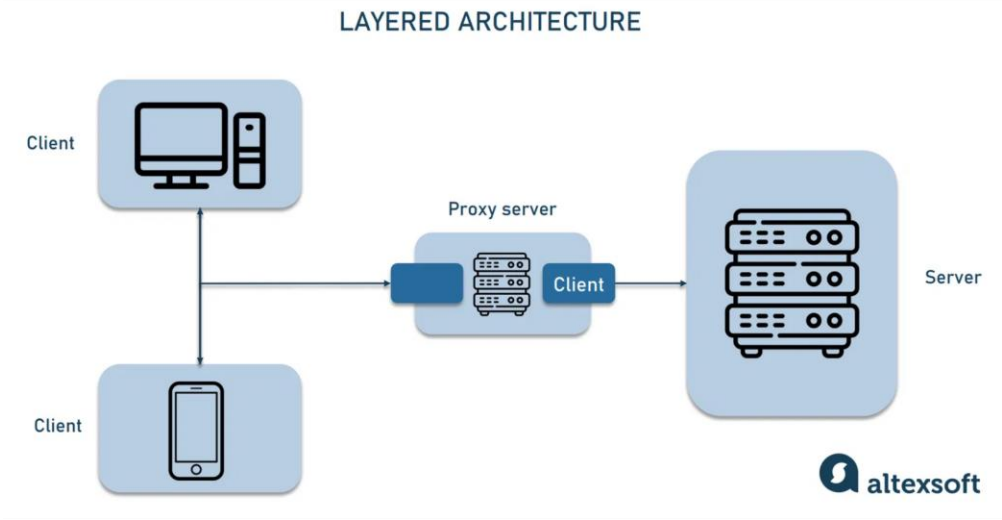


Anatomy of an API Spec

Base URL: The root address of the API

Endpoints: Specific paths for different resources

Methods: GET, POST, PUT, DELETE operations

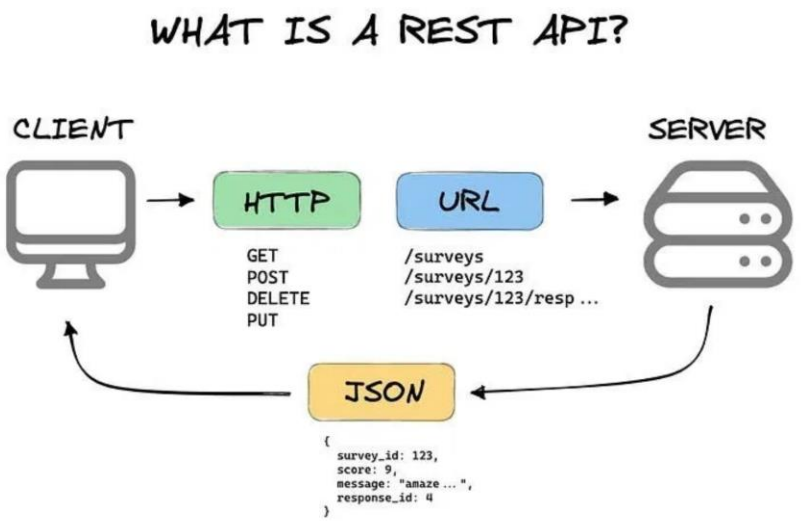


API Spec: Parameters & Headers

Parameters: Query, Path, Body

Headers: Metadata about the request

Authentication: API keys, OAuth, JWT



Introducing Groq

What is Groq?: A low-latency inference platform powered by LPUs (Language Processing Units) built for fast, predictable token generation.

Why it matters: Millisecond-level first-token latency and high throughput → ideal for real-time apps (agentic chat, voice, tooling).

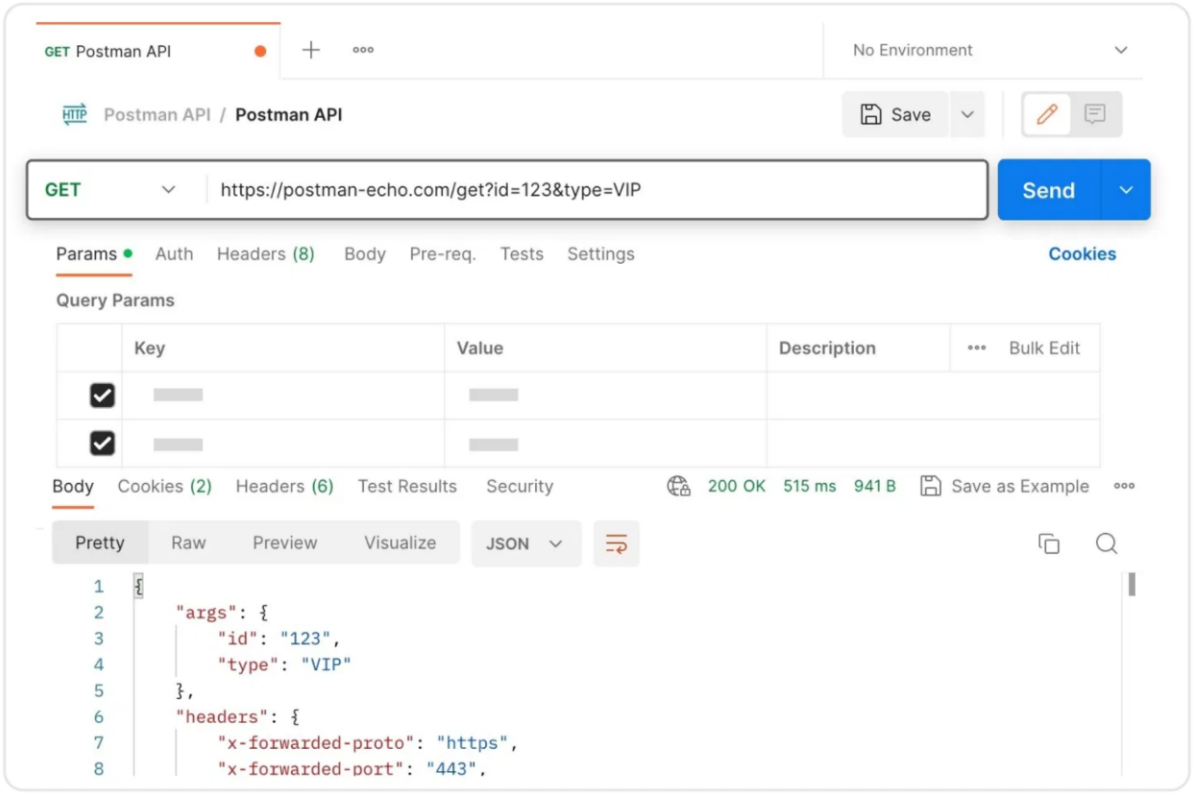
Key strengths: Speed & determinism, simple OpenAI-compatible API, pay-as-you-go usage, great for streaming & multi-turn chats.

Models commonly served: Open-source LLMs (e.g., Llama-3 family, Mixtral/Gemma variants) and fast embedding models; catalog evolves frequently.

Great fits: Live agent assist, customer support widgets, code copilots, rapid RAG answers, mobile/web chat UIs.

Making API Requests with Postman

1. Enter the endpoint URL
2. Select the HTTP method
3. Send and view the JSON response



Create chat completion

POST https://api.groq.com/openai/v1/chat/completions

Creates a model response for the given chat conversation.

Request Body

messages array Required
A list of messages comprising the conversation so far.
Show possible types

model string Required
ID of the model to use. For details on which models are compatible with the Chat API, see available models

compound_custom object or null Optional
Custom configuration of models and tools for Compound.
Show properties

disable_tool_validation boolean Optional Defaults to false
If set to true, groq will return called tools without validating that the tool is present in request.tools. tool_choice=required/none will still be enforced, but the request cannot require a specific tool be used.

curl

```
curl https://api.groq.com/openai/v1/chat/completions -s \
-H "Content-Type: application/json" \
-H "Authorization: Bearer $GROQ_API_KEY" \
-d '{
  "model": "llama-3.3-70b-versatile",
  "messages": [{
      "role": "user",
      "content": "Explain the importance of fast language models
  }]
}'
```

Example Response

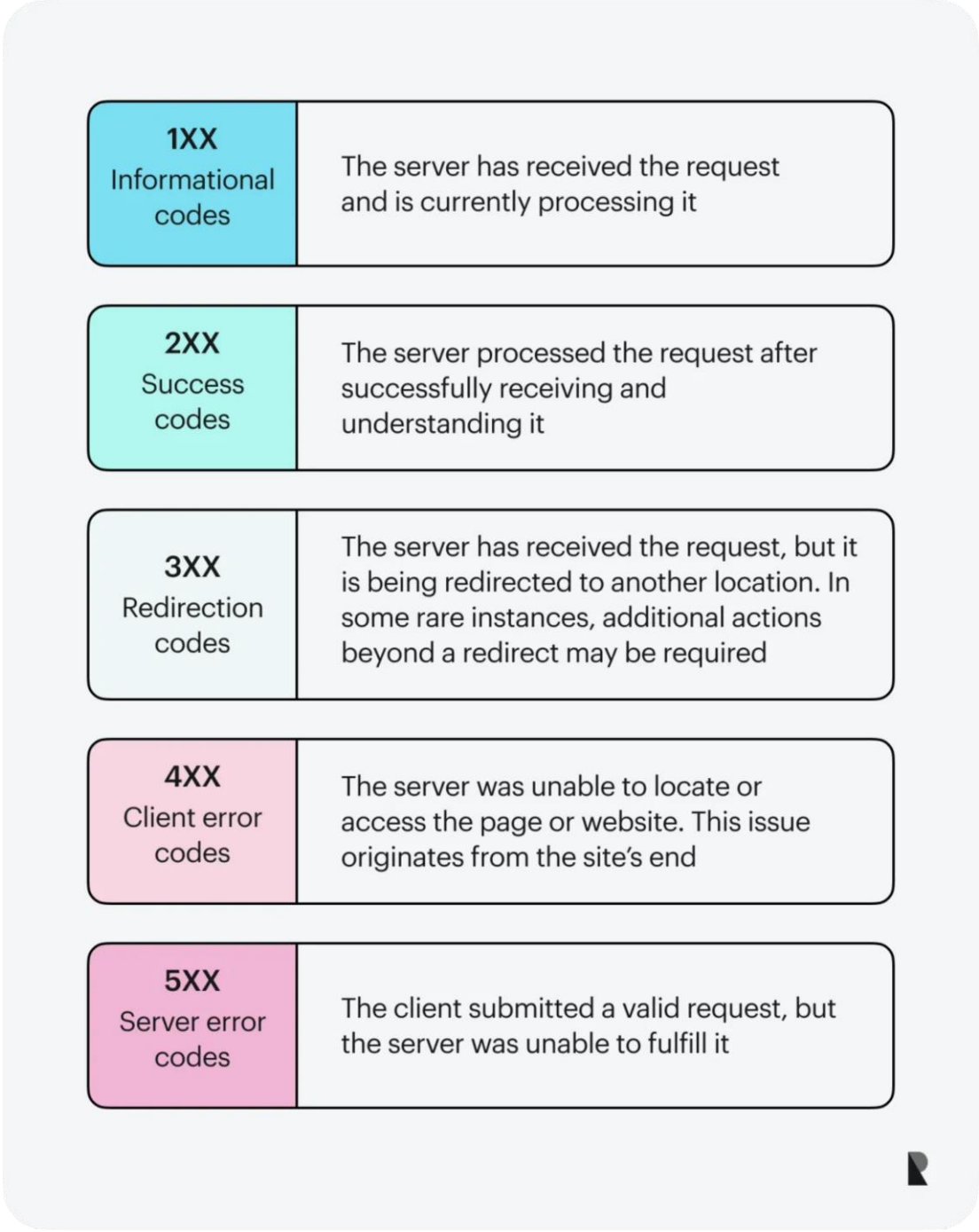

```
{
  "id": "chatcmpl-f51b2cd2-bef7-417e-964e-a08f0b513c22",
  "object": "chat.completion",
  "created": 1730241104,
  "model": "openai/gpt-oss-20b",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "Fast language models have gained significant
      },
      "logprobs": null
```

Understanding HTTP Status Codes

2xx Success: 200 OK, 201 Created

4xx Client Error: 400 Bad Request, 401 Unauthorized, 404 Not Found

5xx Server Error: 500 Internal Server Error



Calling/using APIs

1. Calling API using Postman (API Testing) -
2. Calling API using python (API Integration)
3. Calling API using SDK (API Integration)
4. Building chatbot in 5 mins.
5. Integrate gradio chat UI with LLM
6. Deployed it on HF spaces

Calling/using APIs

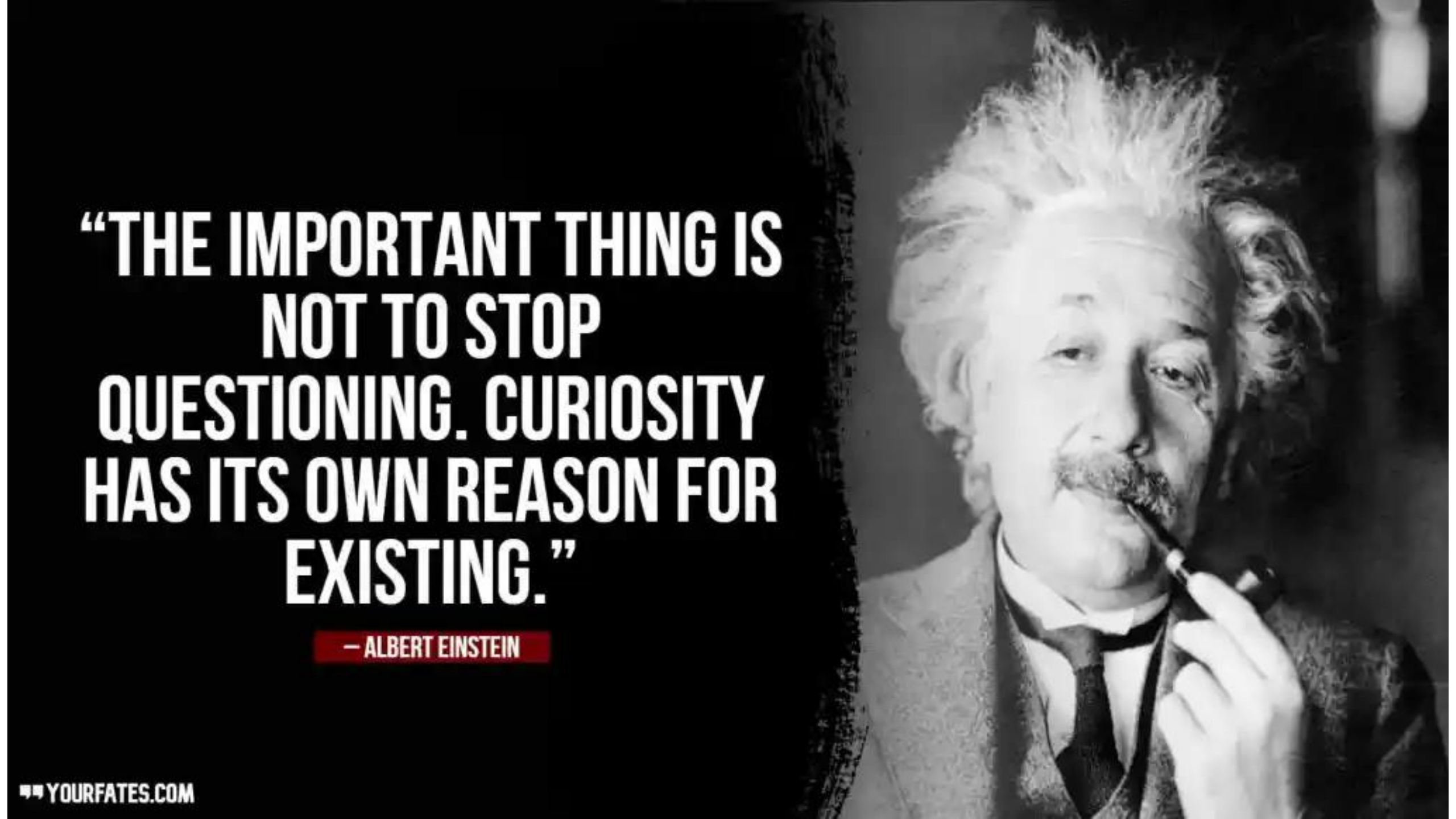
1. Calling API using Postman (API Testing) -
2. Calling API using python (API Integration)
3. Calling API using SDK (API Integration)
4. Building chatbot in 5 mins.
5. Integrate gradio chat UI with LLM
6. Deployed it on HF spaces

Calling/using APIs

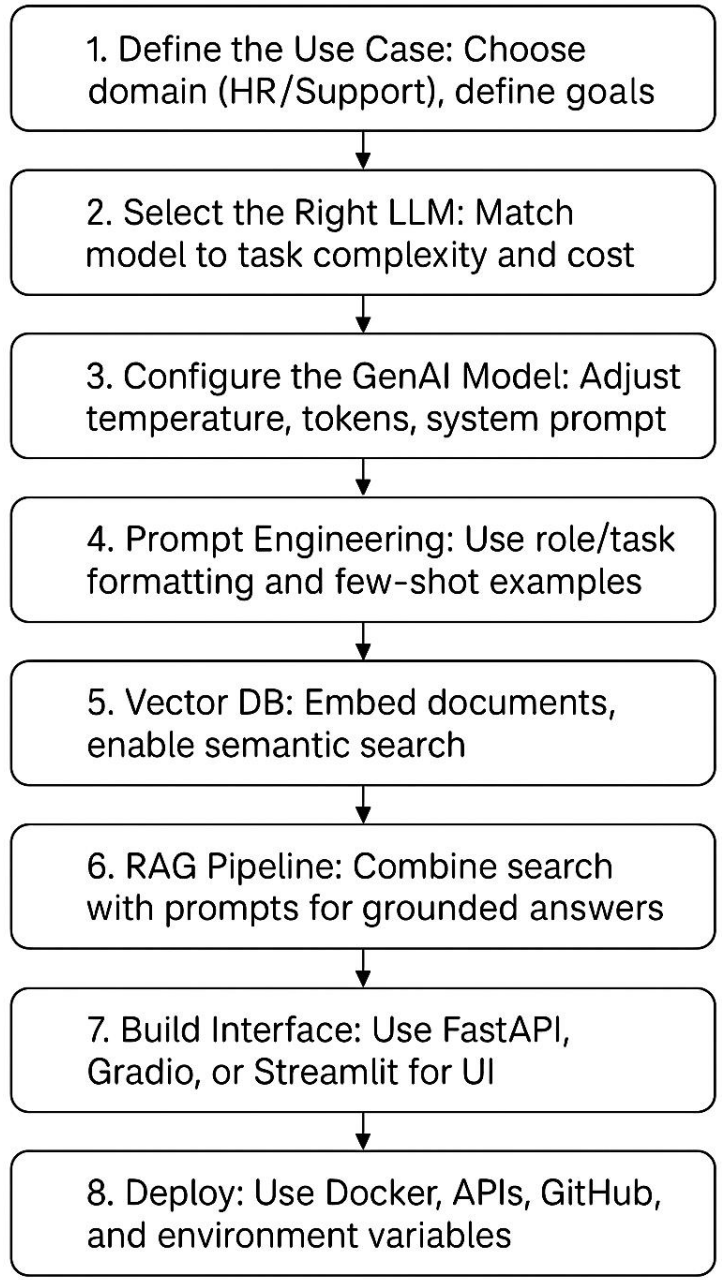
1. Calling API using Postman (API Testing) -
2. Calling API using python (API Integration)
3. Calling API using SDK (API Integration)
4. Building chatbot in 5 mins.
5. Integrate gradio chat UI with LLM
6. Deployed it on HF spaces

Calling/using APIs

1. Calling API using Postman (API Testing) -
2. Calling API using python (API Integration)
3. Calling API using SDK (API Integration)
4. Building chatbot in 5 mins.
5. Integrate gradio chat UI with LLM
6. Deployed it on HF spaces


“THE IMPORTANT THING IS NOT TO STOP QUESTIONING. CURIOSITY HAS ITS OWN REASON FOR EXISTING.”
– ALBERT EINSTEIN
YOURFATES.COM

Summary: GenAI Workshop Steps



Top LLM

Model 243 / 243	Overall	Hard Prompts	Coding	Math	Creative Writing	Instruction Following	Longer Query	Multi-Turn
claude-opus-4-1-…	1	1	1	1	1	1	1	1
gemini-2.5-pro	1	2	3	1	1	1	2	1
chatgpt-4o-lates…	2	4	3	14	2	5	4	1
claude-opus-4-1-…	2	1	1	1	1	1	1	1
gpt-4.5-preview-…	2	5	4	8	1	4	3	1
gpt-5-high	2	2	3	1	7	5	13	7
o3-2025-04-16	2	4	4	1	8	6	14	7
qwen3-max-preview	3	3	3	1	7	4	3	3
gpt-5-chat	5	4	3	7	5	5	4	1
claude-opus-4-20…	8	4	2	4	2	2	2	7
grok-4-0709	8	11	16	1	5	6	5	7
grok-4-fast	8	4	3	-	7	6	3	2
qwen3-235b-a22b-…	8	4	3	2	8	6	5	7
deepseek-v3.1	9	9	7	4	7	8	5	
deepseek-r1-0528	10	10	4	8	7	16	14	
deepseek-v3.1-th…	10	5	4	2	5	5	3	

Top OpenAI Models

Model	Category	Ideal For	Strengths	Input / Output Cost
GPT-5	Flagship	Complex reasoning, coding, agents	Flagship accuracy & capability	$1.25 / $10.00
GPT-5 mini	Flagship	Well-defined tasks, high throughput	Faster & cheaper than GPT-5	$0.25 / $2.00
GPT-5 nano	Flagship	Summarization, classification	Fastest & lowest cost	$0.05 / $0.40
gpt-realtime (text)	Realtime	Low-latency chat/agents	Streaming, concurrency	$4.00 / $16.00
gpt-realtime (audio)	Realtime	Speech-to-speech, voice apps	Multimodal audio	$32.00 / $64.00
GPT-4o mini (text)	Multimodal	Multimodal apps	Images/audio, low cost	$0.60 / $2.40
GPT-4o mini (audio)	Multimodal	Voice agents, bots	Low-latency audio I/O	$10.00 / $20.00

Top Opensource llm (groq)

MODEL ID	DEVELOPER	CONTEXT WINDOW (TOKENS)	MAX COMPLETION TOKENS	MAX FILE SIZE	DETAILS
`llama-3.1-8b-instant`	Meta	131,072	131,072	-	Details
`llama-3.3-70b-versatile`	Meta	131,072	32,768	-	Details
`meta-llama/llama-guard-4-12b`	Meta	131,072	1,024	20 MB	Details
`openai/gpt-oss-120b`	OpenAI	131,072	65,536	-	Details
`openai/gpt-oss-20b`	OpenAI	131,072	65,536	-	Details
`whisper-large-v3`	OpenAI	-	-	100 MB	Details
`whisper-large-v3-turbo`	OpenAI	-	-	100 MB	Details

Which OpenAI Model Should I Use? (Interactive Quiz Style)

Do you need top-tier reasoning or multimodal (text+image+audio) input?

✅ If YES: Use GPT-5 or o3 reasoning models

➡️ If NO: Use GPT-5 mini or nano for lighter tasks

Model Selection Question

Are you building a voice-based chatbot or assistant?

✅ If YES: Use GPT-4o mini (audio) — optimized for real-time voice

➡️ If NO: Use GPT-5 mini or GPT-4.1 mini for chat

Model Selection Question

Do you need to process or generate images?

✅ If YES: Use GPT-image-1 for text-to-image or editing tasks

➡️ If NO: Choose a text-only model like GPT-5 or GPT-5 mini

Model Selection Question

Do you want to fine-tune the model for your domain (e.g., legal, support)?

✅ If YES: Use GPT-4.1 FT or o4-mini RFT for domain-specific control

➡️ If NO: Use a base model like GPT-5 mini or nano

Model Selection Question

Do you need to run AI offline or with full data privacy?

✅ If YES: Use GPT-OSS (open-weight models for local inference)

➡️ If NO: Use hosted models via OpenAI API for convenience and tool access

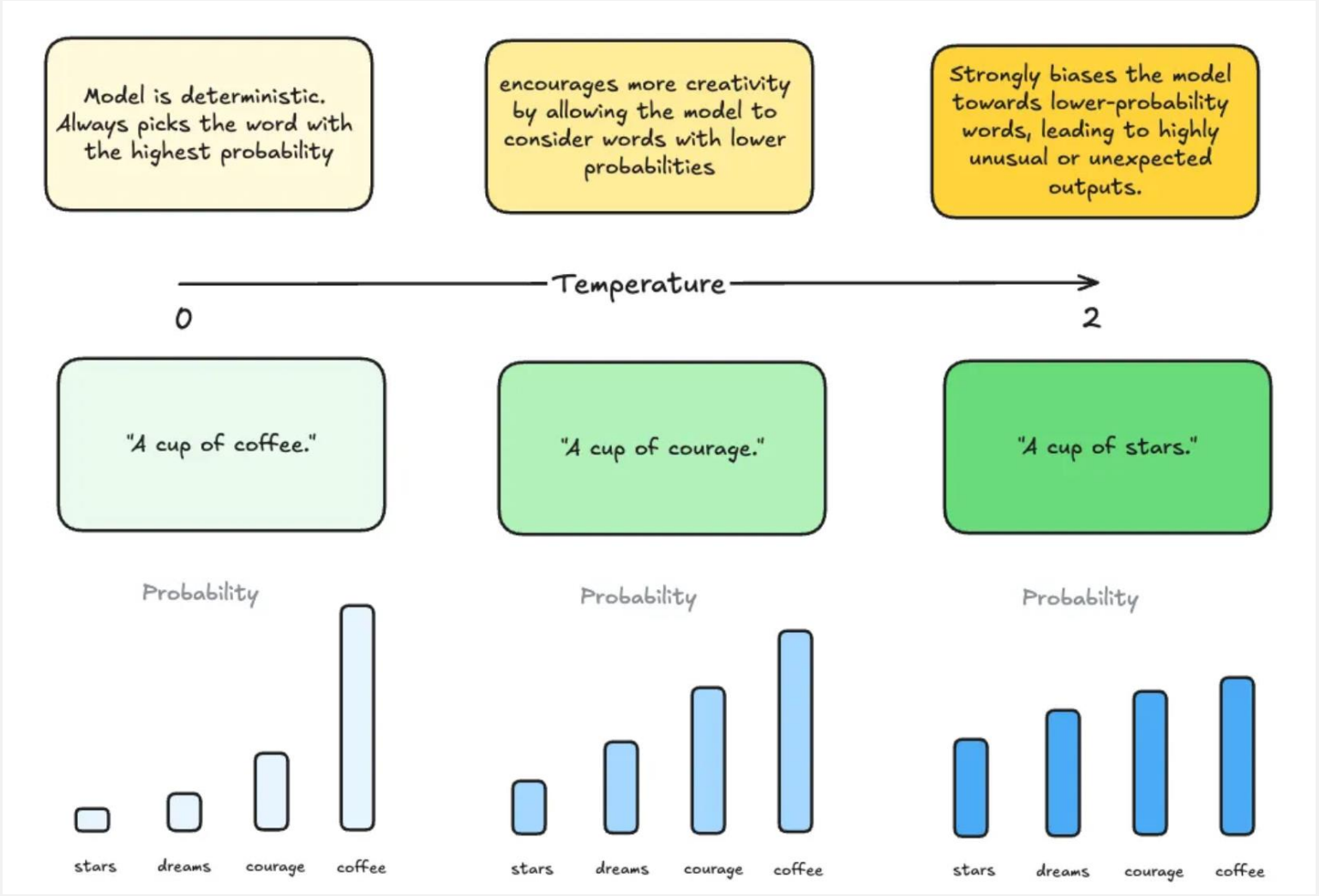
Model	Category	Ideal For	Strengths	Input / Output Cost (1 milion)
GPT-image-1 (text→image)	Image	Image generation	High-fidelity visuals	$5.00 / $40.00
GPT-image-1 (image prompts)	Image	Image-to-image	Strong editing control	$10.00 / $40.00
GPT-4.1 (FT)	Fine-tuned	Custom enterprise tasks	High fidelity via FT	$3.00 / $12.00
GPT-4.1 mini (FT)	Fine-tuned	Cost-sensitive apps	Great price/perf via FT	$0.80 / $3.20
GPT-4.1 nano (FT)	Fine-tuned	Ultra-cheap tailored tasks	Lowest FT inference cost	$0.20 / $0.80
o4-mini (RFT)	Fine-tuned	Policy-tuned agents	RL control tuning	$4.00 / $16.00
GPT-5 (Batch)	Batch	Async jobs	~50% cheaper	$0.625 / $5.00
GPT-5 mini (Batch)	Batch	High-volume async	~50% cheaper	$0.125 / $1.00

https://decodingdatascience.com/openai-guide-to-use-which-model-for-tasks/

Top LLM

Model	Developer	Context Window	Max Output Tokens	Ideal Use Cases
Gemma2-9B-IT	Google	8,192 tokens	Not specified	General-purpose tasks requiring moderate context
LLaMA-3.3-70B-Versatile	Meta	128K tokens	32,768 tokens	Complex reasoning, multilingual applications
LLaMA-3.1-8B-Instant	Meta	128K tokens	8,192 tokens	Fast response tasks, chatbots
LLaMA3-70B-8192	Meta	8,192 tokens	Not specified	Tasks with shorter context requirements
LLaMA3-8B-8192	Meta	8,192 tokens	Not specified	Lightweight applications with limited context

LLM Hyperparameters: Temperature, Top-k, Top-p, and Frequency & Presence Penalties



Choosing the Right Hyperparameters

- Temperature: Controls creativity and randomness.
- Top-k & Top-p: Define the pool of next word choices based on probability.
- Penalties: Manage word repetition to balance diversity and consistency.

Optimization: Tailor hyperparameters based on the specific needs of your task.

Parameter	Purpose	Why It Matters
temperature	Controls randomness in output. Lower = more focused. Higher = more diverse.	Use lower values for customer support to ensure clear, consistent, and reliable responses.
top_p	Controls diversity using **nucleus sampling**.	Keeps sampling within a high-probability set of responses; useful for slight variation without drift.
max_tokens	Limits the total length of the generated response.	Ensures responses are concise and prevents overly long or incomplete replies.
frequency_penalty	Reduces repetition of tokens in the output.	Helps avoid redundant phrasing, especially in step-by-step instructions.
presence_penalty	Encourages introduction of new topics or ideas.	Useful in support roles to gently suggest additional solutions beyond the obvious.

2nd Workshop/Demo

https://github.com/Decoding-Data-Science/omantel

Parameter	Value	Why This Value?
temperature	0.7	Balances creativity with reliability. Keeps responses empathetic yet consistent.
top_p	0.95	Allows for slightly diverse token sampling while ensuring quality responses.
max_tokens	300	Provides enough room for multi-step, detailed telecom support responses.
frequency_penalty	0.2	Minimizes repetitive answers in procedural steps (e.g., “please restart your device”).
presence_penalty	0.3	Slightly encourages adding new ideas (e.g., checking coverage, contacting support).

🎯 Why Prompt Templates Matter in GenAI Apps

- **Clearer tone and persona**
(e.g., empathetic customer support agent)
- **More structured and actionable responses**
(step-by-step troubleshooting instead of vague replies)
- **Higher consistency and reliability**
(same quality across different queries)
- **Improved user experience**
(responses feel human, professional, and helpful)
- **Easier to maintain and scale**
(templates can be reused across tasks and domains)

https://github.com/Decoding-Data-Science/Omantel/blob/main/workshop_1_prompt_genai.ipynb

Prompt Engineering is

👍🏼 Good for:

- Testing and learning early
- When paired with evaluation it provides your baseline and sets up further optimization

👎🏼 Not Good for:

- Introducing new information
- Reliably replicating a complex style of method,i.e. Learning a new programming language
- Minimising token usage

Limitations of LLMs

- Knowledge cutoff
- No citations
- Incorrect data
- Limited context window

Workshop 2

- Llamaindex-rag

Connecting Dots of RAG

A solution to limited context windows

- **Retrieve** most relevant data
- **Augment** query with context
- **Generate** response

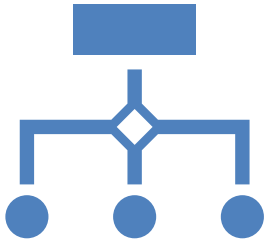
Introduction to LlamaIndex and Retrieval Augmented Generation (RAG)

Enhancing Large Language Models with External Knowledge

Presented by: Mohammad Arshad

What is LlamaIndex?

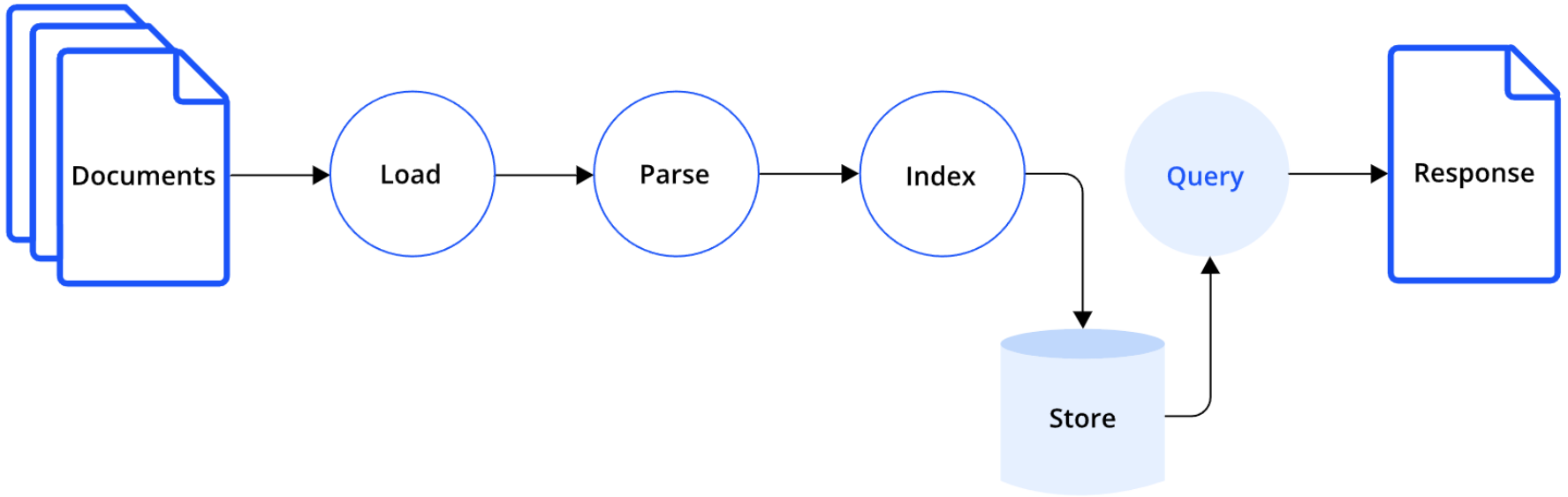
LlamaIndex is an open source project that allows users to build and query indexes over external data sources.



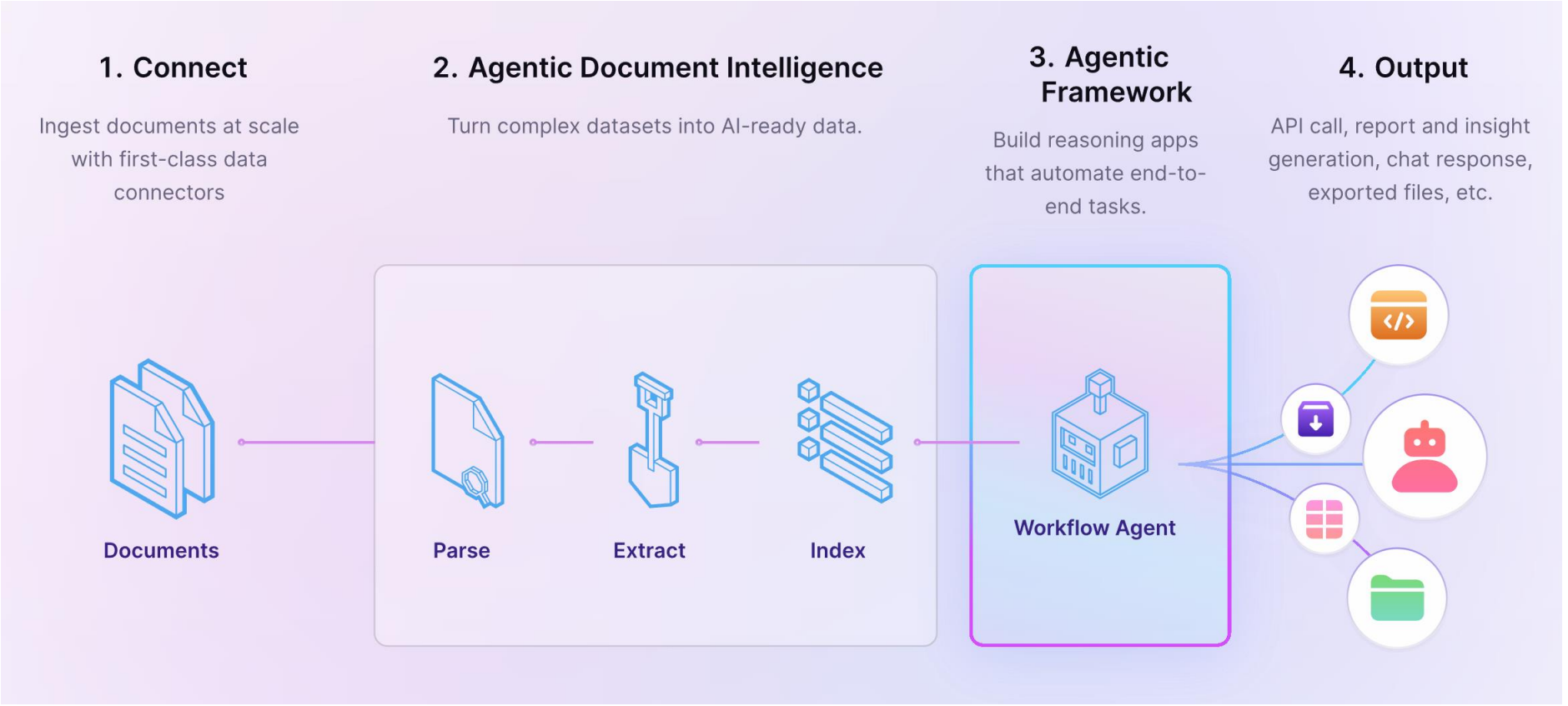
It enhances the capabilities of Large Language Models (LLMs) by providing them with access to structured data.

Before AI Agent Hype

Designed to work with Retrieval Augmented Generation (RAG), a technique that improves LLMs by retrieving relevant information from external data.



After AI Agent Hype



What is Retrieval Augmented Generation (RAG)?

RAG is a framework that combines a retriever with a generator (e.g., a large language model).

The retriever selects relevant documents from a corpus or data source.

The generator (LLM) then uses this information to produce more accurate and context aware responses.

This approach helps overcome limitations of LLMs by enhancing their knowledge with external, up to date information.

Why Use LlamaIndex with RAG?

LlamaIndex helps bridge the gap between large language models and external databases.

LLMs can be limited to the information they were trained on (e.g., knowledge cutoff dates).

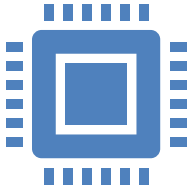
RAG enables LLMs to retrieve relevant, updated, and specific information from large external data sources.

Improves the accuracy and contextual relevance of generated responses by enhancing access to real time data.

Key Components of LlamaIndex

1**. Indexes:** Build indexes on external data (e.g., documents, databases).

2. **Query Engine**: Allows querying the indexed data using natural language queries.



3. **Retrieval Interface**: The interface between the LLM and the external data source to retrieve relevant information during query generation.

How LlamaIndex Works

1. **Data Ingestion :** LlamaIndex connects to external data sources (e.g., text files, databases) and builds an index.

2. **Query Processing :** When a question or query is posed to the LLM, LlamaIndex retrieves relevant information from the index.

3. **Response Generation :** The retrieved data is used by the LLM to generate a more informed and contextually accurate response.

Where to use it (Telco)

NOC / Network Operations

What it does: Find answers from past incidents and runbooks to fix outages faster.
Example: “Show similar fiber cuts in Muscat last year and steps taken.”
Needs: Runbooks, ticket history, alarm logs.

Customer Care (Call Center & Retail)

What it does: Help agents answer plan, device, eSIM and roaming questions quickly.
Example: “What’s the roaming price for KSA for Postpaid X?”
Needs: FAQs, product catalog, policy pages.

B2B / 5G & IoT

What it does: Reuse past proposals and SLAs to reply to RFPs and design solutions.
Example: “Draft a summary for a 5G private network for a hospital client.”
Needs: Old proposals, SLAs, reference designs.

Compliance & Fraud

What it does: Bring up the right policy or case pattern with citations.
Example: “What are KYC rules for prepaid activation? Cite the source.”
Needs: Regulatory docs, internal policies, case notes.

How it works (in 3 steps)

1) Add your documents – Put PDFs, Confluence pages, Jira tickets, FAQs into the index. We also handle Arabic and English.

2) Ask a question – The system finds the most relevant chunks of text and shows the sources.

3) Get a grounded answer – The LLM writes a short, safe answer with links back to the docs.

You can copy it to Jira/CRM.

Hands On: Setting Up LlamaIndex with RAG

1. Install LlamaIndex : pip install llama index

2. Connect to a Data Source : Use LlamaIndex to ingest and index data from text files, databases, or APIs.

3. Query the Index : Use the LlamaIndex API to perform natural language queries on the indexed data.

4. Integrate with LLM : Combine the LLM with LlamaIndex to perform Retrieval Augmented Generation.

LlamaIndex Query Example

```
from llama_index.core import VectorStoreIndex, SimpleDirectoryReader
documents = SimpleDirectoryReader("data").load_data()
index = VectorStoreIndex.from_documents(documents=documents)
query_engine = index.as_query_engine()
response = query_engine.query("who is mentioned about in document")
print(response)
```

Benefits of Using LlamaIndex with RAG

Enhanced Knowledge : Access to a broader knowledge base beyond the LLM's training data.

Real Time Updates : Ability to fetch and retrieve the most up to date information.

Improved Accuracy : More contextually relevant and accurate responses to user queries.

Scalability : Works with large, external datasets or documents without overloading the LLM.

Getting started (developers)

Minimal stack: LlamaIndex + embeddings (OpenAI/Azure) + BM25 + Rerank. Store vectors in a simple DB (e.g., Pine Cone).

Quick win pilot: Pick ONE domain: e.g., customer FAQs. Load 200–500 docs. Build a Q&A chat with citations.

Good defaults: Chunk by headings, add metadata (domain, language), top_k=5, max_output_tokens ~ 300.

Safety: Always show sources. Hide sensitive data. Log feedback from agents to improve.

Conclusion

LlamaIndex enhances the capabilities of LLMs by integrating with external data sources.

Retrieval Augmented Generation (RAG) enables LLMs to produce more informed and accurate responses.

Together, LlamaIndex and RAG offer powerful solutions for data intensive and real time applications.

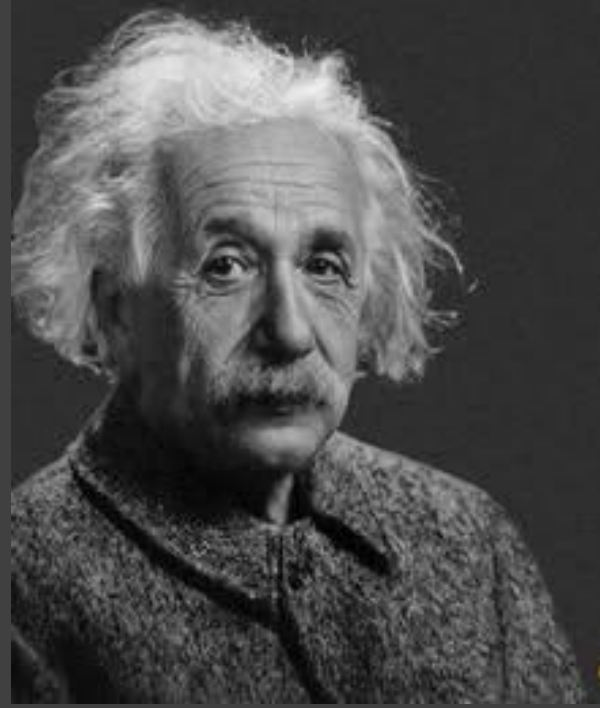
Questions?

Any questions on LlamaIndex or Retrieval Augmented Generation?

Further resources available on GitHub and official LlamaIndex documentation.

The more I learn,
the more I realize
how much I don't know.

-Albert Einstein

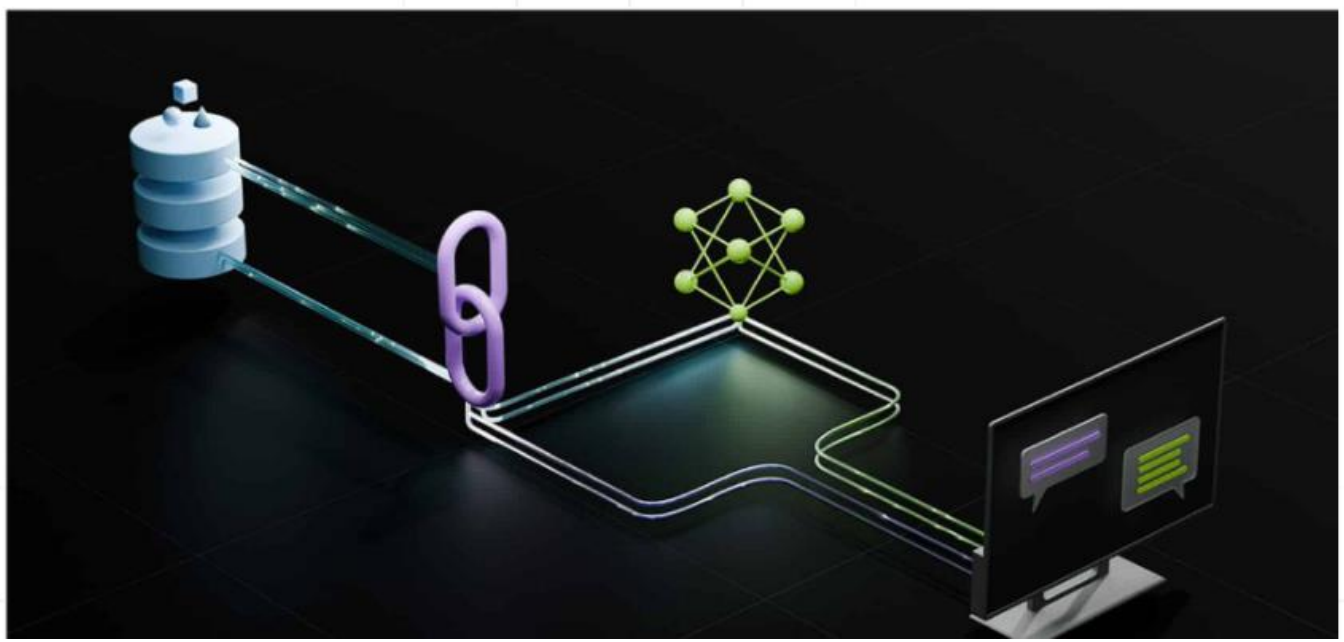


Day 2: AI for Programmer - Evaluation & Attendance

Advancing to Retrieval-Augmented Generation (RAG)

If more context is needed, RAG becomes essential.

This technique involves giving the model access to a domain-specific knowledge base, which it can query to enhance its responses. RAG is particularly useful for introducing new, accurate information and reducing model hallucinations.



RAG is

👍🏼 Good for:

- Introducing new information to the model to update it's knowledge
- Reducing hallucinations by controlling content (grounding)

👎🏼 Not Good for:

- Embedding understanding of a broad domain
- Teaching the model to learn a new language, format or style
- Reducing token usage

Stages of RAG

1. Load
2. Index/embed
3. Query
4. Generate
5. Evaluate

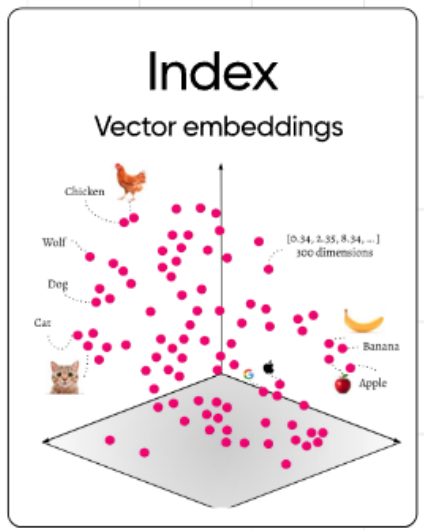
Your Data

Database
Structured

Documents
Unstructured

API
Programmatic

Load



Query 💬

User

response

Prompt + query
+ relevant data

LLM 🤖
Model like GPT, llama

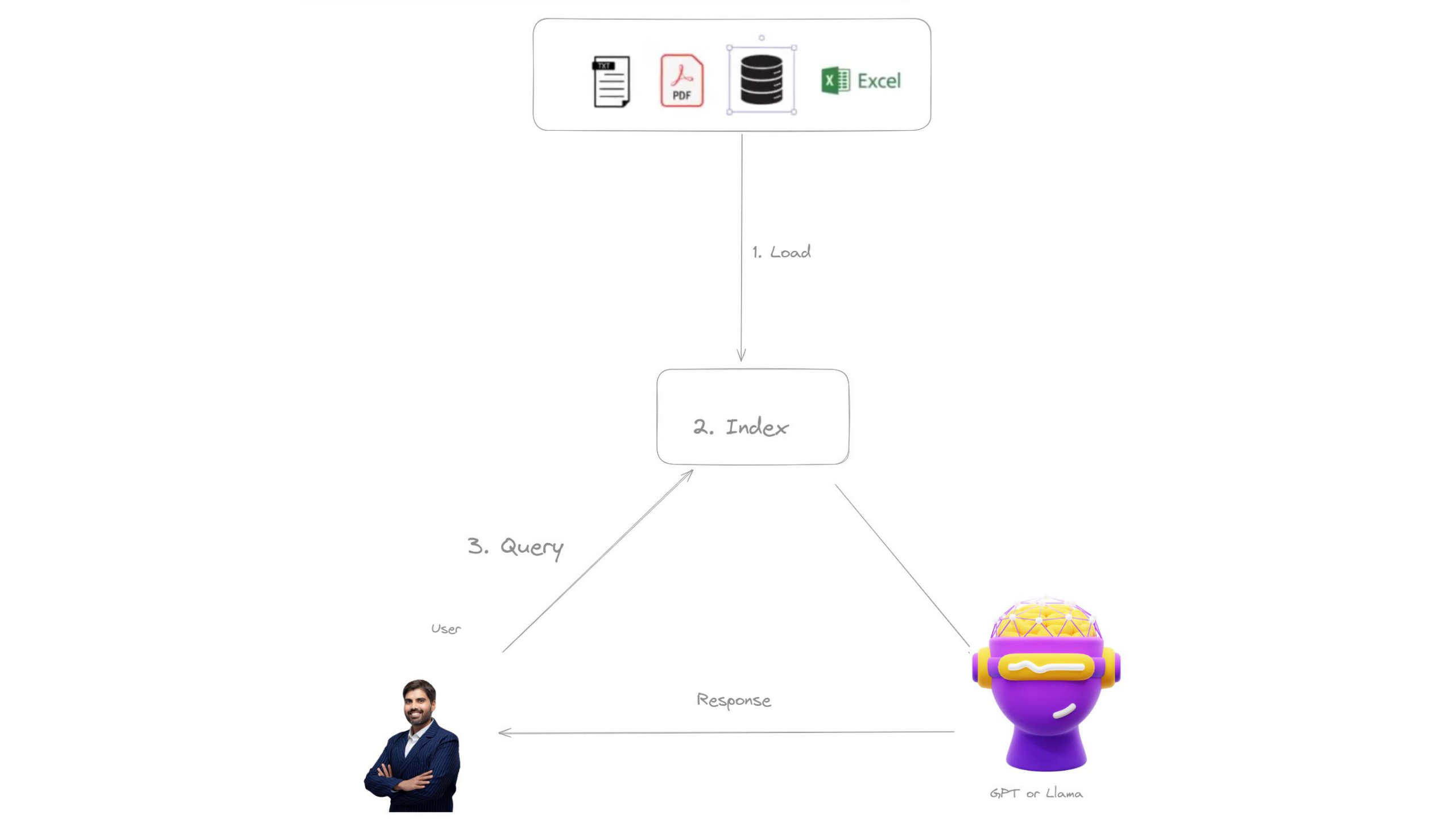
We can solve some of the limitations by giving the relevant context

Elements of a Prompt

A prompt is composed with the following components:

- Instruction
- Context
- Input Data
- Output Indicator

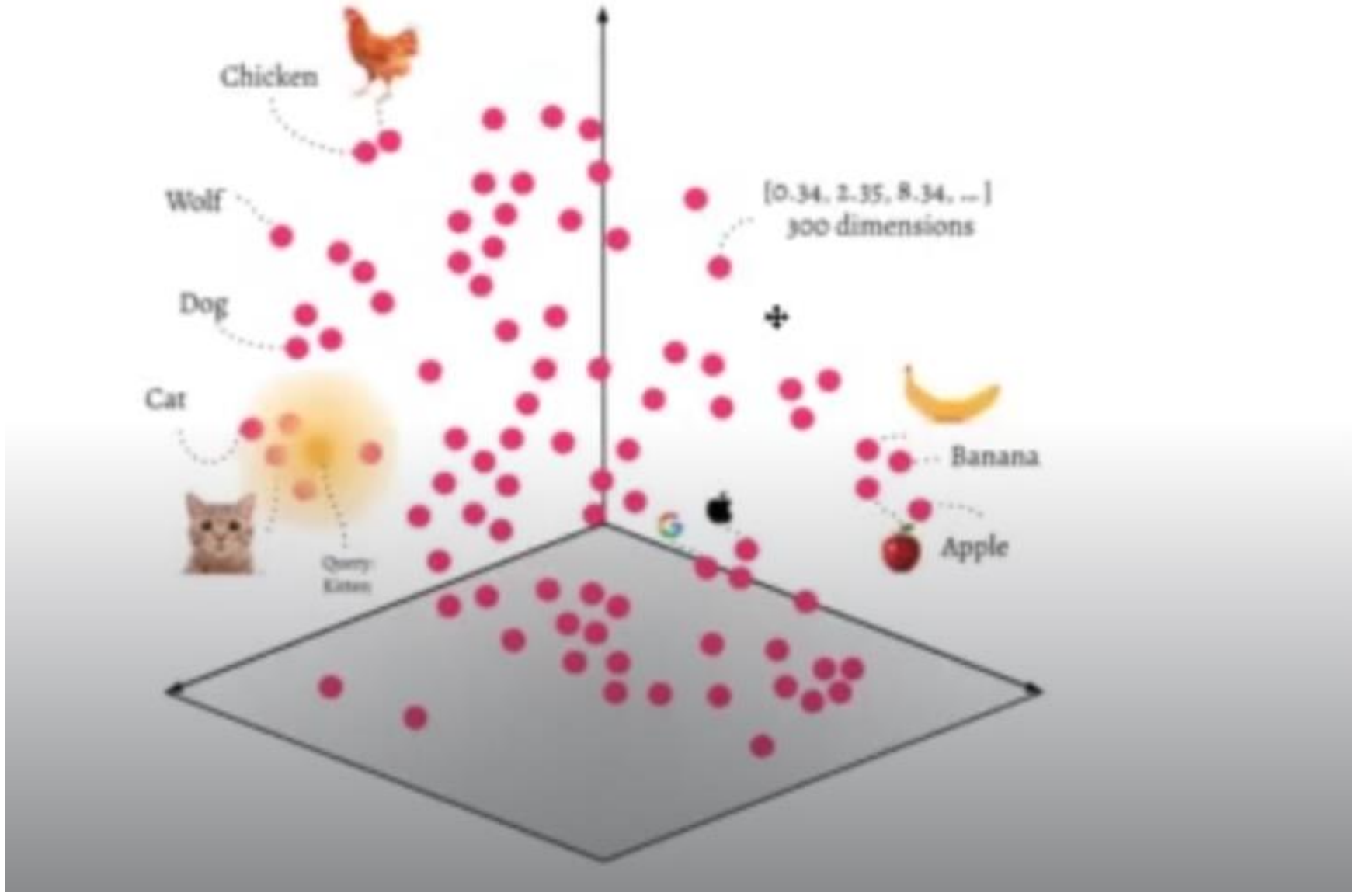
```
Classify the text into neutral,
negative or positive

Text: I think the food was okay.

Sentiment:
```


TXT
PDF
Excel
1. Load
2. Index
3. Query
User
Response
GPT or Llama

Take Aways

- You all will get certificate from Cognit-dx
- Check Cognit-dx.com fir future Courses
- Evaluation & Attendance
- Group Photo
- Keep Learning & being curious

2. Index (vector embeddings)



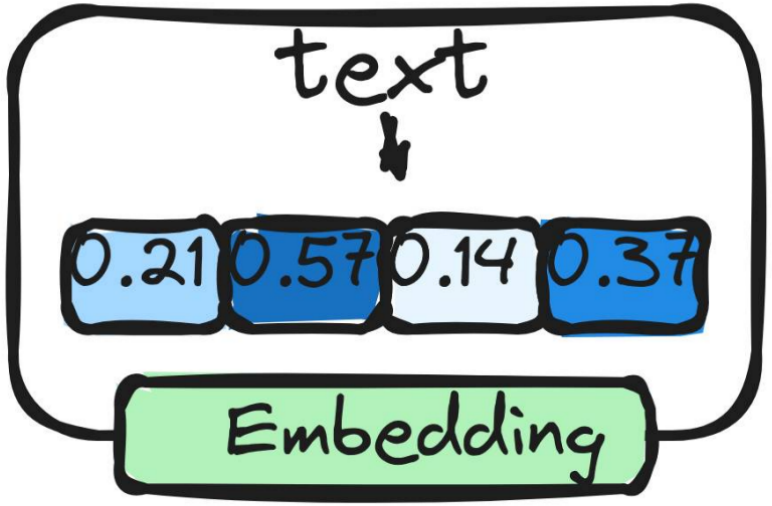
Embeddings

Embeddings are compact numerical representations of words or entities that help computers understand and process language more effectively. These representations encode the meaning and context of words, allowing machines to work with language in a more meaningful and efficient way.

- OpenAIEmbeddings
- HuggingFaceEmbeddings
- GPT4AllEmbeddings
- etc

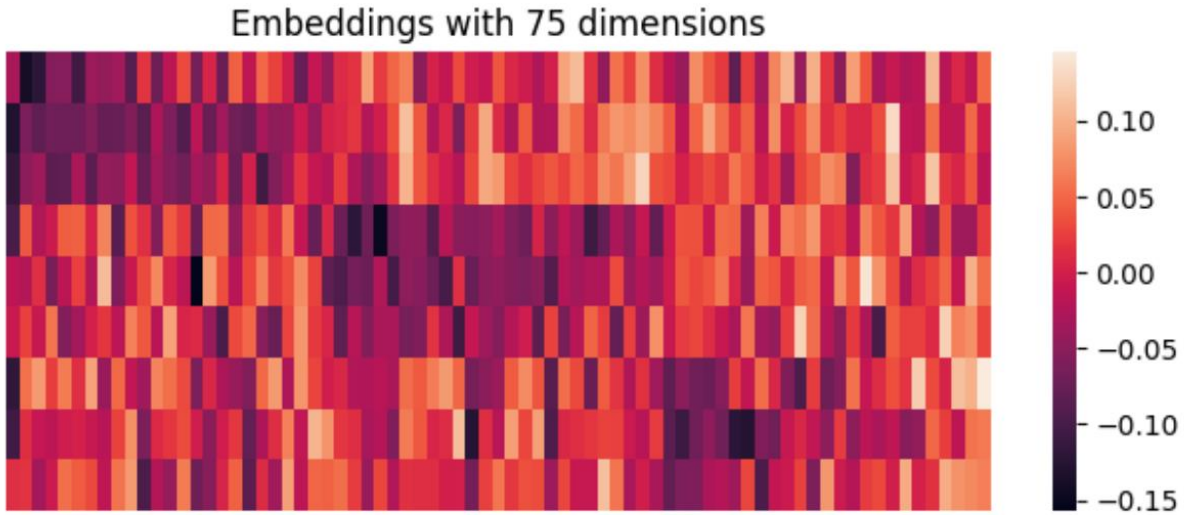
- SpacyEmbeddings

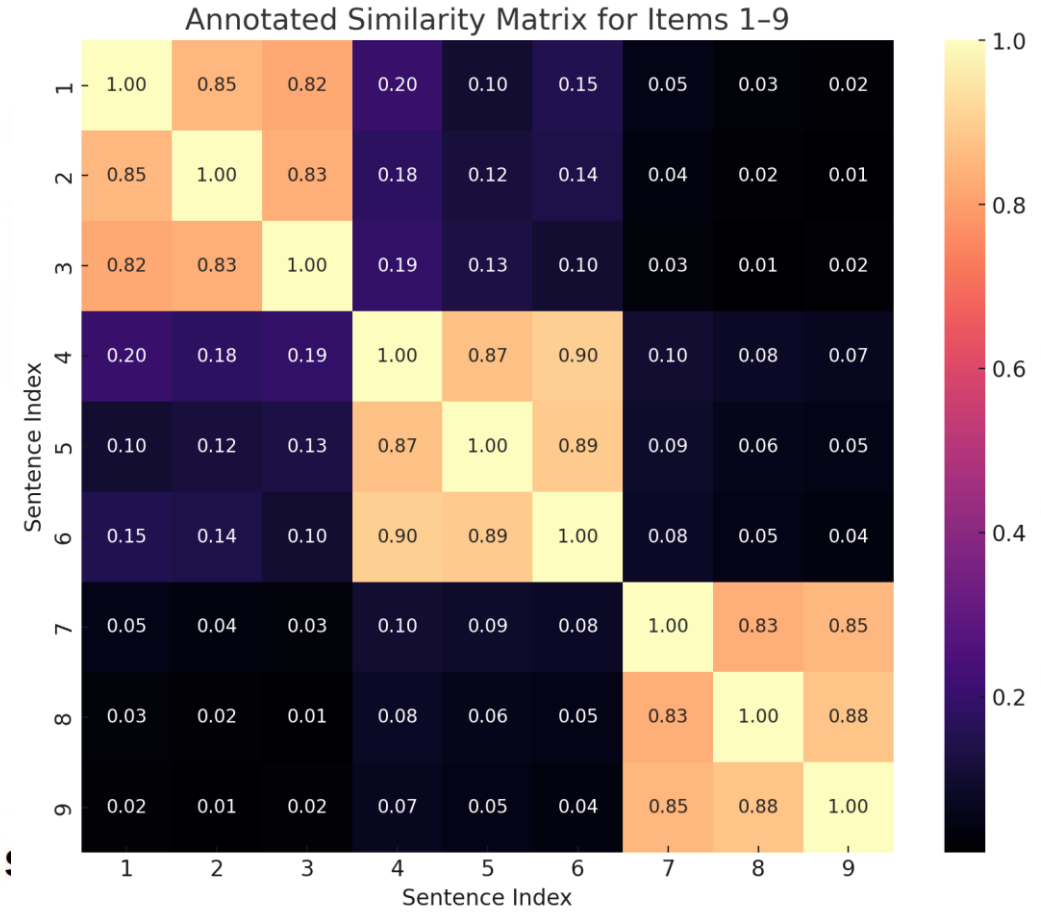
- FakeEmbeddings



Embeddings example

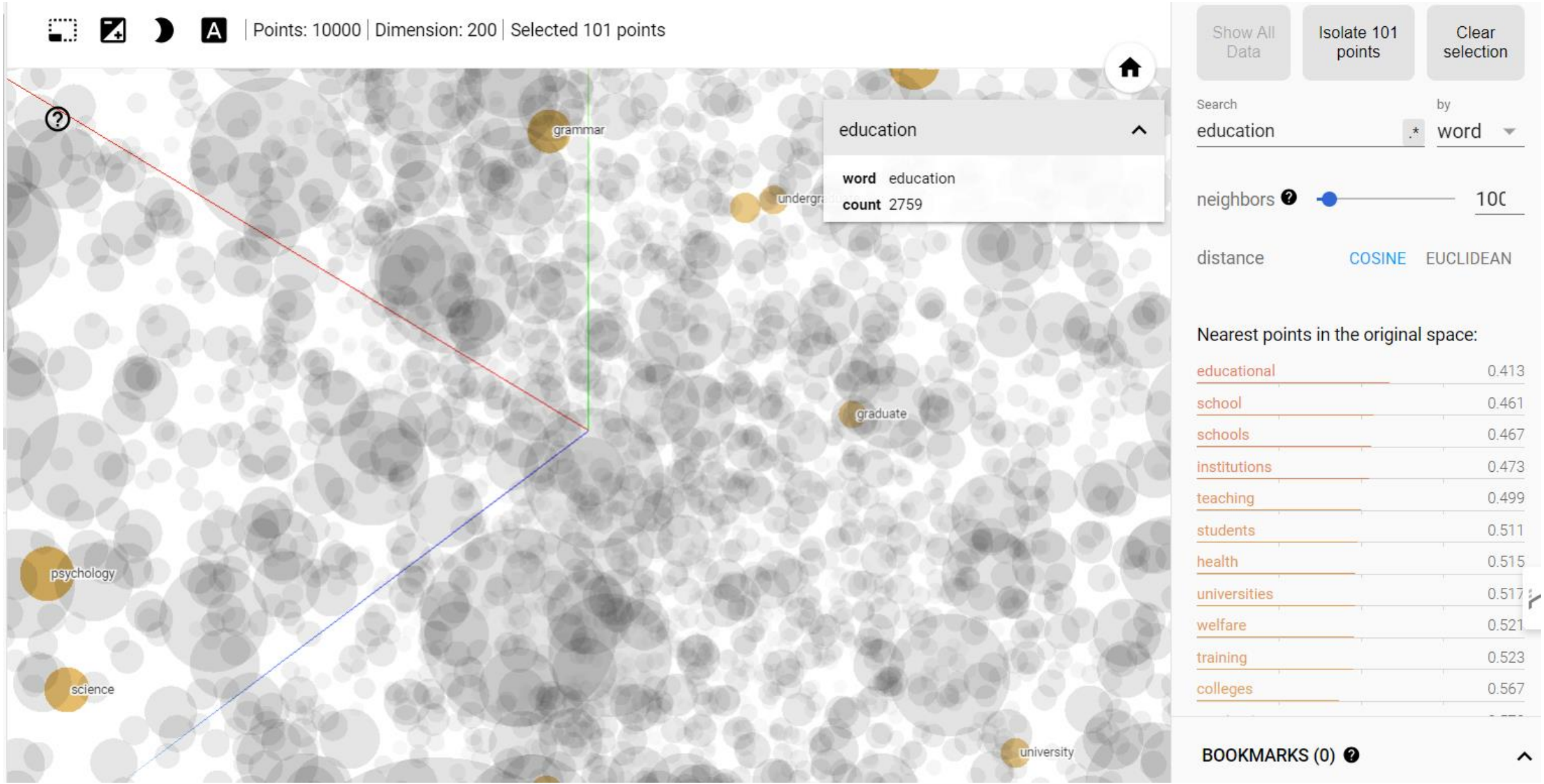
1)Best travel neck pillow for long flights
2)Lightweight backpack for hiking and travel
3)Waterproof duffel bag for outdoor adventures
4)Stainless steel cookware set for induction cooktops
5)High-quality chef's knife set
6)High-performance stand mixer for baking
7)New releases in fiction literature
8)Inspirational biographies and memoirs
9)Top self-help books for personal growth





Demo

https://projector.tensorflow.org/



Llama index exercise

Where to use it (simple examples)

NOC / Network Operations

What it does: Find answers from past incidents and runbooks to fix outages faster.
Example: “Show similar fiber cuts in Muscat last year and steps taken.”
Needs: Runbooks, ticket history, alarm logs.

Customer Care (Call Center & Retail)

What it does: Help agents answer plan, device, eSIM and roaming questions quickly.
Example: “What’s the roaming price for KSA for Postpaid X?”
Needs: FAQs, product catalog, policy pages.

B2B / 5G & IoT

What it does: Reuse past proposals and SLAs to reply to RFPs and design solutions.
Example: “Draft a summary for a 5G private network for a hospital client.”
Needs: Old proposals, SLAs, reference designs.

Compliance & Fraud

What it does: Bring up the right policy or case pattern with citations.
Example: “What are KYC rules for prepaid activation? Cite the source.”
Needs: Regulatory docs, internal policies, case notes.

Demo RAG in 5 lines of Code

- # Load data
- documents = SimpleDirectoryReader("data").load_data()

- # Create an Index
- # Chunk data and convert it into vector embeddings

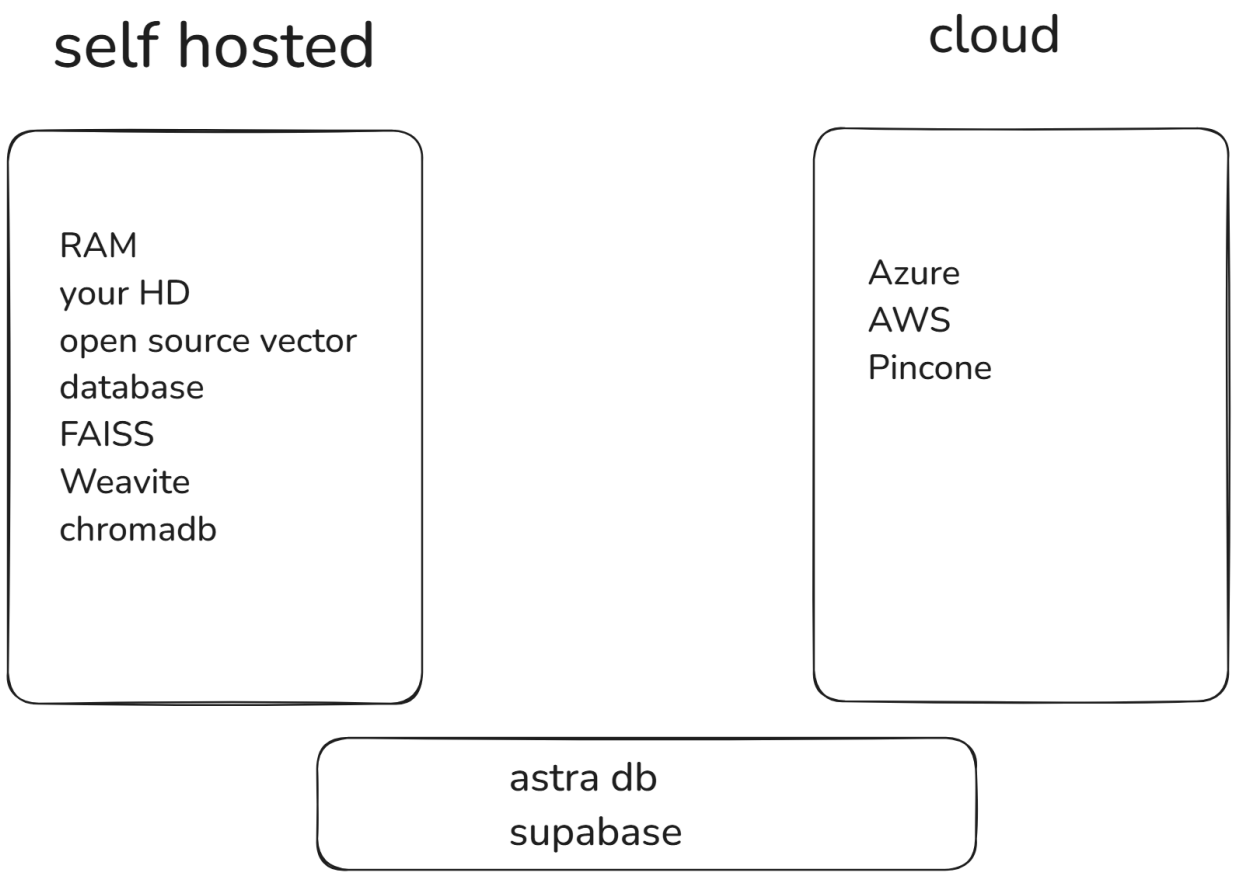
- index = VectorStoreIndex.from_documents (documents)

- # Create a QueryEngine for Retrieval & Augmentation
- query_engine = index.as_query_engine()

- # Generate response by asking a query to the QueryEngine
- response = query_engine.query("What did the author do growing up?")

- print(response)

Vector Stores



Pinecone is a cloud-native vector database offering a seamless API and hassle-free infrastructure. It eliminates the need for users to manage infrastructure, allowing them to focus on **developing and expanding their AI solutions**. Pinecone excels in quick data processing, supporting metadata filters, and sparse-dense index for accurate results.

Key Features

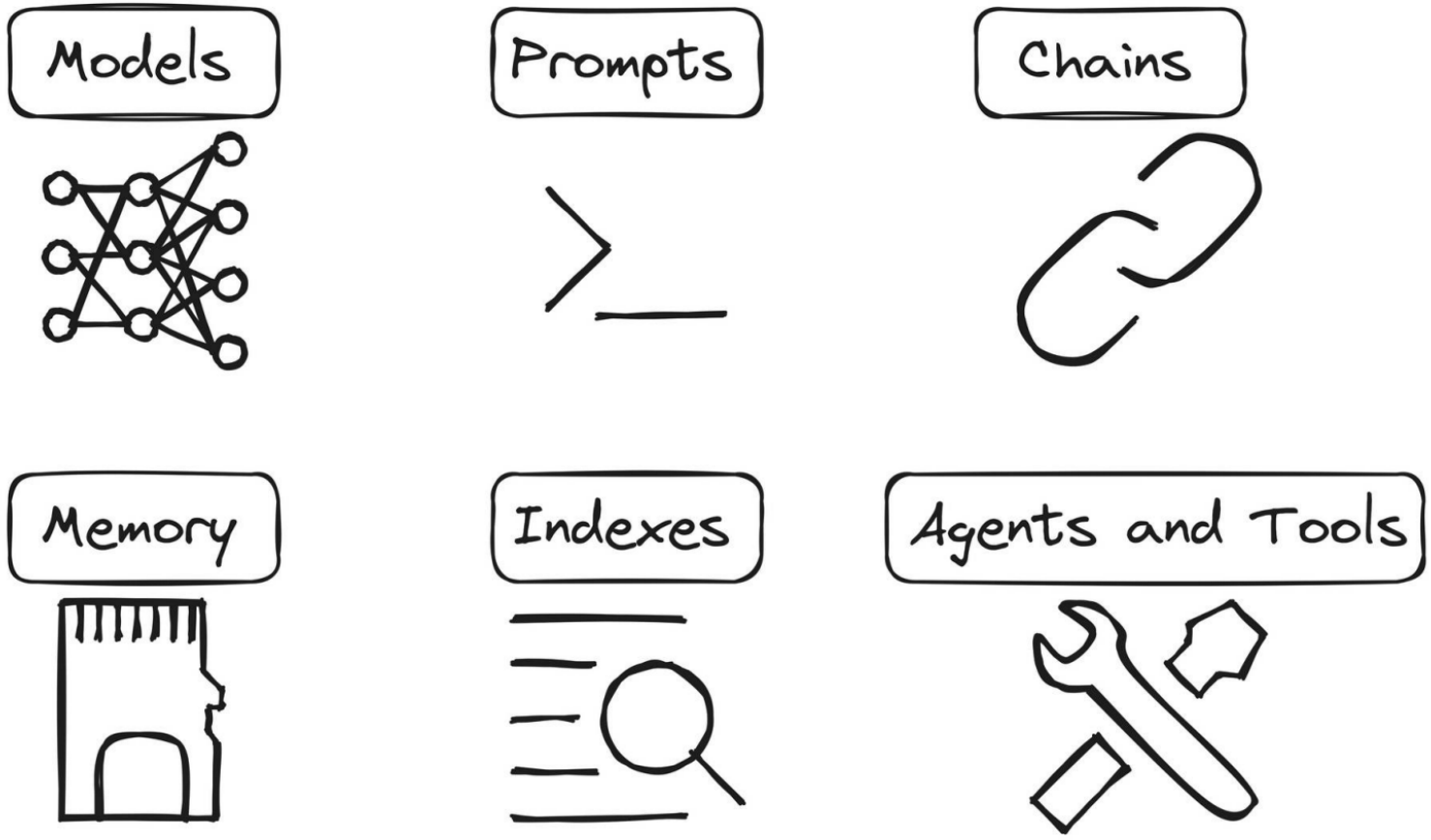
- ✓Duplicate detection
- ✓Rank tracking
- ✓Data search
- ✓Classification
- ✓Deduplication

Demo Class

https://colab.research.google.com/drive/1yqU6cyPbhZxGtrp1zeBzxU75WNIlH-d1?usp=sharing

What is LangChain? Quick glance

LangChain is a framework that makes it easier to create applications using large language models (LLMs).

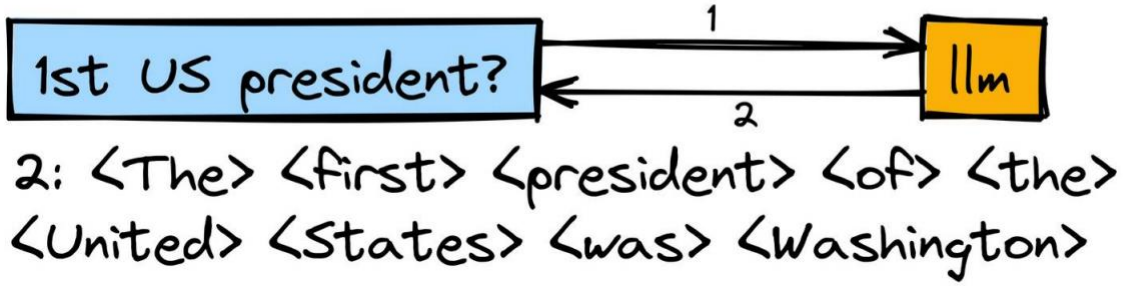


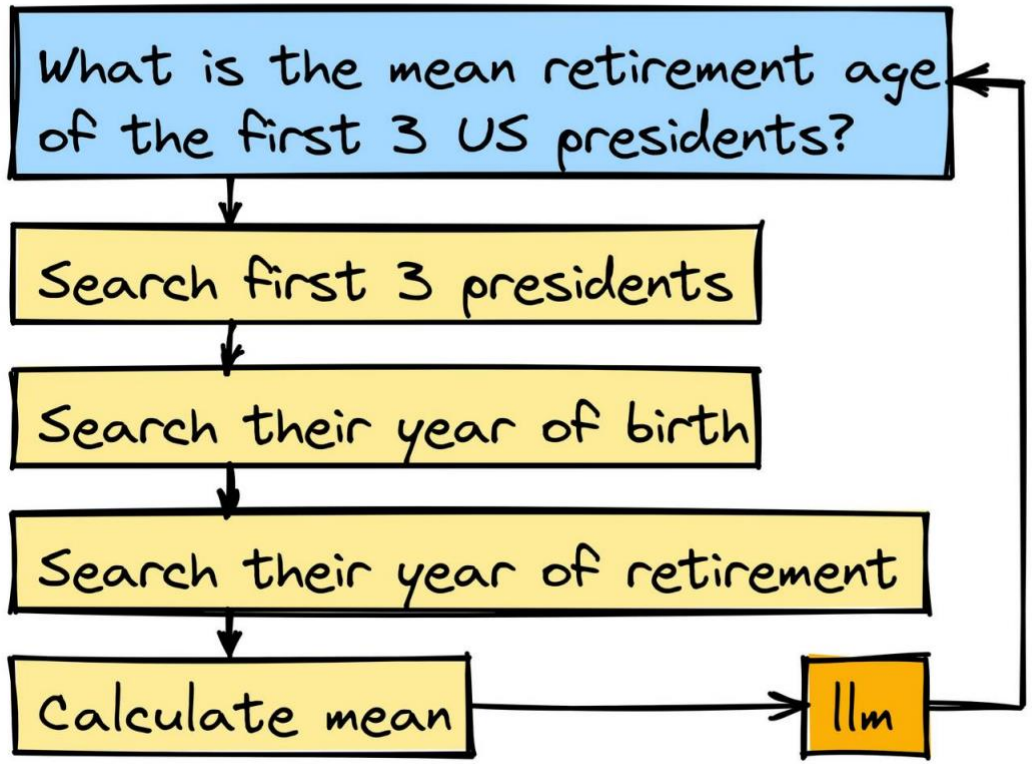
LLM LangChain Application

LLMs are used in LangChain applications in several purposes:

- language model to answer user response

- “llm butter” for LangChain components

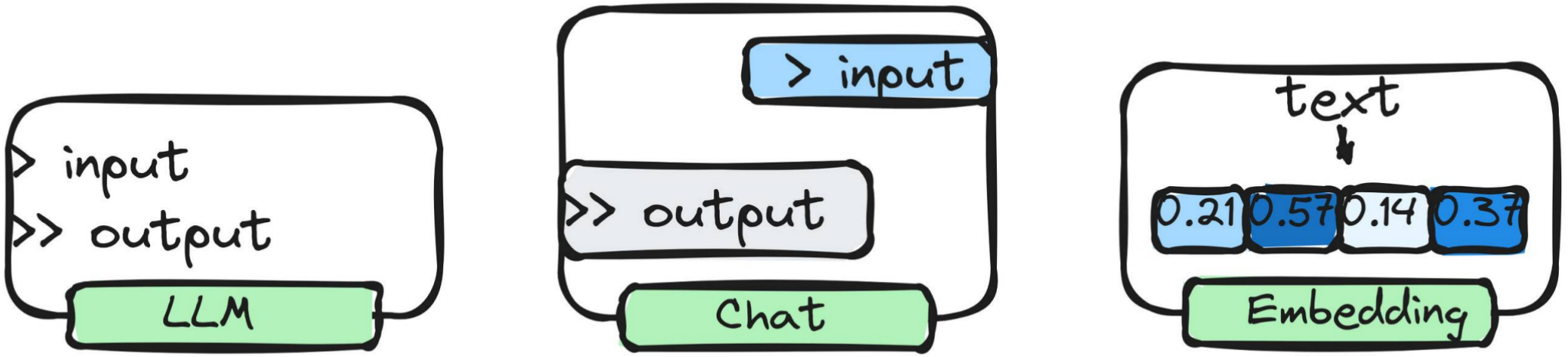




Summary: Models

LangChain supports a variety of different models, so you can choose the one that suits your needs best:

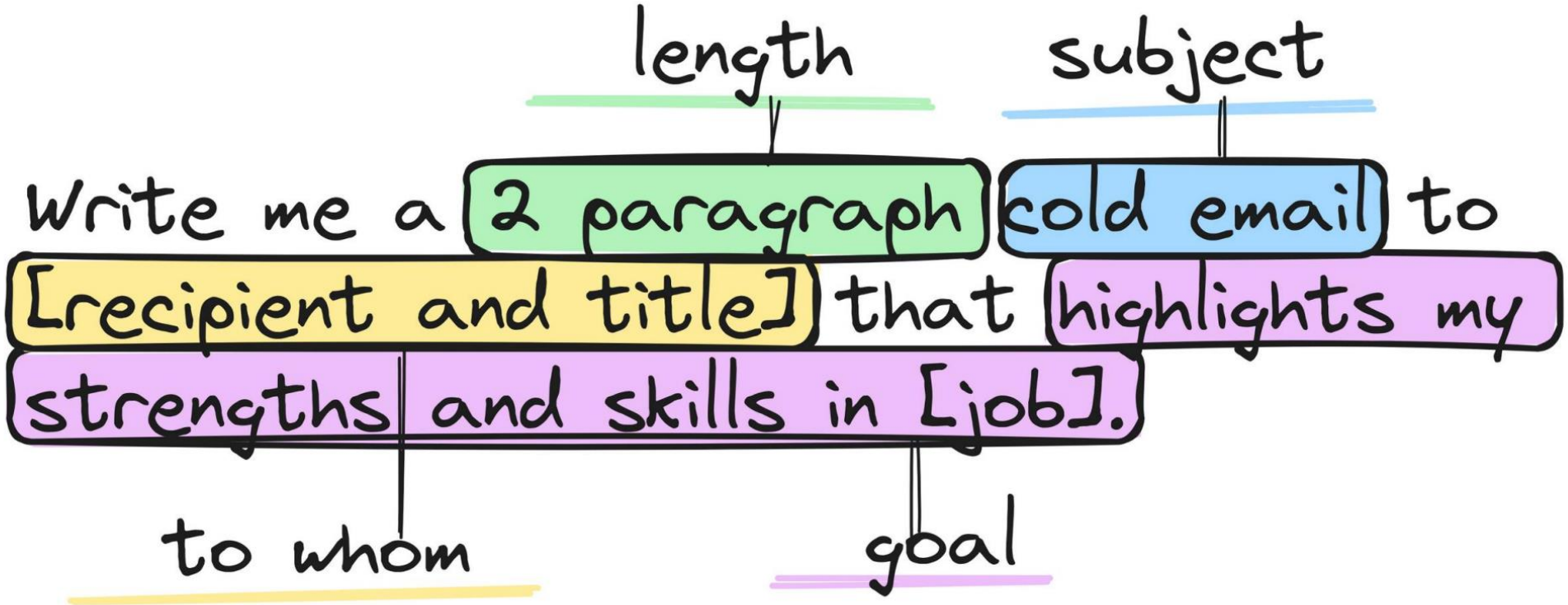
- **Language Models** are used **for generating text**
    - LLMs utilize APIs that take input text and generate text outputs
    - ChatModels employ models that process **chat messages** and produce responses
- **Text Embedding Models** convert text into **numerical representations**



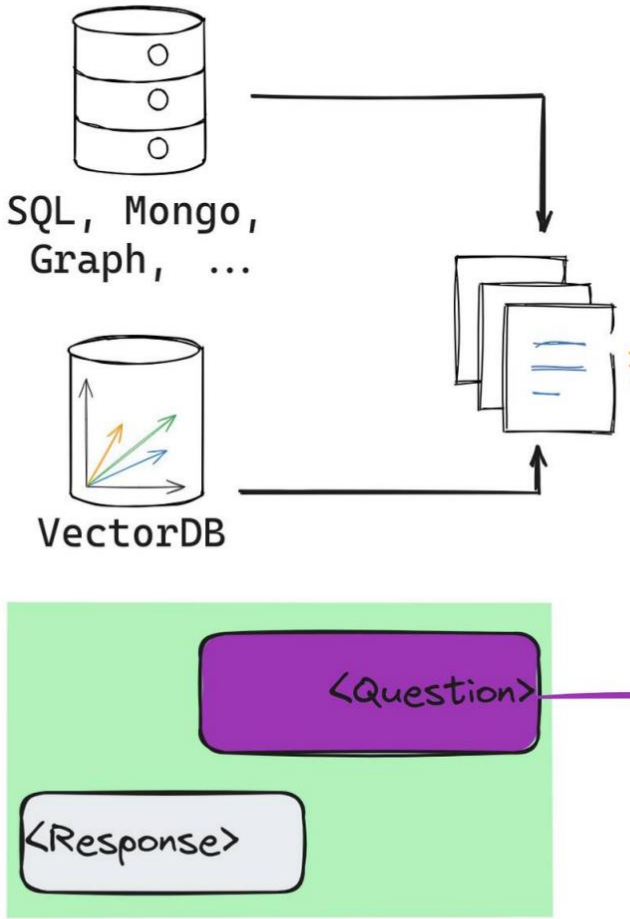
Prompts

Prompts are the instructions that you give to the LLM. They tell the LLM what you want it to do, and how you want it to do it.

Prompts can take the form of a string (for Language Models) or a list of messages (for Chat Models).



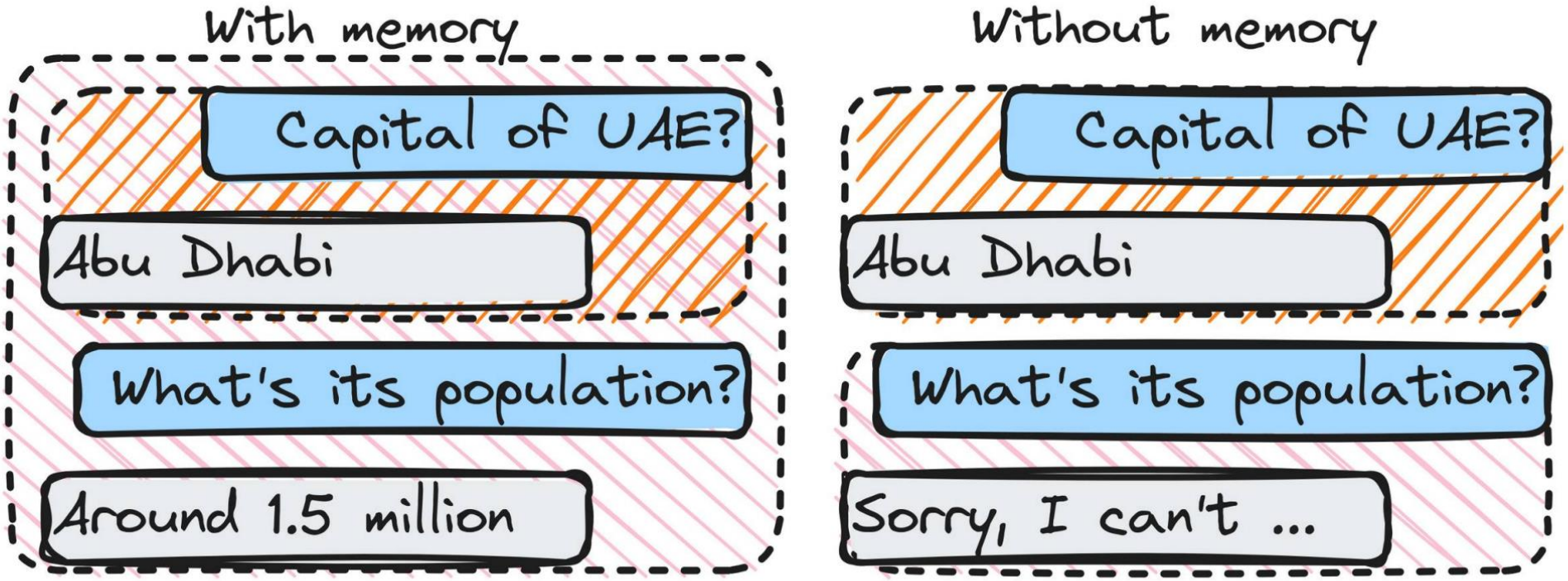
Prompts




```python
def setup_rag(**kwargs):
    retriever = chroma_db.as_retriever(**kwargs)
    template = """ You are an expert in  ...
    Answer the question based only on the following context:
    {context}

    Question: {question}
    """
    prompt = ChatPromptTemplate.from_template(template)

    output_parser = StrOutputParser()
    setup_and_retrieval = RunnableParallel(
        {"context": retriever,"question": RunnablePassthrough()}
    )
    rag_chain = setup_and_retrieval | prompt | llm | output_parser
    return rag_chain
```

Memory

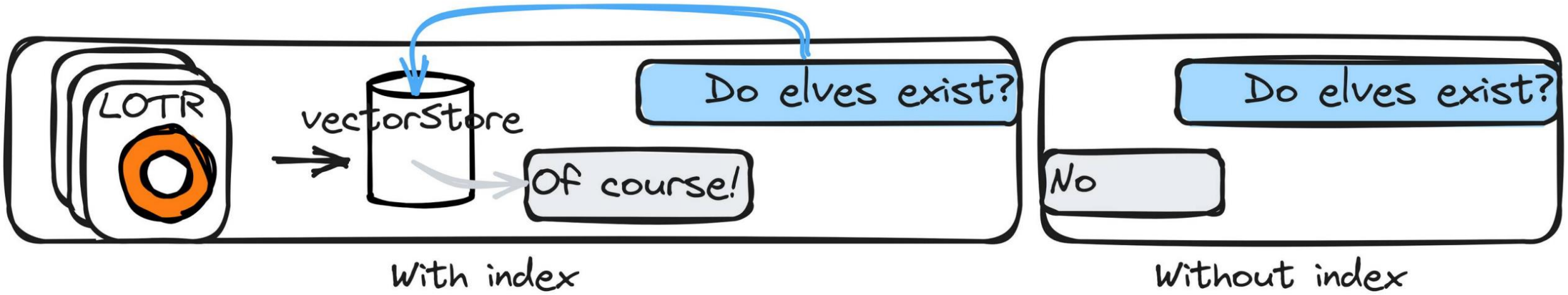
LangChain introduces memory components that enable the retention and utilization of past interactions. By default, LangChain components are stateless, treating each query independently. Memory in LangChain allows for the storage and management of previous chat messages.



Indexes

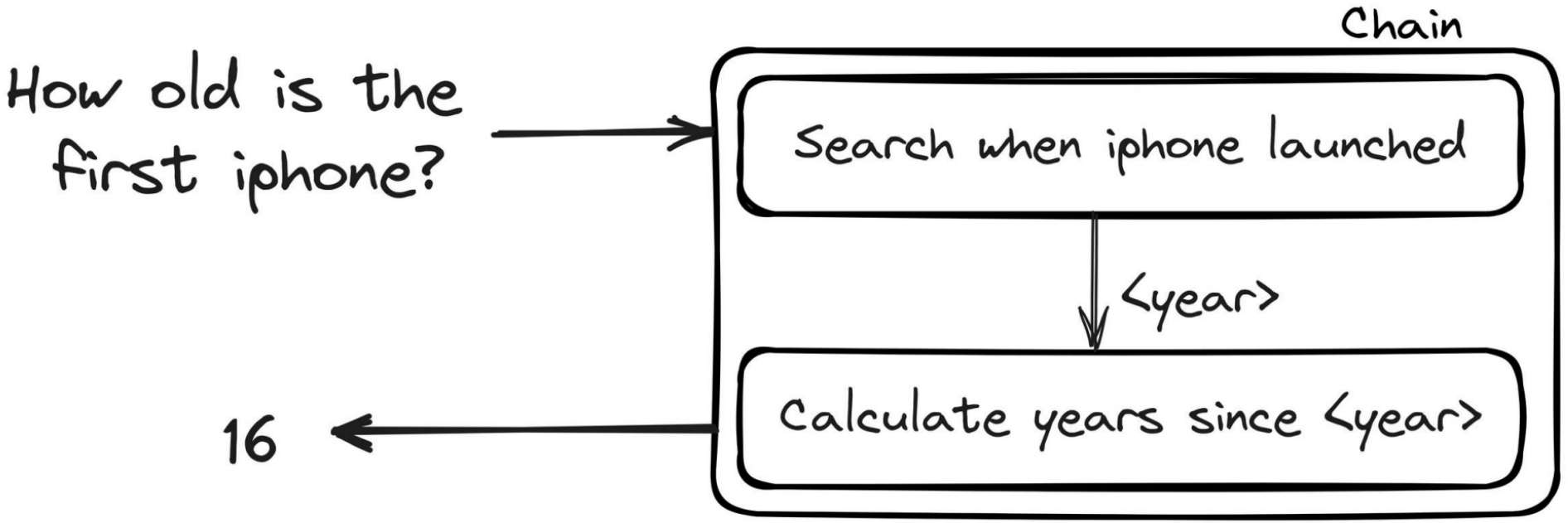
Indexes are databases of information that can be used to provide context to the LLM.

For example, if you are asking the LLM to answer a question about a particular topic, you could use an index to provide the LLM with a list of relevant articles or websites.



Chains

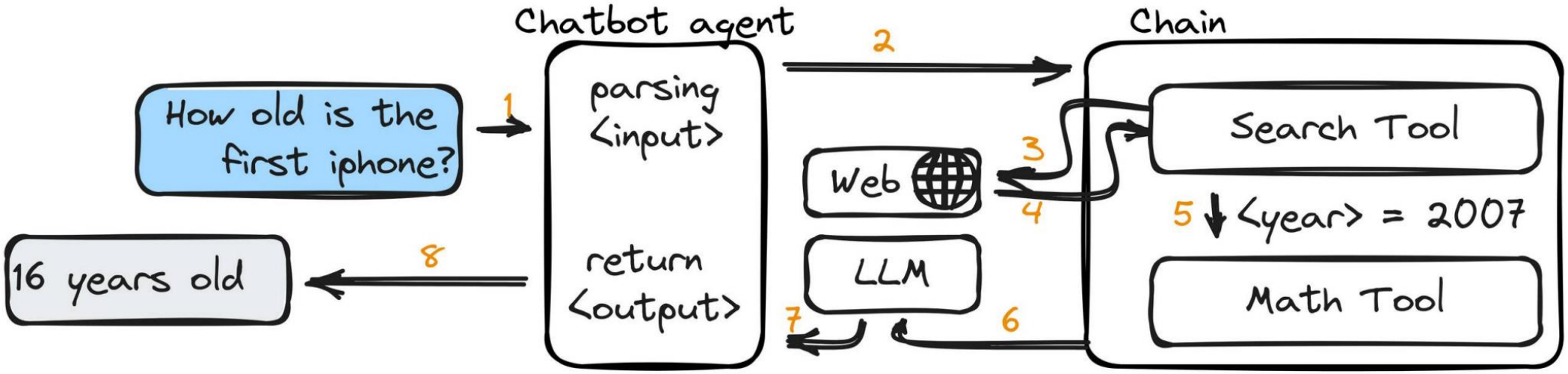
Chains are sequences of calls to the LLM. They allow you to perform complex tasks by chaining together multiple calls to the LLM. There are multiple types of chains to solve different cases.



Agents and Tools

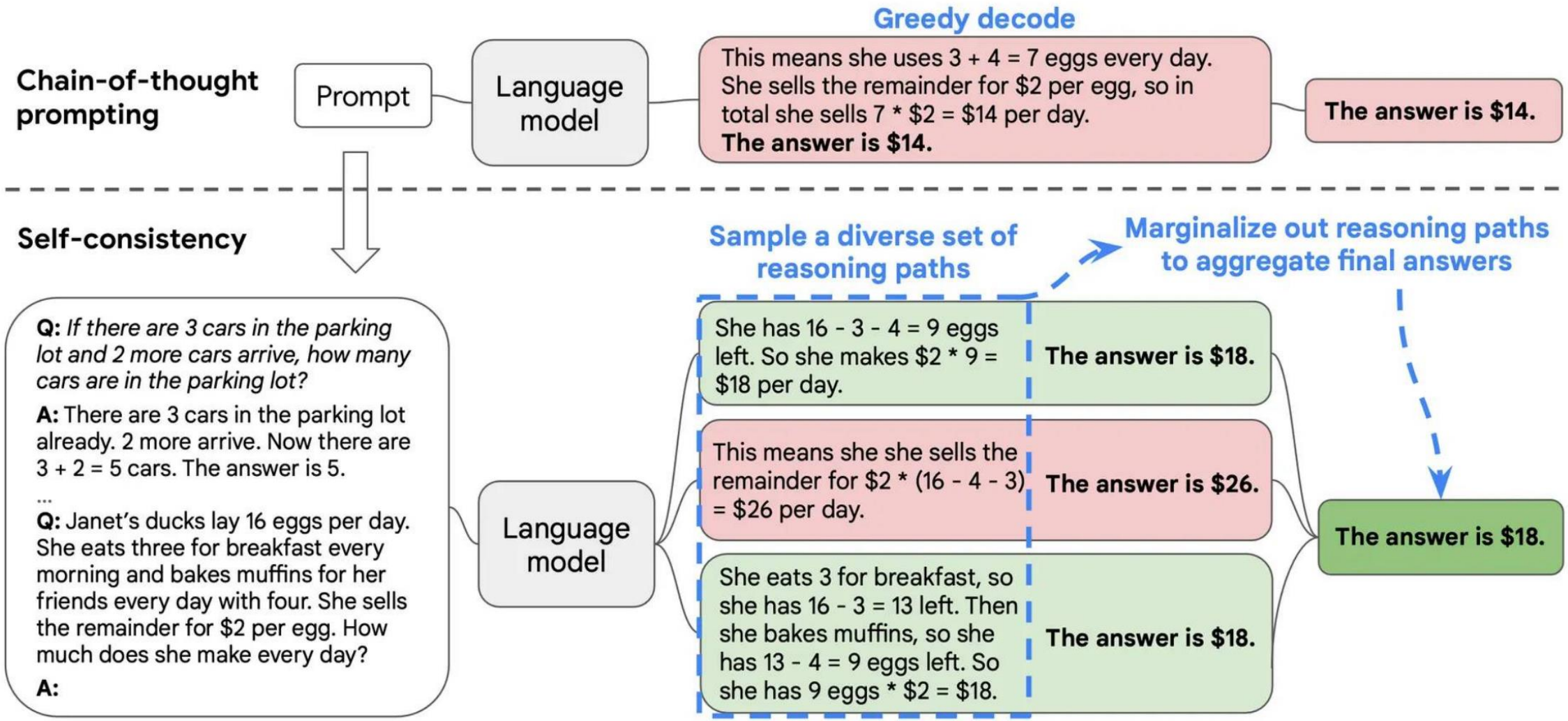
Agents are objects that manage interaction between the user and the LLM. They handle things like *parsing input*, *generating output* and determining the sequence of actions to follow and tools to use.

A tool is a function designed to perform a specific task. Examples of tools can range from *Google Search*, *DB lookups*, *API calls*, to other chains. The standard interface for a tool is a function that accepts a string as input and returns a string as output.



Tools: Data Loaders

Agents and Tools: Prompt Alternative



Appendix/Extra

Langchain + Agent Demo

Q & A

Lang Chain Memory

Crew AI