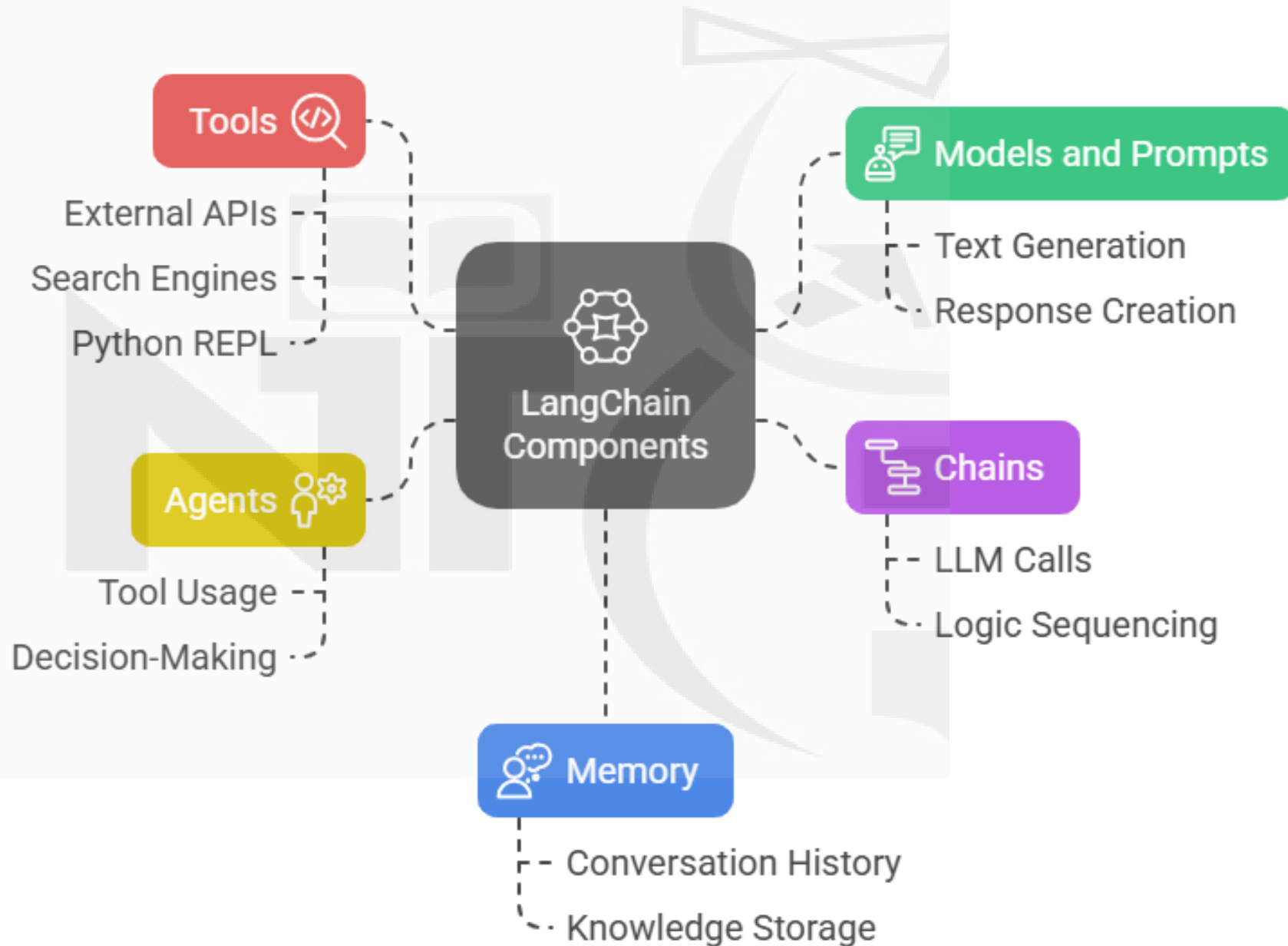


Models

LANGCHAIN

MUKESH KUMAR

LangChain Components and Their Functions



Available chat models in LangChain

<https://python.langchain.com/docs/integrations/chat/>

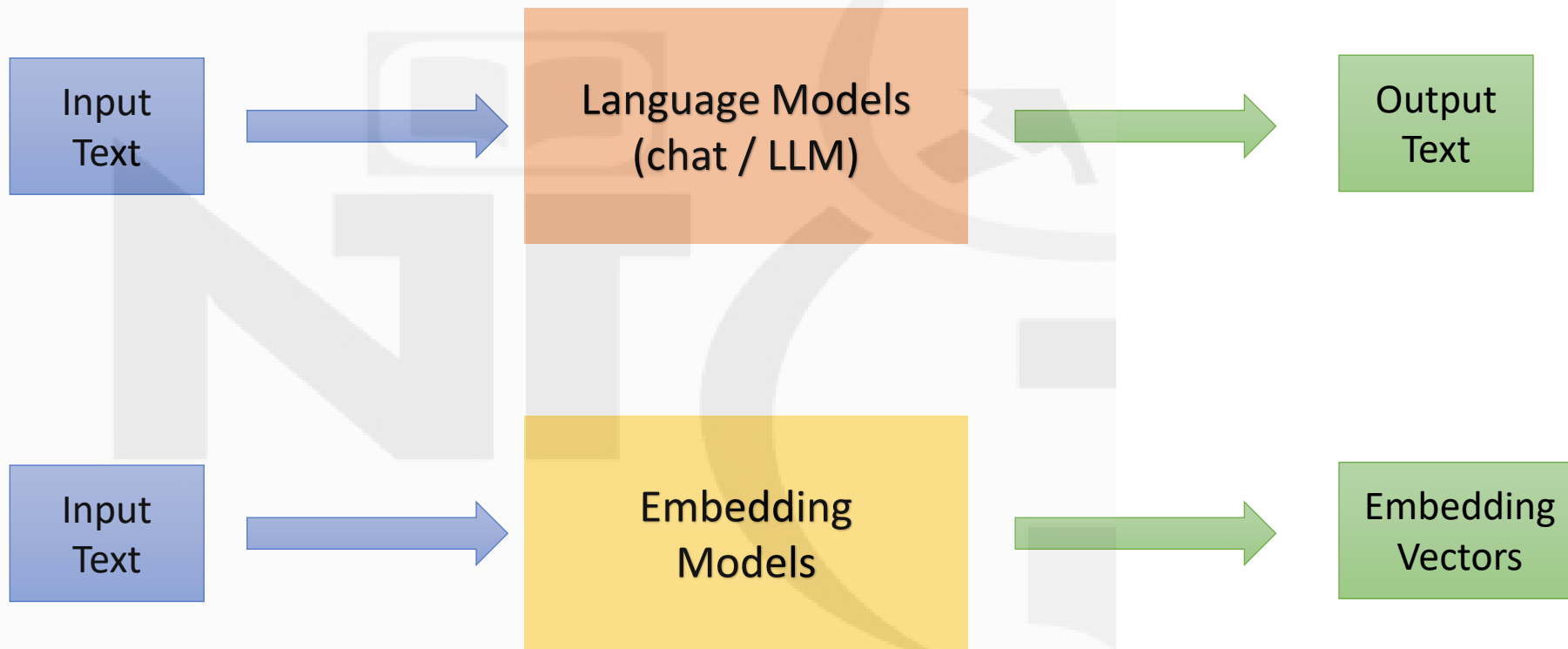
What is the Models Component?

- LangChain provides **standardized interfaces** to interact with various types of AI models.
- Model component in LangChain facilitates interaction with these AI Models

Common Model Types in LangChain

| Model Type | Use Case | Example Classes |
|------------------------|---------------------------------|--|
| ChatModel | Chat-style interaction | ChatOpenAI, ChatAnthropic, ChatGooglePalm |
| LLM | Single text completion | OpenAI, HuggingFaceHub, Cohere |
| TextEmbedding Model | Convert text into embeddings | OpenAIEmbeddings, HuggingFaceEmbeddings |

Language Vs Embedding Models

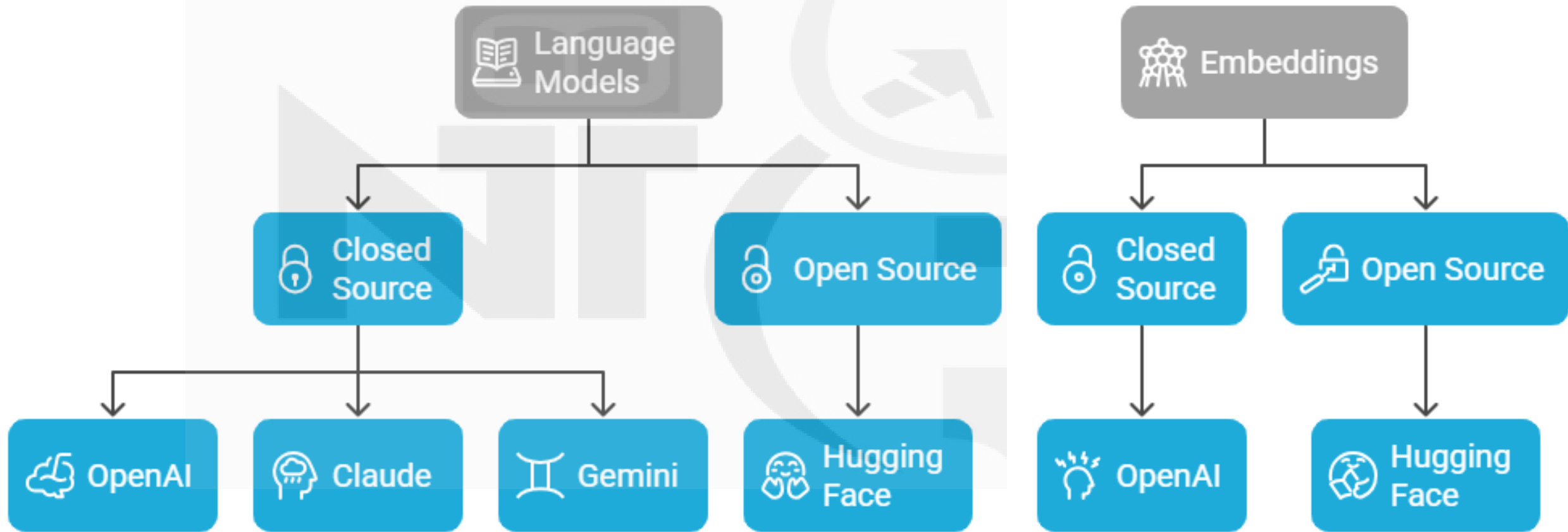


Language models are used for building chat bot like apps
Embedding models are used for simentic search which are used to build RAG based apps

Summary

- LangChain has a component call model component which helps us interact with AI models
- There are two types of models Embedding and language models

We will work on all these





Working with Language Models

```
graph TD; A[Large Language Models] --> B[LLMS]; A --> C[Chat Models]
```

Large Language
Models

LLMS

Chat Models

LLMs Vs Chat Models

- LLMs are general purpose models that take single string as input and generate single string as output
- Chat models are specialized model used in conversation model(used to build Agents , chatbots, coding assistant)
- Chat models

LLMs

LLMs (langchain.llms)

- Type: Text-in, Text-out
- Input: A single string prompt
- Output: A single string completion
- Examples: GPT-3, Mistral (text variant), Falcon, LLaMA-2 (non-chat), T5

Use When:

- You want a simple text completion
- You're working with non-chat APIs
- You don't need structured conversation or message roles

LLM Example

```
from langchain.llms import OpenAI  
  
llm = OpenAI(model_name="text-davinci-003")  
response = llm("What is LangChain?")
```



LLMs are being replaced by chat models in Langchain

What are Chat Models?

Chat Models (`langchain.chat_models`)

- Type: Message-in, Message-out
- Input: List of `HumanMessage`, `SystemMessage`, `AIMessage`, etc.
- Output: One `AIMessage` (structured)
- Examples: gpt-3.5-turbo, claude-3-haiku, mistral-instruct, llama3-chat

Use When:

- You're working with chat-style APIs
- You need system messages, multi-turn memory, or structured input
- You want to build agents, chains, or tools

Chat Model Example

```
from langchain.chat_models import ChatOpenAI
from langchain.schema import HumanMessage

chat = ChatOpenAI(model_name="gpt-3.5-turbo")
response = chat([HumanMessage(content="What is LangChain?")])
```


Example Flow:

- Provider: OpenAI
- Model: gpt-3.5-turbo
- LangChain Interface: ChatOpenAI

```
from langchain.chat_models import ChatOpenAI
from langchain.schema import HumanMessage

chat = ChatOpenAI(model_name="gpt-3.5-turbo")
response = chat([HumanMessage(content="What's LangChain?")])
```

Example 2

- **Provider:** Ollama
Model: mistral / mistral:7b-instruct
LangChain Interface: ChatOllama

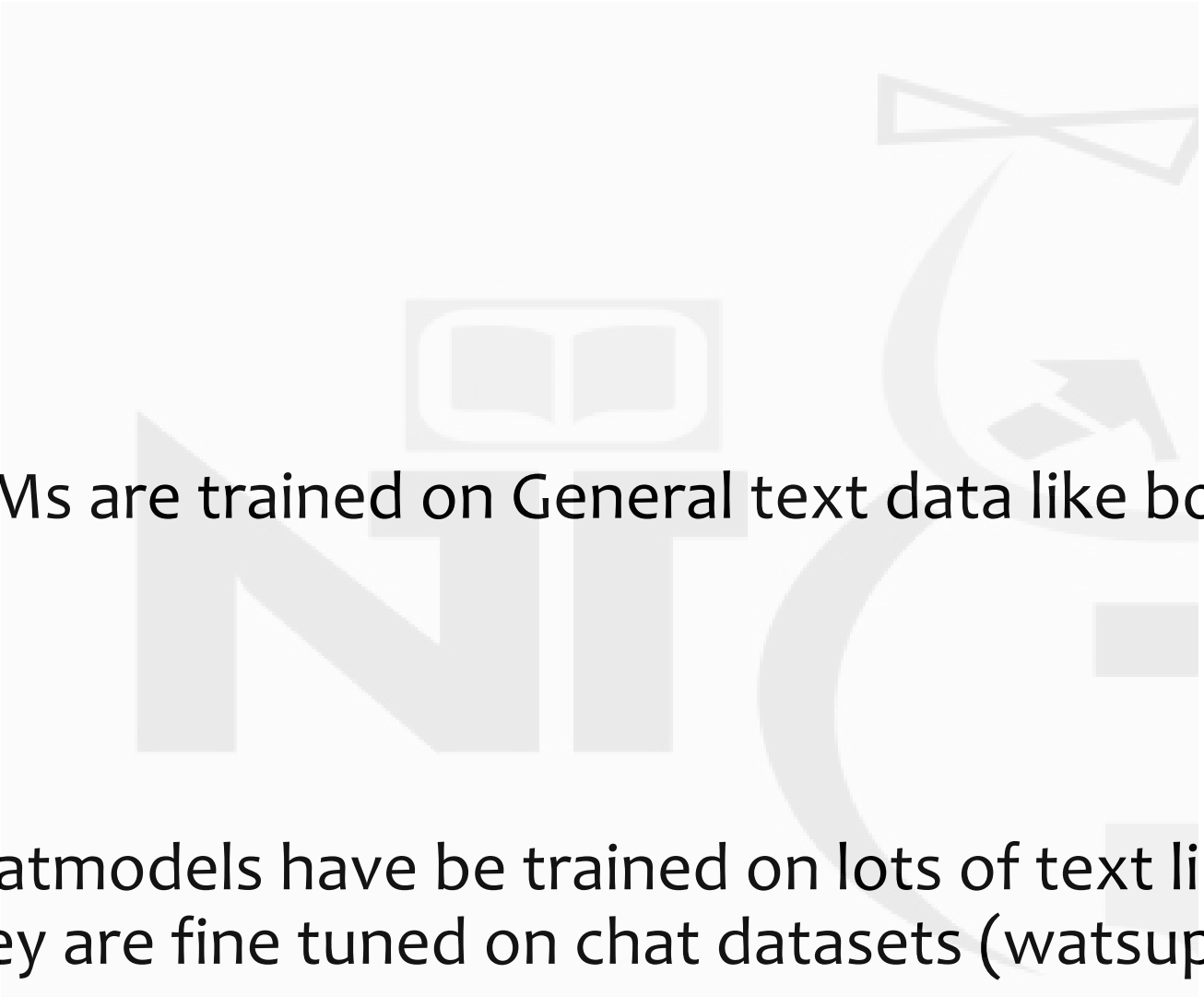
```
from langchain.chat_models import ChatOllama
from langchain.schema import HumanMessage

chat = ChatOllama(model="mistral") # or "mistral:7b-instruct"

response = chat([
    HumanMessage(content="Explain the benefits of open-source LLMs.")
])
```

Comparison: LLM vs ChatModel

| Aspect | LLM (Base Models) | ChatModel (Instruction-Tuned) |
|------------------|--------------------------------------|---|
| Goal | Single-shot text generation | Multi-turn, role-aware conversations |
| Input Format | Plain string | Structured messages (user/system/AI) |
| Context Handling | No memory, stateless | Tracks conversation history |
| Role Awareness | Not role-aware | Recognizes user, system, and AI roles |
| Training Style | Generic text sources | Tuned on dialog-heavy, chat-style data |
| Common Models | GPT-3, Mistral-7B, LLaMA-2-7B | GPT-4, Claude, LLaMA-2-Chat, Mistral-Instruct |
| Use Cases | Code gen, summarization, translation | Chatbots, tools, agents, customer support |

- 
- LLMs are trained on General text data like book, big text, Wikipedia ...
 - Chatmodels have be trained on lots of text like LLM but after that they are fine tuned on chat datasets (watsup , reddit ..etc)

In Practice:

- **LLM** is ideal for traditional NLP tasks (e.g., summarization, classification).
- **ChatModel** is preferred for conversational agents, assistants, or workflows.

The background features a large, light gray watermark of the NICE logo. The logo consists of the letters 'NICE' in a bold, sans-serif font. Above the 'I' is a stylized icon of an open book. To the right of the letters is a large, curved arrow pointing upwards and to the right.

Let's Code

Closed Source LLMs models code walkthrough

- OpenAI
- Gemini
- Claude



OPENAI LLM

We need API key to work with OpenAI

- Go to <https://platform.openai.com/>
- Create an account
- Login
- Click setting icon on top-right
- Click API keys in left pane
- Create an API token
- Copy the token while creating else it won't be available again
- You will need to recharge your openAI account , no free credits available anymore
- Get the env variable name from here : <https://github.com/openai/openai-python>

Project setup

- Create venv
- Install requirements
- Create a .env file
- Save your API key in env file

Langchain integration package with OpenAI

When we want to use OpenAPI

```
# Import the OpenAI wrapper from LangChain  
from langchain_openai import OpenAI
```

```
# Import the dotenv loader to read environment variables (like your OpenAI API key)  
from dotenv import load_dotenv
```

```
# Load environment variables from a .env file (required for accessing OpenAI)  
load_dotenv()
```

```
# Initialize the OpenAI language model (LLM) with the desired model name  
# 'gpt-3.5-turbo-instruct' is an instruction-tuned model that understands direct prompts  
# https://platform.openai.com/docs/models/gpt-3.5-turbo  
llm = OpenAI(model='gpt-3.5-turbo-instruct')
```

```
# Use the LLM to get an answer for the given prompt  
response = llm.invoke("What is the capital of India")
```

```
# Print the result to the console  
print(response)
```

LLM : String as input and string as output



Chat Model

OpenAI Chat model

```
from langchain_openai import ChatOpenAI
from dotenv import load_dotenv

load_dotenv()

model = ChatOpenAI(model='gpt-4', temperature=0, max_completion_tokens=10)
# model = ChatOpenAI(model='gpt-4', temperature=1.5, max_completion_tokens=10)

response = model.invoke("Write a 3 line poem on India")

print(response)

print(response.content)
```

Temperature

- **temperature=0.0** → more deterministic, focused, less creative (always similar answers)
- **temperature=1.0+** → more random, diverse, and creative (more surprising output)

Temperature Range: 0.0 to 2.0

| Temperature | Behavior | Notes |
|-------------|-----------------------------------|-------------------------------------|
| 0.0 | Fully deterministic | Always gives the most likely answer |
| 0.3 | Low randomness | Slight variation, mostly consistent |
| 0.7 | Balanced creativity & reliability | Good for chat and general use |
| 1.0 | High creativity | Diverse, imaginative outputs |
| 1.5+ | Very high randomness | Often creative, but less reliable |
| 2.0 | Maximum randomness | Can be chaotic or nonsensical |

Recommended Values

| Use Case | Recommended Temperature |
|---------------------|-------------------------|
| Factual Q&A | 0.0 - 0.3 |
| Chat, summarization | 0.5 - 0.7 |
| Poetry, stories | 0.8 - 1.2 |

Claude AI

- Go to <https://console.anthropic.com/>
- Create an account
- Login
- Go to billing
- Click 'BuyCredit'
- Complete payment
- Create api key
- Save the api key in .env file >> `ANTHROPIC_API_KEY`

Claude API chat model

```
from langchain_anthropic import ChatAnthropic
from dotenv import load_dotenv

load_dotenv()

model = ChatAnthropic(model='claude-3-5-sonnet-20241022')

result = model.invoke('What is the capital of India')

print(result)

print(result.content)
```

Gemini API key

- Setup could account – refer jupyter notebook
- Go to ai.google.dev
- Click “view gemeni docs”
- Click “create api key”
- Copy the api key
- Use this variable to save api key : `GEMINI_API_KEY`

```
from langchain_google_genai import ChatGoogleGenerativeAI
from dotenv import load_dotenv

load_dotenv()

model = ChatGoogleGenerativeAI(model='gemini-1.5-pro')

response = model.invoke('What is the capital of India')
print (response)
print(response.content)|
```

Closed Source Advantages

| Category | Benefits |
|------------------|---|
| Model Quality | Often state-of-the-art (GPT-4, Claude 3, Gemini) |
| Simplicity | Just call an API → no infrastructure or model management required |
| Reliability | High uptime, autoscaling, and support from large providers |
| Constant Updates | Regularly improved models, fine-tuning, instruction-following |

Closed Source Disadvantages

| Category | Drawbacks |
|--------------|--|
| Cost | Pay-per-token usage can be expensive at scale |
| Data Privacy | Your prompts/data go to external servers (may violate compliance) |
| No Control | Can't see model internals, can't fine-tune or change behavior deeply |
| Lock-in | Risk of vendor lock-in , changing prices, rate limits |

The background features a large, faint watermark of the NIT logo. The logo consists of the letters 'NIT' in a bold, sans-serif font. Above the 'I' is a square containing an open book icon. To the right of the 'NIT' text is a large, stylized 'C' that encircles a smaller icon of a hand holding a torch. The entire watermark is rendered in a light gray color.

Open Source Models

Open-Source Advantages

| Category | Benefits |
|---------------|--|
| Cost | Can run locally or self-hosted → avoids API costs |
| Customization | Full access to weights → allows fine-tuning , domain adaptation |
| Privacy | Data stays on your infrastructure (no external transmission) |
| Transparency | Source code, weights, and architecture are open → auditable and trusted |
| Integration | Can deploy anywhere: local, edge, cloud → high flexibility |

Open-Source Disadvantages

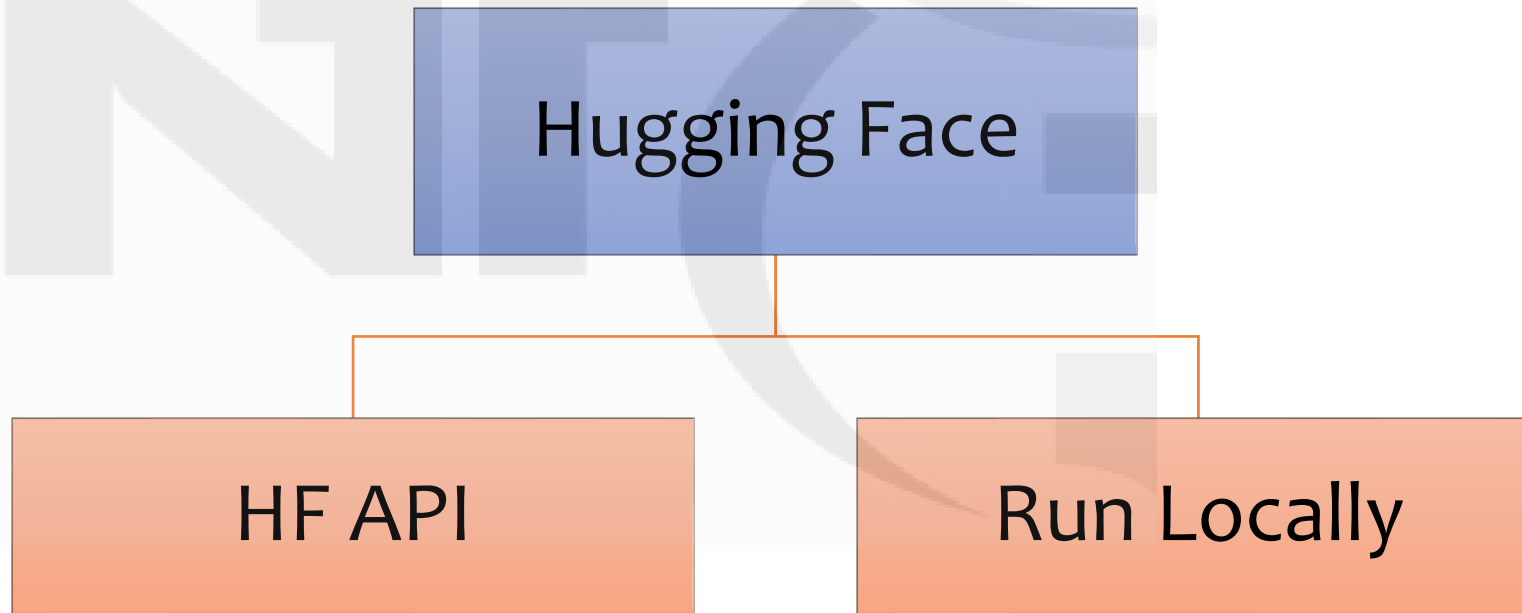
| Category | Drawbacks |
|-----------------------------|---|
| Model Quality | May lag behind state-of-the-art closed models in reasoning/chat |
| Setup Effort | Requires technical expertise to host, serve, and scale |
| Inference Cost | Needs good GPUs or optimized CPU inference engines (e.g., GGUF) |
| License Limits | Some models (e.g., LLaMA) have restricted commercial use |
| RLHF & Alignment | Most open-source models lack RLHF, which may reduce instruction-following quality and safety compared to closed models. |

Where can we find Open Source LLMs?

Hugging Face

Open-Source Models

- Hugging Face – 1000s of open-source models available



- Hugging face API has a free tier.



Working with Embedding Models

What are Embedding Models?

- Convert text into high-dimensional vectors.
- Enable similarity search, document retrieval, RAG pipelines.

```
from langchain.embeddings import OpenAIEmbeddings
embedder = OpenAIEmbeddings()
vectors = embedder.embed_documents(["Hello world", "LangChain is cool"])
```

```
graph TD; A[Embedding Models] --> B[OpenSource]; A --> C[Closed Source]
```

Embedding
Models

OpenSource

Closed Source

OpenAI Embedding Models

- <https://platform.openai.com/docs/guides/embeddings/embedding-models>
- By default, the length of the embedding vector is 1536 for text-embedding-3-small or 3072 for text-embedding-3-large.

OpenAI Embedding Model

Import this for embedding models

```
from langchain_openai import OpenAIEmbeddings
from dotenv import load_dotenv

load_dotenv()

embedding = OpenAIEmbeddings(model='text-embedding-3-large', dimensions=32)

response = embedding.embed_query("Delhi is the capital of India")

print(str(response))
```

This will give us 32-dimensional embedding vector

For single sentence

Document Embedding

```
embedding = OpenAIEmbeddings(model='text-embedding-3-large', dimensions=300)

# Define a list of documents to compare against
documents = [
    "C. V. Raman was awarded the Nobel Prize in Physics for his work on light scattering, k",
    "Homi Bhabha was the father of India's nuclear program and played a key role in foundin",
    "APJ Abdul Kalam, known as the Missile Man of India, later served as the President of I",
    "Satyendra Nath Bose collaborated with Einstein, leading to the development of Bose-Ein",
    "Venkatraman Ramakrishnan won the Nobel Prize in Chemistry for his work on the structur",
]

# Define the user query
query = "who worked with Einstein?"

# Convert the documents into embeddings (vector representations)
doc_embeddings = embedding.embed_documents(documents)
```

For processing a document, we call this method

Open Source Embedding model

```
from langchain_huggingface import HuggingFaceEmbeddings

embedding = HuggingFaceEmbeddings(model_name='sentence-transformers/all-MiniLM-L6-v2')

documents = [
    "Rose is known as the king of flowers",
    "Tulip is a popular spring-blooming flower",
    "Sunflower follows the direction of the sun"
]

vector = embedding.embed_documents(documents)

print(str(vector))
```

It maps sentences & paragraphs to a 384 dimensional dense vector space and can be used for tasks like clustering or semantic search.

Popular Hugging Face embedding models

| Model Name | Type | Languages | Description / Use Case | Link |
|---------------------------------------|-------------------|----------------|--|----------------------|
| all-MiniLM-L6-v2 | Text Embedding | English | Lightweight, fast; good for semantic similarity & clustering | Link |
| all-mpnet-base-v2 | Text Embedding | English | More accurate than MiniLM; great for sentence-level tasks | Link |
| intfloat/e5-base | Instruction-tuned | English | Retrieval-focused; use prompts like query: and passage: | Link |
| BAAI/bge-base-en-v1.5 | Text Embedding | English | Optimized for dense retrieval and reranking tasks | Link |
| nomic-ai/nomic-embed-text-v1 | General Embedding | English | Designed for clustering, search, and visualization | Link |
| paraphrase-multilingual-MiniLM-L12-v2 | Multilingual | 50+ languages | Multilingual semantic similarity and clustering | Link |
| intfloat/multilingual-e5-base | Multilingual | 100+ languages | Instruction-tuned multilingual embeddings for retrieval | Link |
| openai/clip-vit-base-patch32 | Multimodal | English (Text) | Joint text + image embeddings for cross-modal tasks | Link |

Document Similarity Search demo

- Convert the documents into embeddings
- Convert the query into an embedding
- Compute cosine similarity between query and each document embedding
- Find the index of the most similar document

Token Pricing

- <https://platform.openai.com/docs/pricing>

Summary

| Model Type | Purpose | Used In |
|-----------------|------------------------------|------------------------------|
| LLM | Text generation | Chains, Prompt Templates |
| Chat Model | Multi-turn dialogues | Agents, Conversational Apps |
| Embedding Model | Semantic search & similarity | Vector DBs, RAG, Q&A Systems |