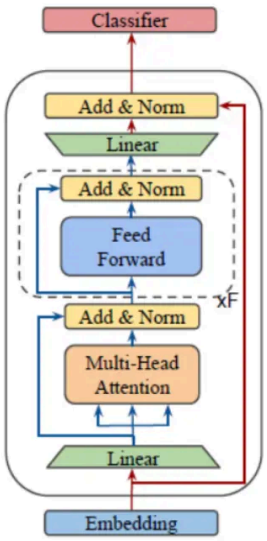
What we will learn today?

- [ ] What are Large Language Models?
- [ ] LLM SDK's
- [ ] Explore SDK's like OpenAI, Langchain, etc.
- [ ] Get free API keys from Google Gemini and Open Router.
- [ ] Use Gemini models to generate completions.
- [ ] Explore Structured Outputs.
- [ ] Explore Function Calling.
- [ ] Explore Image inputs.
- [ ] Difference between OpenAI Client and Langchain.
- [ ] Understanding Model Parameters.

Large Language Models:



- A *Large Language Model* is a DL model trained on massive corpora of data to understand and generate human-like text.
- In this course, we will focus on using prebuilt foundational models instead on building our own.
- In order for LLM to generate the best responses it needs to be given a proper context i.e *instructions, data, etc*
- When building AI Applications the first step is to select an appropriate model for the respective task.

LLM SDK's:

1. ***OpenAI :***

- Official OpenAI SDK to connect to it's models like GPT-4, o3, etc.
- It's API standard is followed by most service providers.
- Can be used to connect to any LLM exposed by `/v1/api/chat/completions` API.
- Paid API.
- Free services:
    1. Open Router free models.
    2. Gemini OpenAI compatibility(we will use this).
    3. Local Services like LM Studio, Ollama, vLLM.

2. ***Langchain :***
    - Not an official LLM SDK but rather an abstraction layer to call models.
    - It makes development a lot easier.
    - Switching between providers is very easy, just change the *ChatModel* and everything will still work.
    - We will use this as it is beginner friendly and also makes Tool Calling easier.

OpenAI Client:

- Most AI cloud providers as well as local apps serve LLM's that can be accessed by *openai* client.
- Exposes LLM via:
    1. *Completions API* :
        - The original API provided by OpenAI.
        - Now deprecated in favor of **Responses API**.
        - Supports `text-generation`, `structured output`, `function calling`, `audio & vision understanding`.
        - 3rd party providers use this API.
    2. *Responses API* :
        - Latest API provided by OpenAI.
        - Supports `tool-calling`, `mcp`, `web_search`, etc.
        - Not implemented by 3rd party providers.

Langchain:

- It provides various LLM providers integration like:
    1. OpenAI
    2. Gemini
    3. Groq
    4. LiteLLM

    ... and more.

Structured Outputs:

- Guide LLM to provide output in the specified format

```
from pydantic import BaseModel
from typing import List
from openai import OpenAI
from langchain_openai import ChatOpenAI

client = OpenAI()
llm = ChatOpenAI(model="<model-name>")

class Todo(BaseModel):
    title: str
    completed: bool

class Todos(BaseModel):
    todos: List[Todo]

# OpenAI official client approach
response = client.chat.completions.parse(
    model="<model-name>",
    messages="Create a todo list for developing a simple login page.",
    response_format=Todos
)
```

```
print("OpenAI official client response:")
print(response.choices[0].message.parsed)

# LangChain approach
response = llm.with_structured_output(Todos).invoke(
    "Create a todo list for developing a simple login page."
)
print("Langchain response:")
print(response)
```

```
OpenAI official client response:
Todos(
    todos=[
        Todo(title="Create a login webpage", completed=False),
        Todo(title="Design user interface", completed=False),
        Todo(title="Implement authentication logic", completed=False),
        Todo(title="Add form validation", completed=False),
        Todo(title="Style the login form", completed=False),
        Todo(title="Test login functionality", completed=False)
    ]
)

Langchain response:
Todos(
    todos=[
        Todo(title="Create a login webpage", completed=False),
        Todo(title="Design user interface", completed=False),
        Todo(title="Implement authentication logic", completed=False),
        Todo(title="Add form validation", completed=False),
        Todo(title="Style the login form", completed=False),
        Todo(title="Test login functionality", completed=False)
    ]
)
```

Function Calling:

- Provide functions to the llm so that it can execute it to get the result it doesn't know.

```
  from openai import OpenAI
  client = OpenAI()

  tools = [
  {
    "type": "function",
    "function": {
      "name": "get_current_weather",
      "description": "Get the current weather in a given location",
      "parameters": {
        "type": "object",
        "properties": {
          "location": {
            "type": "string",
            "description": "The city and state, e.g. San Francisco, CA",
          },
          "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]},
        },
        "required": ["location"],
      },
    }
  }
]
  messages = [{"role": "user", "content": "What's the weather like in Boston today?"}]
  completion = client.chat.completions.create(
    model="gpt-5",
    messages=messages,
    tools=tools,
    tool_choice="auto"
  )

  print(completion)
```

```python
from pydantic import BaseModel, Field


class GetWeather(BaseModel):
    """Get the current weather in a given location"""

    location: str = Field(..., description="The city and state, e.g. San Francisco, CA")


llm_with_tools = llm.bind_tools([GetWeather])
response = llm_with_tools.invoke("What's the weather in Mumbai")
```

Image Inputs:

- Pass image data to LLM's by base64 encoding.

```python
import base64


def encode_image(image_path):
  with open(image_path, "rb") as image_file:
      return   base64.b64encode(image_file.read()).decode("utf-8")
```

```
{
    "type": "image_url",
    "image_url": {
        "url": "{base64_image}",
    }
}
```

Langchain vs OpenAI:

Test Results for 5 Runs:

Run	*SDK*	*Output Tokens*	*Response Time (s)*
1	**Langchain**	52	3.514
1	**OpenAI**	49	3.894
2	**Langchain**	49	4.534
2	**OpenAI**	48	4.641
3	**Langchain**	25	2.952
3	**OpenAI**	46	3.908
4	**Langchain**	27	3.4
4	**OpenAI**	48	3.713
5	**Langchain**	26	3.871
5	**OpenAI**	48	3.793

Average Results:

SDK	*Output Tokens*	*Response Time (s)*
Langchain	35.8	3.6542
OpenAI	47.8	3.9898

Test Parameters:

- *Input Tokens*: 585

- Tested on Tool Calling with `web_search`, `summarize_doc`, `generate_visualization` tools.

Model Parameters:

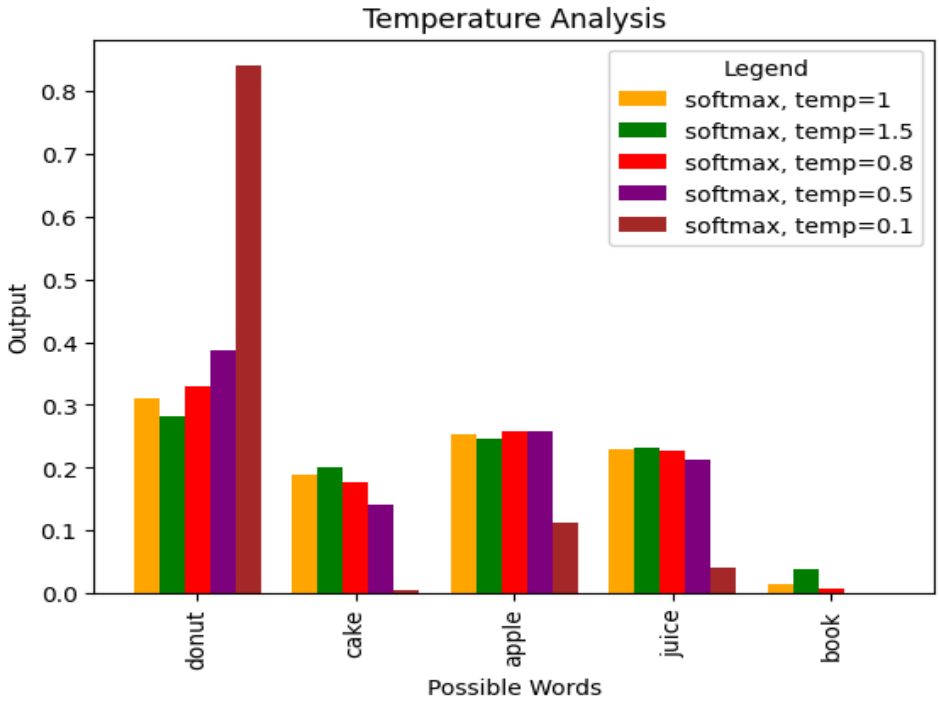
1. *Temperature* :
   - Influences variety (creativity).
   - How probabilistic you want it to be?
   - Before setting temperature:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

   - After setting temperature (T):

$$softmax(z_i;\ T) = \frac{e^{\frac{z_i}{T}}}{\sum_{j=1}^{K} e^{\frac{z_i}{T}}}$$

   - As temperature decreases, the probabilities become sharp, one of the probs will be close to 0.
   - As temperature increases, the probabilities become more uniform.



2. *Top P* :
   - Sets the cumulative probability value.
   - Selects the tokens up to that value.
   - $p = 0.9$ selects top tokens whose prob add up to 0.9.

> **NOTE**
>
> *Avoid* using both *temperature* and *top_p* togther.

3. *Top K* :
   - Selects the top `n` tokens.
   - `top_k = 1` will greedy select the most probable token.
   - *Deprecated* in favor of *top_p*.
