**Generative AI**

Generative AI, or Gen AI, refers to artificial intelligence systems that create new content based on patterns learned from data. These systems, often built on machine learning models like Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and especially large language models (LLMs), can generate images, text, audio, video, and more.

Some examples include:

* **Text Generation**: GPT models (like ChatGPT) can generate coherent text based on prompts, enabling applications in chatbots, summarization, content creation, and more.
* **Image and Art Creation**: Tools like DALL-E or Stable Diffusion generate new images based on textual descriptions.
* **Code Generation**: Codex and other models generate programming code based on natural language instructions.

**Configuration of Together API**

As open api key is paid service I have used the TogetherAPI key which gives 5$ free credit. I write is in **together\_ai.py**  
  
**Configuration of LangChain**

In LangChain, "memory" refers to the way the system retains and recalls information from past interactions within a conversation chain, enabling more contextual and coherent responses. Here’s a breakdown of different types of memory in LangChain:

**1. Buffer Memory**

* **Description**: Stores recent interactions in a simple buffer (like a list). Buffer memory retains a limited number of recent messages in the conversation.
* **Use Case**: Ideal for shorter conversations where only recent context is necessary, without needing to remember the entire conversation history.
* **Example**: Retaining the last 5 messages exchanged in a customer support conversation.

**2. Conversation Summary Memory**

* **Description**: Summarizes and stores information from the entire conversation up to a point, allowing memory to be compact and concise.
* **Use Case**: Best for longer conversations where a summary suffices for context, rather than retaining all details verbatim.
* **Example**: Summarizing a user’s preferences in a shopping assistant interaction.

**3. Knowledge Graph Memory**

* **Description**: Creates a knowledge graph by extracting entities and relationships from interactions, organizing them in a structured format.
* **Use Case**: Useful for applications that involve complex entities and relationships, such as tracking project updates or managing customer details.
* **Example**: Building a knowledge graph of user preferences, such as favorite products, locations, and interactions.

**4. Vector Store Memory**

* **Description**: Stores past interactions as embeddings in a vector store, enabling similarity search based on semantic similarity of past interactions.
* **Use Case**: Suitable for cases where retrieving similar past interactions can help in generating a more relevant response, especially in Q&A or recommendation settings.
* **Example**: Retrieving past questions and answers based on their similarity to new questions, which is helpful in an FAQ bot.

**5. Entity Memory**

* **Description**: Specifically tracks key entities in a conversation (like user’s name, preferences, etc.) to retain their state and context throughout interactions.
* **Use Case**: Ideal for personalized conversations where specific information about the user or other entities needs to be preserved and updated as the conversation evolves.
* **Example**: Tracking user preferences such as favorite genres in a book recommendation system.

**6. Combined Memory**

* **Description**: Combines multiple memory types to leverage the benefits of different memory structures in a single conversation.
* **Use Case**: Useful when a mix of recent message retention, summary, and entity-specific memory is needed for complex conversational flows.
* **Example**: A virtual assistant for customer support that uses buffer memory for immediate context, summary for session history, and entity memory to track user details.

**Configuration of Hugginface API with Langchain**

I have used the **hugging face** free api key and one of the free models of **Mistral AI** to integrate with langchain for conversational chantbot. That code is present in **hugg\_mistral7b\_langchain**. I have use the **conversation buffer memory**.

**Memory config in langchain**

I have given some extra configurations to conversation buffer memory such as buffer\_size. And I am saving the conversation that are stored in langachain in to json file of the local file system and restoring the conversation into the langchain and reusing the json file conversation when I am running the bot again. This code is present in the **mistral\_lang\_memory\_config.py** and we can see the conversation in the **memory.json** file.