Project Plan

Table of Contents

Project Background

With the rise of ChatGPT in 2023, we can see the potential of using large language models (LLMs) as general assistants for different kinds of language tasks. The pre-trained model processed a great statistical understanding of language that it can generate wanted answers based on user prompts. However, I see 3 potential changes from the current stage of LLMs.

First, most virtual assistants or chatbots heavily rely on text as the interface between humans and machines. For example, ChatGPT does not provide channels like audio files or documents as input without extension. Some niche assistants like midjourney or Phind (code assistant) are also text-based. I think it will hinder the adaptation of using such technology because we will require a dedicated input device like a keyboard to interact with the LLM. Places like counters of restaurants, and receptions may have access to a large screen and network but not a keyboard. In that case, we cannot leverage LLM to handle customer's inquiries.

Second, most of the chatbots or voice assistants have no virtual avatar. By adding a virtual avatar to the chatbot, we can make the user experience more personalized and strengthen user connection with the services provided by the chatbot. Ultimately building trust and loyalty from customers. Not to mention adding an avatar can help to differentiate the chatbot from other existing services and create visual appeal.

Lastly, the chatbots we use nowadays may show signs of hallucination. The training materials are usually not up to date. If we ask about things that occurred very recently, the LLM cannot obtain such knowledge. Also, due to the randomness during the process of text generation, we may even obtain information that does not exist in the real world. Therefore, by combining chatbots and a knowledge base, we can provide up-to-date and accurate information with citations as well.

There are some start-ups like Asiabots that create similar applications that provide multi-media chatbots with virtual avatars. However, they are not free nor open-sourced. Also, they usually do not provide service about connecting to a knowledge base.

At the end, I want to create a chatbot connected to knowledge that can answer questions with specific domain knowledge and a virtual avatar.

Project Objective

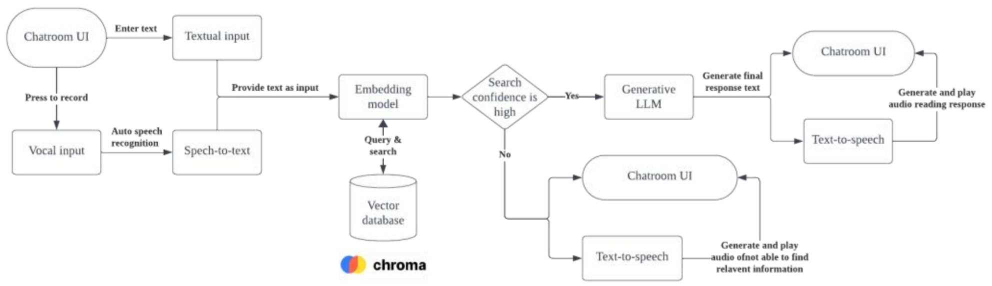
This project aims to develop a chatbot with a web-based UI using Python package named 'Gradio'. This package allows me to quickly prototype and implement most of the logic of the application with its native support of machine learning models. The chatbot is expected to provide information about meals and canteens of HKU retrieved from a knowledge base. The chatbot should also be able to take plain text and audio file as input. The chatbot should generate responses from a LLM and display the responses on the UI. The chatbot should be able to identify queries that have no relevant response.

To conclude with, this project is expected to:

1. Create web app UI for the chatbot that can record transcribe the audio with auto speech recognition model. Also, the chatbot can take textual input from keyboard.
2. Collect information and create a vector database for queries.
3. Make the chatbot able to query from a vector database.
4. Make the chatbot able to send query results and chat history to LLM as prompts.
5. Make the chatbot able to generate and play audio of a voice reading the response.
6. Create testing scripts to evaluate each model choice based on speech and accuracy.
7. Create example test cases to demonstrate capability of the application.

Project Methodology

To begin with, the following is the workflow of the entire application.



The following part is to list expected functionality of each component:

1. Chatroom UI
   - Text field for user to type their queries.
   - Button to start recording their vocal queries.
   - Text area to display chat history between user and the voice assistant.
   - Button to clear chat history.
   - View that display virtual avatar that read out the response to user.
2. Speech-to-text
   - Accept recording audio as input.
   - Perform auto speech recognition to transcribe text of the audio.
   - Return transcribed text as output.
3. Embedding model & Vector database
   - Embedding model converts text to a vector.
   - Vector database uses the converted vector to perform similarity search inside the database.
   - Return the confidence and result.
4. Generative LLM
   - Accept search result from vector database as context of the query.
   - The LLM is prompted to generate textual output in a predetermined manner.
   - Return final textual response as output.
5. Text-to-speech
   - Accept response text from LLM as input.
   - Create and play audio generated based on textual input.

The following part is to list models to be used in each component and reason behind:

1. Speech-to-text
   - nvidia/stt_en_fastconformer_ctc_large (best accuracy with very low latency, but I could not initialize the model on website so there could be some bugs)
   - openai/whisper-large-v2 (top3 accuracy with high latency)
   - Metrics are found in Open_ASR_Leaderboard.
2. Embedding model & Vector database
   - bge-small-en (highest score in retrieval tasks, very small model size of 0.13GB and small embedding dimensions of 384)
   - Metrics are found in MTEB

- Use Chroma as vector database (open sourced, free, able to run in memory, many integration with platforms like Hugging Face and LangChain)

3. Text generation
    - llama-2-7b-chat-hf (recently released LLM, performance of 70B parameters are on par with GPT3.5. Specifically, finetuned on safety content generation, score 0 on Toxigen and 57.04 in TruthfulQA. Choosing smaller model for latency issue)
4. Text-to-speech
    - Google Text to Speech API for simplicity.
    - If time allows, I want to fine-tune a text-to-speech model with my voice or some celebrities.

Technology introduction

This project will mostly use language models hosted on Hugging Face, which is a company that aims to develop tools and a platform for people to train and share machine learning models. Hugging Face provides many state-of-the-art models in natural language processing (NLP). The reason for using Hugging Face is that all models available are open-sourced and free to run in a local environment. Also Hugging Face provides options for developers to access via API. This API approach can shorten the development time because developers can quickly and easily swap to test models and even fine-tune existing models with minimal codes.

Usually, LLMs process some general knowledge about the world with an abundance of training materials. However, they perform poorly when asked about internal business information. This is because the data used to train LLM does not contain that information. Therefore, I choose to integrate vector database and embedding model to handle user's queries. The embedding model converts sentences to a vector the context of the sentences, and then the vectors can be stored in a vector database as a knowledge base. When the user makes a query, the query is first converted to a vector and use it to perform a similar search inside the vector database. In the end, we can retrieve relevant information as the basis of the chatbot's responses.

The relevant information is then used as a prompt to the actual generative chatting LLM to generate a response to users. The response is then updated on the UI and played as audio in the background along with the virtual avatar. This approach allows generative LLM to answer users with accurate answers with a customized tone or a pre-defined template. As a result, the format and tone of responses are controlled while providing relevant information.

Project Schedule and Milestones

The following table is a temporary schedule with expected deliverables. It has included both official deadlines and personal deadlines that used for personal monitoring.

Deadline	Tasks
1 Oct 2023	Submit detailed project plan. Set up project website.
1 Oct 2023	Create UI for chatbot with Gradio (Python). It can accept keyboard typing & record voice. The chatbot at this moment should answer by repeating keyboard input or "recording" when audio input is detected.
8 Oct 2023	Integrate Whisper model to perform auto speech recognition. The chat UI should print the transcribed text directly.
1 Nov 2023	Create a local vector database with Chroma. Create a Python script to convert pdf stored in a folder named 'data' to embeddings. The process of chunking and embedding is done with the help of a package named 'LangChain'. The embeddings will be stored in the vector database with a fold named 'embeddings'
15 Nov2023	Integrate the vector database to the chatbot. The chatbot can create connection to database at launch. Embed user input as query. Perform similar search in the database and print the retrieved information on UI.
15 Dec 2023	Integrate LLM to paraphrase and format the text output. The chatbot for now should be able to process single query at a time, without referring the chat history. The LLM is prompted to apologize when no relevant information found in the database.
1 Jan 2023	Integrate Google text-to-speech API to generate voice audio to play.
8-12 Jan 2024	First presentation
21 Jan 2024	Submit interim report, preliminary implementation.
15 Feb 2024	Gather data source of HKU canteen and create a set of PDFs to store information related. Then convert the corpus collected to embeddings to replace the database used in earlier stage.
15 Mar 2024	Fine-tune a text-to-speech model with Hugging Face with my own voice or use the TTS service in Azure. The objective is to provide a lively speech without sacrificing too much latency.
15 Apr 2024	Create and pass UAT for overall user experience: 1. Create 3 intended use case for demo purposes 2. Test behaviour against toxic input 3. Null input (empty string or silent audio) 4. Extreme long input or long waiting time 5. Audio quality
23 Apr 2024	Submit finalized tested implementation, final report.
26 Apr 2024	Project exhibition.