Companion Chatbot

杨永卓
Fudan University
AICS
22307140078@m.fudan.edu.cn

***Abstract*—In this project, we have set up a companion chatbot on a cloud server. The chatbot is designed from the perspective of a human companion and is equipped with functionalities such as voice chat, visual capture, and long-term and short-term memory. Additionally, as an AI conversational partner, it includes internet search capabilities to acquire knowledge from the web. To ensure smoother communication with users, we have optimized the system to significantly reduce latency. Finally, users can interact with the AI chatbot on the server by sending requests via the HTTP protocol from their local devices.**

***Index Terms*—Companion chatbot, Voice chat, Visual capture, Long-term memory, Short-term memory, Latency reduction, HCI**

I. Introduction

With the rapid pace of life and technological advancements, people increasingly face feelings of loneliness and psychological stress, and traditional modes of interpersonal communication sometimes fail to provide adequate support. Thus, chatbots have been designed as a technological solution to fill this emotional void, offering users instant and reliable companionship and emotional exchange. By mimicking human conversation, chatbots can provide not only informational assistance but also emotional interactions, such as comfort, encouragement, and empathy. Their design goal is to establish emotional connections with users through text or voice, making people feel understood and supported. This technology has shown great potential in alleviating psychological stress, providing social interaction, and enhancing the emotional aspect of users' daily lives. Furthermore, improvements on this foundation can expand their functionality to include convenient online services such as customer support, information inquiry, and educational tutoring.

The rapid development of large models has made all this possible, but relying solely on large models as chatbots currently has three serious flaws: First, communication between people, whether emotional or conversational, needs to be based on previous exchanges, requiring chatbots to have some memory. Second, many communications are based on the surrounding environment, such as Chinese people typically starting conversations with weather conditions as a way to exchange pleasantries, hence chatbots need the ability to observe their surroundings. Lastly, for a chat companion, we hope to discuss fresh topics rather than always talking about known things, which requires chatbots to have the capability to acquire new knowledge.

Unfortunately, current large models alone cannot effectively perform these tasks. Therefore, this project combines various technologies to integrate multiple functions into a chatbot centered around a large model, creating a multifunctional chatbot.This project introduces a companion chatbot that is meticulously designed as a human-like conversational partner and hosted on cloud servers. This chatbot integrates features such as voice chatting, visual capturing, and both long and short-term memory, providing users with a rich and interactive experience.

The chatbot is equipped with voice interaction functionality, which is a basic requirement for personal interaction. Moreover, its designed visual capturing capability enables the robot to process and respond to visual inputs, allowing it to ”see what you see,” and not limited to just text and voice. This multimodal interaction ensures that users can communicate with the robot in a more intuitive and comprehensive manner.

Memory management is another critical aspect of the system. The chatbot is equipped with long and short-term memory capabilities, allowing it to retain and recall past conversations and respond relevantly based on the existing chat content. This memory ability helps provide a more personalized and context-relevant user experience, enhancing the continuity and familiarity in interactions.

Additionally, to enhance the conversational capabilities and fully utilize its functions, the chatbot integrates internet search capabilities. This enables it to obtain real-time information from the web, thus providing accurate and up-to-date responses. This feature is particularly important for answering factual questions and dynamically expanding the robot's knowledge base, allowing the robot to stay informed about current events and discuss the latest matters with individuals.

Reducing latency is one of our system's key optimization focuses. In normal human-to-human communication, the back-and-forth interaction is essential for ensuring communication. By minimizing response times, we ensure that interactions with the chatbot are quick and efficient, thereby increasing user satisfaction and engagement.

Users can interact with the chatbot via HTTP protocol from their local devices conveniently, using the flexibility and accessibility of the Flask architecture. This setup not only simplifies user access, aligning with real-world usage scenarios but also ensures that the source code can be easily updated and maintained.

II. Related Work

In this project, the various functionalities of the chatbot are comprised of multiple modules working in coordination. Next, I will provide an overview of these modules, briefly summarizing

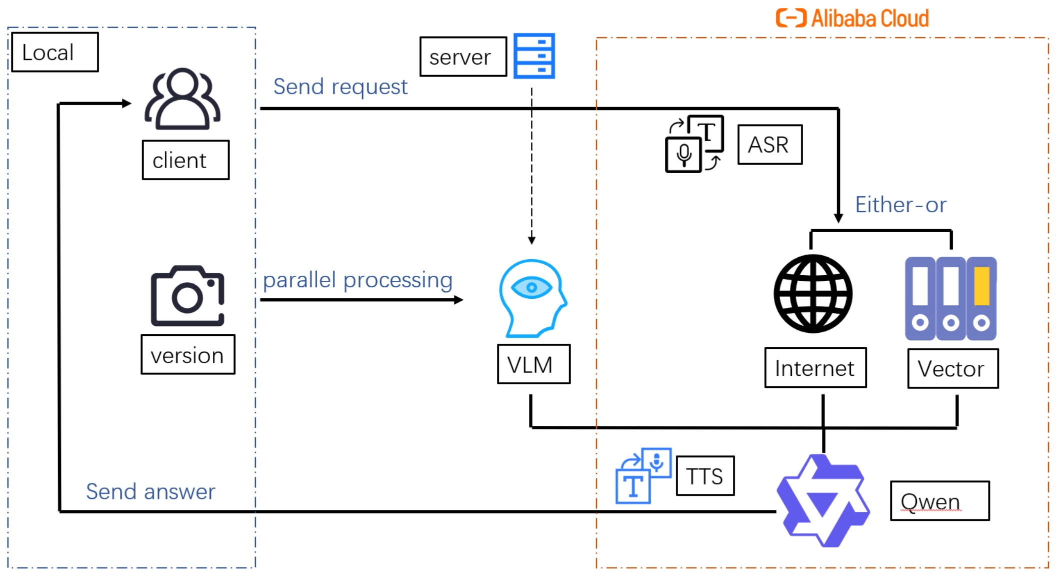


Fig. 1. Workflow of the companion chatbot system

their related work, thereby illustrating the diverse technologies that make up this project.

ASR & TTS — Automatic Speech Recognition (ASR) and Text-to-Speech (TTS) technologies play a crucial role in human-machine interaction. ASR technology converts user's spoken information into text, while TTS technology transforms text information into spoken output. These technologies provide foundational support for this project, allowing user's voice inputs to be converted into text, then processed by a large language model, and finally converting the model's text responses back into voice, so that users can hear the answers directly.

VLM — Visual Language Models (VLMs) in this project are used to identify objects "seen" by the chatbot. VLMs analyze received images and perform image-to-text conversions, enabling the large language model to "see" and understand the user's surroundings. This process enhances the robot's environmental perception capabilities, allowing it to provide richer and more accurate responses during interactions.

LLM — The Large Language Model (LLM) is the core of this project. It processes various types of user inputs, such as conversational content, surrounding visual information, and past chat records, to generate appropriate interactive responses. The advanced language processing capabilities of this model ensure that communication with users is both natural and efficient, maintaining smooth interactions in complex dialogue scenarios.

Vector Database — The vector database used in the project is designed for efficient storage, querying, and retrieval of vector data. By converting important memories into vector form for storage, this database can quickly retrieve related memories through similarity searches when users make subsequent requests, supporting the chatbot's long-term memory functionality and making conversations more personalized and coherent.

Web Spider — A web spider technology in this project is used to implement efficient web search functionality. By crawling web information related to user queries and passing the collected data to the large language model for processing and summarization, the chatbot can obtain real-time information during interactions with users.

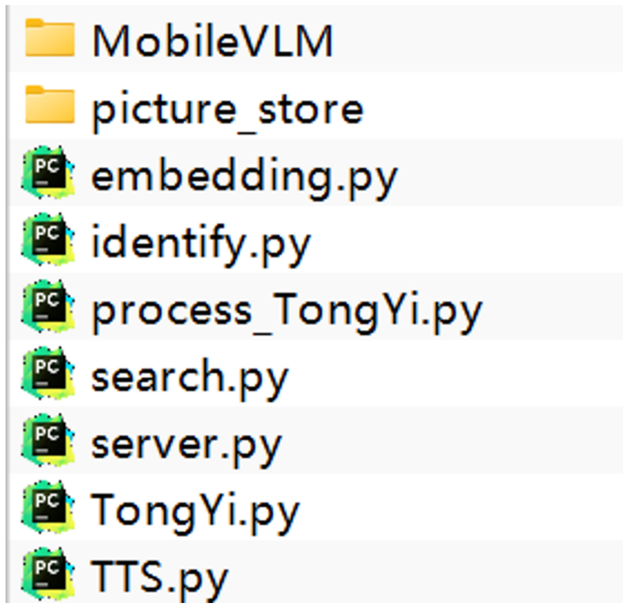
Flask — Using Flask, a lightweight web application framework, we have built an information exchange channel between the server and local devices. This framework supports the transmission of user's voice inputs to the cloud server via HTTP protocol, and after processing, the responses are sent back to the local device.

III. Method

In this part, we will provide a detailed introduction to the specific implementation process of each functional module. We will explain the APIs or models used, and showcase the code for important parts.

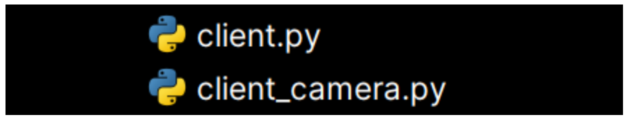
A. Workflow

Before delving into the various components of the project, let' s provide a brief overview of the workflow. As illustrated in Figure 1, the process can be broadly divided into three stages: local operations, server processing, and Alibaba Cloud services. In the local operations stage, the system is responsible for receiving voice input and capturing visual information. These two types of information are swiftly transmitted to the server side via separate paths. The server is equipped with a Visual Language Model (VLM) that recognizes .jpg format image files and outputs corresponding textual descriptions. Concurrently, voice data (.pcm format) is also sent to the server and processed using Alibaba Cloud' s Intelligent Speech Interaction API for speech recognition. Subsequently, based on the user' s voice content, the chatbot determines whether to search for relevant information online or review chat history to extract context-related information for the current conversation. At this point, the chatbot has integrated visual information, current voice data, and either online search results or historical chat data. Next, these diverse pieces of information are consolidated and submitted to the Tongyi Qianwen large model for comprehensive processing. We utilize API calls to engage this large model to generate responses. The generated responses are recorded and, by invoking Alibaba Cloud' s Text-to-Speech (TTS) service API, converted into audio signals, which are ultimately sent back to the local endpoint. Finally, upon receiving the audio signals at the local end, the system plays them back, thus completing a full interaction cycle. This workflow is designed to ensure the accurate transmission and efficient processing of information, delivering a seamless and intelligent user interaction experience.



MobileVLM
picture_store
embedding.py
identify.py
process_TongYi.py
search.py
server.py
TongYi.py
TTS.py

Fig. 2. server code structure.



client.py
client_camera.py

Fig. 3. client code structure

B. Codereview

The server-side code of this project consists of eight files, as depicted in Figure 2. The first file, MobileVLM, is used for deploying the VLM visual model locally, containing the model files and additional scripts written for utilizing the model through function calls. The pictur_store file is used to store all visual information; the local camera takes a photo every thirty seconds and sends it to this folder for storage.

- **MobileVLM:** Deploy the VLM visual model locally, includes model files and scripts for model function calls.
- **picture_store:** Stores all visual information, receives photos from the local camera every thirty seconds.
- **embedding.py:** Implements storing processed chat records into a vector library.
- **identify.py:** Uses Alibaba's intelligent voice API to convert speech to text.
- **process_TongYi:** Sends all chat records to a large model for summarization and saving.
- **search.py:** Executes web searches using a crawler.
- **server.py:** Core file with definitions and logic for the three main routes.
- **TongYi.py:** Contains two functions, *call_with_messages* and *call_with_search*, for interacting with historical memory and conducting web searches.
- **TTS.py:** Converts text to speech using Alibaba's intelligent voice API.

The local operations are handled by two files:

- **client.py:** Manages voice recording, photo taking, server communication, and text playback.
- **client_camera.py:** Manages the visual logic, capturing and sending photos to the server every thirty seconds.

C. ASR & TTS

This module is responsible for implementing the speech-to-text and text-to-speech functionalities. The implementation method directly calls the APIs provided by Alibaba Cloud's intelligent speech services, which offer robust ASR (Automatic Speech Recognition) and TTS (Text-to-Speech) capabilities.

In the ASR implementation, the first step is to load the audio file and read its binary data. By creating an instance of NlsSpeechTranscriber, various callback functions are configured, such as those for sentence start, sentence end, recognition start, error handling, and connection closure. The audio data is then sliced into chunks of 640 bytes and sent to the server for processing.

For the TTS implementation, the text file's content needs to be read first. An instance of NlsSpeechSynthesizer is then created, with appropriate callback functions configured to handle the generated data, errors, and completion status during the synthesis process. By calling the start method with the text and voice type, the text-to-speech synthesis is initiated, and the generated PCM data is written to a file.

Through these processes, Alibaba Cloud's intelligent speech services provide powerful and flexible speech processing capabilities, enabling efficient speech-to-text conversion and text-to-speech synthesis.

D. VLM

Visual-Language Models (VLM) have become a research hotspot in the field of artificial intelligence. By integrating Large Language Models (LLM) with multimodal capabilities, VLMs exhibit unprecedented versatility. This project uses MobileVLM V2, introduced by Meituan and Zhejiang University. Due to MobileVLM's powerful performance with a small parameter size and its comprehensive on-device real-time inference solution, MobileVLM has gained widespread attention from the open-source community both domestically and internationally upon its release.

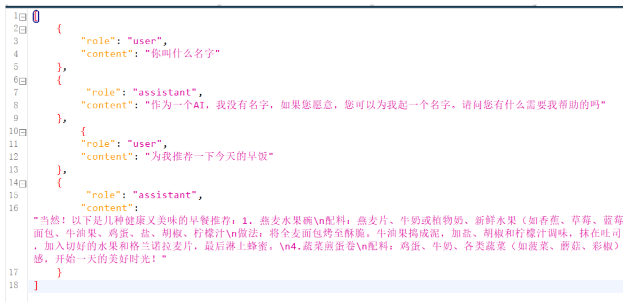
MobileVLM V2 builds on the advantages of its predecessor, introducing significant improvements with novel architectural designs, a customized training scheme tailored for mobile VLMs, and enriched high-quality data. These enhancements not only elevate the performance of MobileVLM V2 but also provide the industry with new insights for on-device visual-language model solutions [1].

For local deployment, follow these steps to complete the model setup: first, prepare the environment and Python packages. Then, download the MobileVLM model and the clip-vit-large-patch14-336 model from Huggingface, configuring the paths for both models accordingly. Next, specify the absolute path of the image to be recognized by the model. After providing an appropriate prompt, the local MobileVLM model can begin recognizing the image content.

To improve the model's image recognition accuracy, we used the following prompt after multiple comparisons: "Summarize the content of the image. Use several words." This prompt clearly defines the task for the model. By using a summary approach, the model can quickly extract the key information from the image, while the phrase "simply use several words" ensures that the visual information provided by the model is focused and avoids the interference of too many elements on the model's visual judgment.

E. Long-short memory

To make the chatbot more human-like, the project implements a long-term and short-term memory mechanism, which is one of the unique features of this project. We envision a conversation scenario between two people: when they mention a conversation from a few days ago, they might remember the general content and information, but not the specific details of each sentence. This represents the "long-term memory" in our project. Meanwhile, during an actual conversation, they can remember the last few sentences from each other and continue the conversation based on that information, representing the "short-term memory."



```
[
    {
        "role": "user",
        "content": "你叫什么名字"
    },
    {
         "role": "assistant",
        "content": "作为一个AI，我没有名字，如果您愿意，您可以为我起一个名字。请问您有什么需要我帮助的吗"
    },
        {
        "role": "user",
        "content": "为我推荐一下今天的早饭"
    },
    {
         "role": "assistant",
        "content":
"当然！以下是几种健康又美味的早餐推荐：1. 燕麦水果碗\n配料：燕麦片、牛奶或植物奶、新鲜水果（如香蕉、草莓、蓝莓
面包、牛油果、鸡蛋、盐、胡椒、柠檬汁\n做法：将全麦面包烤至酥脆。牛油果捣成泥，加盐、胡椒和柠檬汁调味，抹在吐司
，加入切好的水果和格兰诺拉麦片，最后淋上蜂蜜。\n4.蔬菜煎蛋卷\n配料：鸡蛋、牛奶、各类蔬菜（如菠菜、蘑菇、彩椒）
感，开始一天的美好时光！"
    }
]
```

Fig. 4. Short-term memory standard format

To mimic short-term memory in human conversations, this project records each dialogue between the user and the chatbot, organizes it into a standard format (as shown in Figure 2), and adds it to an answer.json file. With each new dialogue, the content of the file increases accordingly. In each round of dialogue, the contents of the answer.json file are provided to the large model so that it can recall the content of the last few conversations.

For long-term memory, at the end of each conversation, an independent function is called to pass all the contents of the answer.json file to the large model. The model organizes this information according to the format of time, place, people, and main events, and stores it in a vector database (details to follow). During each conversation round, based on the user's input, the system searches the vector database for the most relevant conversation record and includes it in the current prompt to help the chatbot recall the chat history.

By adopting the long-term and short-term memory mechanism, this project successfully simulates the memory characteristics of human conversations, enabling the chatbot to remember not only short-term information from the conversation but also recall long-term memories when needed. This approach improves the coherence and context understanding of the dialogue, making the interaction between the robot and the user more natural and human-like.

F. LLM

In this project, the Qwen-Turbo large model provided by Alibaba is used to accomplish tasks. The model's invocation logic is divided into two methods: using web search or using historical chat records to answer questions. By default, the model uses the historical chat records method, but if the user's interaction begins with "web search," it switches to the web search method for responses.

When using historical chat records for responses, the prompt given to the large model is constructed as follows:

```
final_questions = "This is the information you already know:" + pre_conversation + "This is what you can see:" + picture_questions + "This is our recent conversation:" + history + "Note:" + inputQuestions
```

Where:

- `pre_conversation`: Information from long-term memory.
- `history`: Information from the recent conversation.
- `picture_questions`: Visual information.
- `inputQuestions`: The current interaction information from the user.

Organizing the prompt in this way can effectively prevent the large model from becoming confused by receiving too much information.

When using web search for responses, the prompt given to the large model is constructed as follows:

```
final_questions = pre_conversation + "Summarize the information from the above web pages, using the form 'These contents...'"
```

This approach allows the large model to organize the vast amount of information retrieved from the web, highlight the key points, and communicate these key points with the user.

G. Vector Database

Vector databases enable quick and efficient similarity searches by converting text, images, or other data into vectors. This method is particularly useful for applications that require rapid retrieval of related information. The specific implementation steps are as follows: First, initialize the DashVector client and load data from the specified JSON file path. Then, process the data in batches to efficiently handle each batch. Use a text embedding model to generate embedding vectors for each batch of text data, and handle any potential exceptions appropriately. Next, ensure that the collection in the vector database can be successfully accessed; if the retrieval fails, throw an exception and terminate the program. For each batch of data, generate text embeddings and batch-insert them into the collection along with their corresponding document IDs and field information. During the insertion process, be sure to capture and address any potential exceptions. After each batch is processed, update and save the progress.

H. Web Spider

In this web scraping functionality, we first simulate a real user's browser request by defining request headers that include a browser user agent. Then, we construct the URL for the search query and send an HTTP GET request to the 360 search engine using requests.get. The HTML response received is parsed using the etree module from the lxml library. Through XPath, we extract the title, descriptive summary, and link of each search result from the HTML. These data are organized into dictionaries and compiled into a list. Finally, the search results are printed to the console and saved as a JSON file for subsequent use and analysis. We will pass this list to the Qwen Turbo large model, which will summarize and generalize based on our questions and produce chat outputs.

I. Flask

Flask is a lightweight Python web application framework based on the Werkzeug WSGI toolkit and Jinja2 template engine. It features a simple and flexible design, allowing developers to freely choose the components they use without being forced to use specific tools or libraries. In this project, Flask is used to facilitate the sending of information between the server and local systems.

The project defines three routes: upload, upload_photo, and process_inf. The upload route is the core route, where the basic logic is completed. This involves uploading a .pcm file to the server, recognizing it as text, and then depending on the content, interacting with historical memory information or performing a web search. The response is then converted into speech and sent back locally.

The upload_photo route acts as a supplementary route, running in parallel with the upload route. This route handles the uploading of visual information (.jpg files) to the server. Once uploaded, the server's deployed MobileVLM model recognizes the content, records the recognition information on the server, and sends a success message back to the local system.

The process_inf route is used to handle the processing of chat results. The local system sends a request to the server, which then ends the conversation, hands all chat records over to a large model for summarization and storage in a vector library. Upon successful completion, a success message is sent to the local system, and the conversation is terminated.

IV. Reduce latency

In actual user scenarios, one of the most crucial tasks is to minimize the latency of the chatbot, ensuring that it responds to user queries as quickly as possible to maintain a good user experience. We observed that except for the Text-to-Speech (TTS) part, every step can generally be completed within one second. However, the TTS module exhibits significant latency. Therefore, we focused on the TTS module to reduce latency.

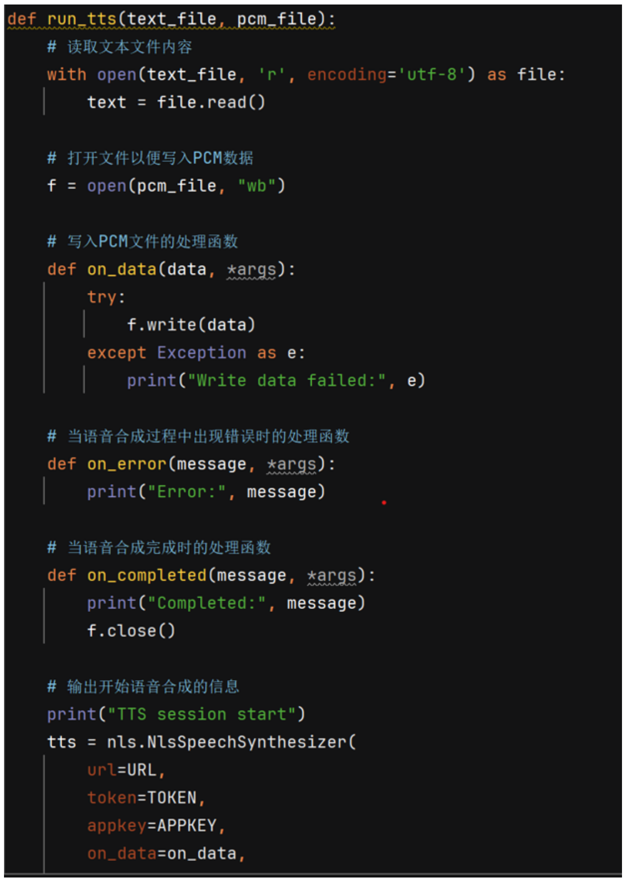
The code in Figure 5 implements Text-to-Speech (TTS) synthesis and handles the writing of PCM data, error processing, and completion notifications through callback functions. In the original code provided by the official sources, a multi-threaded implementation is used. However, in practical applications, it was found that multi-threading introduces substantial latency. The reasons include the CPU time required for context switching between threads, which involves saving and restoring thread states and switching memory pages, potentially leading to increased overall latency. Furthermore, when multiple threads access shared resources (such as files or memory), resource contention can occur. This contention forces threads to wait for others to release resources, thus increasing latency. To ensure thread safety, synchronization mechanisms such as locks are necessary. These mechanisms not only complicate the code but also introduce additional overhead. Experiments showed that multi-threading led to multiple recognitions of a single sentence and merging of results for optimal output. However, it was discovered that a single recognition is sufficient for high accuracy, eliminating the need for multiple recognitions. Thus, the project was improved by switching from multi-threading to single-threading, where the audio input is recognized only once, effectively reducing latency. In single-thread mode, reading text files and writing PCM files can be sequentially performed without interference or competition from other threads. In multi-thread mode, callback functions like on_data, on_error, and on_completed might be called in different threads, causing unnecessary synchronization overhead. In single-thread mode, these callbacks are called sequentially along the same execution path, reducing latency.

After implementing these changes, it was found that the revised code significantly reduced latency, decreasing it from over ten seconds to under five seconds, demonstrating the effectiveness of this project in reducing latency.

V. Limitations and Discussion

Although the project has achieved its intended functionalities and added some unique features, there are still many areas that need improvement.

Firstly, further latency reduction is needed. Although using a single-threaded approach greatly improved the recognition



```python
def run_tts(text_file, pcm_file):
    # 读取文本文件内容
    with open(text_file, 'r', encoding='utf-8') as file:
        text = file.read()

    # 打开文件以便写入PCM数据
    f = open(pcm_file, "wb")

    # 写入PCM文件的处理函数
    def on_data(data, *args):
        try:
            f.write(data)
        except Exception as e:
            print("Write data failed:", e)

    # 当语音合成过程中出现错误时的处理函数
    def on_error(message, *args):
        print("Error:", message)

    # 当语音合成完成时的处理函数
    def on_completed(message, *args):
        print("Completed:", message)
        f.close()

    # 输出开始语音合成的信息
    print("TTS session start")
    tts = nls.NlsSpeechSynthesizer(
        url=URL,
        token=TOKEN,
        appkey=APPKEY,
        on_data=on_data,
```

Fig. 5. Single-threaded

speed, the process requires the complete conversion of a .txt file to a .pcm file before playback, causing significant latency. An engineering approach that sends each character generated by the large model to local playback immediately could greatly reduce this latency. However, this method also has many problems. For example, the semantics in daily conversations largely depend on tonal semantics. Transmitting and playing single characters might lose semantic meaning, making the played-back tone incomprehensible.

Secondly, the performance of the VLM (Visual Language Model) needs to be improved. Due to hardware limitations, it is not feasible to deploy large VLM models on servers, leading to errors in image recognition or biased understanding. Also, the time taken for model inference has increased, and it's not possible to quickly and efficiently infer each image, only solvable by increasing sampling time.

Furthermore, in interactions with large models, there is a desire to provide more information to the models to improve their ability to communicate with users using multiple information sources. However, this also confuses the large models, making them unable to accurately respond to queries. A solution to this problem is to provide more precise prompts to the large models, helping them clearly understand the role of each part of the information. Yet, determining the optimal prompt still requires careful consideration.

Lastly, the functionality of web searches relies on web crawlers. However, most websites have crawler detection and anti-crawling mechanisms, which pose certain difficulties and restrictions on web searches. In practice, obtaining permissions from some search engines can help ensure the stability of the chatbot.

VI. Conclusion

This project showcases the potential and impact of interactive chatbots, integrating cutting-edge technology with the practical needs of society for companionship and information acquisition. By incorporating voice chat, visual capture, and sophisticated memory management technology, this project has successfully developed a chatbot that not only understands and responds effectively to user inputs but also personalizes interactions by remembering past conversations.

Utilizing cloud-based technology and optimizing our system to minimize latency, the chatbot provides an efficient interactive experience, mimicking a human conversation partner. Although there are still challenges in further reducing latency, enhancing model performance, and dealing with limitations in web searching, the foundation laid by this project provides a solid base for future enhancements.

The application of vector databases for memory storage, the implementation of single-threading to reduce latency, and strategic use of web crawlers to gather information demonstrate exploratory approaches to solving complex technical issues. Future developments in these technologies will enhance the system's capabilities, offering more refined interactions and greater accessibility.

Designing this chatbot has provided new insights into my studies in human-computer interaction, recognizing the immense potential of AI technologies like LLM in enhancing human-machine interactions. I will continue to focus on such tasks in my future studies, contemplating how to design a chatbot that closely meets practical needs.

References

[1] Chu, X., Qiao, L., Zhang, X., Xu, S., Wei, F., Yang, Y., Sun, X., Hu, Y., Lin, X., Zhang, B., & Shen, C. (2024). *MobileVLM V2: Faster and Stronger Baseline for Vision Language Model*. arXiv. https://doi.org/10.48550/arXiv.2402.03766