ELEC4848 Senior Design Project 2023-2024

JIANG Feiyu (3035770800)

Development of a Full-stack Aphasia AI Web App based on REACT-FASTAPI

Supervisor: Dr. A.T.L. Lee
Second Examiner: Dr. G. Pang

Abstract

As there are currently no technological solutions for aiding aphasiac communicate effectively, the increasing prevalence of aphasia has left many individuals struggling in daily activities. With the increasingly maturing AI technology, AI-driven personalized treatment approaches and assistive systems are considered among the viable solutions for enhancing the quality of life for patients in the current landscape. Therefore, there is need for developing a full-stack web application combining AI language models to address this pressing need. In this project, a model of utilizing the FASTAPI, AI and React full-stack model was investigated, and a web application was developed based on Python and JavaScript. By interacting with the database, the web application can provide all the fundamental functionalities including login, register and chat box function. With this technical stack, the application is accessible from any internet-connected device and seamlessly integrates user-trained AI models via API, providing rapid responses to speech inputs. The application showcases promising recognition accuracy for the 46 most commonly used Cantonese Chinese phrases, achieving a commendable 97.9% accuracy rate. Moreover, the average response time from the API remains controlled at less than 4 second, provided there is smooth network transmission, significantly enhancing convenience for users with aphasia. While the overall performance of the application is deemed acceptable, certain inefficiencies in network transmission and privacy concerns have been identified. However, it is anticipated that these challenges can be addressed through measures such as dissecting audio data and implementing robust encryption algorithms.

Acknowledgement

I would like to express my sincere gratitude to several individuals and organizations who have provided invaluable support throughout my undergraduate studies.

First and foremost, I am deeply thankful to my advisors, Professor Lee and Professor Pang. Their enthusiasm, patience, insightful comments, valuable information, practical advice, and continuous brainstorming sessions have been a constant source of guidance and inspiration. Their unwavering support and willingness to share their knowledge and expertise in the fields of artificial intelligence and software development have been instrumental in the success of my research and the completion of this paper. I am truly grateful for their mentorship, which has shaped my academic journey in profound ways.

I would also like to extend my heartfelt appreciation to the Engineering Faculty of The University of Hong Kong for accepting my project. The opportunity to conduct research in such a prestigious institution has been an honour and a privilege. The resources, facilities, and academic environment provided by the university have greatly contributed to my learning experience and overall growth as a student.

In addition, I am immensely grateful to the EEE department for their financial support. Their generous funding has allowed me to pursue my research goals and explore new avenues of knowledge in my chosen field. Their investment in my education has been invaluable and has motivated me to strive for excellence in my academic endeavours.

Furthermore, I would like to express my thanks to all the professors, lecturers, and teaching assistants who have imparted their knowledge and expertise in various courses throughout my undergraduate studies. Their dedication to teaching, their passion for their respective fields, and their commitment to nurturing the intellectual growth of their students have left a lasting impact on me. I am grateful for their guidance, encouragement, and the valuable insights they have shared during my academic journey.

Last but not least, I would like to acknowledge the support and encouragement of my family and friends. Their unwavering belief in my abilities, their constant encouragement, and their understanding during challenging times have been a source of strength and motivation. I am truly fortunate to have such a strong support system, and I am grateful for their love, encouragement, and unwavering support throughout my academic journey.

Once again, I would like to express my deepest appreciation to all the individuals and organizations who have played a part in my academic journey. Your support, guidance, and encouragement have been invaluable, and I am truly grateful for the opportunities and experiences that I have been fortunate to have. Thank you all for being a part of my undergraduate studies and for contributing to my growth and development as a researcher and as an individual.

Table of Content

List of tables

List of Figures

Abbreviations

A.A.C. = Augmentative and Alternative Communication

AI = Artificial intelligence

AWS = Amazon Web Services

CORS = Cross-Origin Resource Sharing

DNS = Domain Name System

EC2 = Elastic Compute Cloud

SPA = Single Page Application

SSH = Secure Socket Shell

SSL = Secure Sockets Layer

1. Intoduction

Aphasia, a language disorder primarily resulting from brain injury or stroke, imposes considerable impediments on individuals' everyday communication capabilities. The absence of tailored technological solutions to aid those with aphasia exacerbates their struggles in effectively expressing themselves, emphasizing the pressing need for innovative approaches to address this issue within the academic and clinical domains.

Until today, in the context of obstacle-oriented therapy, computer programs have been applied in aphasia rehabilitation with positive outcomes. However, limitations in reading and writing hinder their access to the internet, and computer applications aimed at functional and social participation are still imperfect [1]. Reference [2] emphasized the need for significant advancements in AI before its widespread application in aphasia rehabilitation. Nevertheless, there's clear potential for AI to be foundational in innovative A.A.C. devices or applications, promising to revolutionize aphasia assessment and therapy. Recent research [3] has primarily focused on AI-driven automated assessment and therapy in aphasia, yet there's a growing interest in utilizing AI for personalized assistive systems. However, there's a noticeable gap in developing applications tailored to improving the daily lives of aphasia patients, highlighting the urgency to prioritize solutions for enhancing their quality of life. Achieving such advancements requires enhanced collaboration between aphasia rehabilitation and AI development fields to bridge existing gaps.

The primary objective of this project is to address the substantial challenges faced by individuals with aphasia in their daily communication endeavours. By harnessing advanced technologies like AI, our aim is to empower individuals with aphasia to communicate more effectively and regain autonomy in their lives. This project seeks to develop a comprehensive web application integrating AI language models to facilitate communication for individuals contending with aphasia. The scope encompasses design, development, and deployment, with a specific emphasis on user authentication, registration, and chat functionality. Leveraging the FASTAPI, AI, and React architecture, a user-friendly platform that enables seamless interaction and expression is intended to be created. With the capability for accessibility from any internet-connected device, users will possess the flexibility to communicate seamlessly from diverse geographical locations. Moreover, the integration of user- trained AI models will ensure swift and personalized responses to speech inputs. This endeavour signifies advancement in empowering individuals with aphasia to communicate independently and effectively in their day-to-day lives.

The paper is structured as follow. Section 2 explains the User Requirements, which is utilized to define the development goals of different parts. Section 3 describes the main technical methods and structure of the project. Section 4 descries the optimization methods and section 5 describes the deployment details. Finally, section 6 presents the concluding remarks.

2. Requirements Analysis

Clear user requirements involve considering both functional and non-functional requirements. Functional requirements focus on the system's behaviour and features, such as user authentication, chat functionality, and integration of AI language models. Non-functional requirements, on the other hand, concentrate on the system's performance, security, usability, and documentation to ensure efficiency, security, user-friendliness, and maintainability.

2.1 Functional Requirements

(1) User Authentication and Registration: Given that aphasia patients may present with cognitive and reading impairments, this registration process should minimize unnecessary verifications (e.g., email addresses, CAPTCHA) to reduce operational steps. Simultaneously, options such as voice playback should be provided when necessary.

(2) Chat Functionality: The core functionality of the application is a chat interface that allows users to input voice and receive responses. Users can complete voice input by local files, drag and drop, and online audio input. Upon receiving user voice, the backend AI will automatically help parse it into Chinese characters and output responses in the chat window. To enhance user experience, the chat interface should be concise and intuitive, eliminating all unnecessary activities (e.g., creating group chats, sending images) apart from inputting voice and displaying text. The page should clearly demonstrate how to transmit voice, how to record, and provide clear indications of the voice processing status and text parsing.

(3) Personalized Integration of AI Language Models: The application should integrate advanced AI language models capable of understanding and generating responses. Based on the frequency information extracted from the Hong Kong Mid-20th Century Cantonese Corpus [4], 46 commonly used Cantonese words closely related to daily life were selected (Figure 1). These models should be trained to recognize and adapt to the 46 commonly used Cantonese characters in the lives of aphasia patients.

46 keywords

0	1	2	3	4	5	6	7	8	9
我	要	去	廁	所	返	睡	房	書	廚

A	B	C	D	E	F	G	H	I	J
刷	牙	洗	面	開	電	腦	門	燈	出

K	L	M	N	O	P	Q	R	S	T
客	廳	睇	視	叫	人	鐘	想	上	床

U	V	W	X	Y	Z
落	攞	手	機	畀	話

!	,	+	(	)	$	%	#	@	T
你	飲	茶	唔	水	吃	麵	飯	早	餐

Figure 1. 46 commonly used Chinese characters.

(4) Accessibility and Device Compatibility: The web application should adhere to accessibility standards to ensure that individuals with different abilities can easily navigate and use the platform. This includes support for screen readers, keyboard navigation, and other input methods. This encompasses browser compatibility and screen adaptability development.

(5) Data storage: The audio uploaded by the user will be stored in the database according to the tag for subsequent use.

2.2 Non-Functional Requirements

(1) Performance: The application should prioritize high responsiveness, aiming for minimal latency between user input and system response. Specifically, it should strive to maintain a time lag of within three seconds from the user inputting audio to the display of text parsing on the screen. Achieving this requires efficient processing of AI models and optimization of both server-side and client-side code.

(2) Security: Robust security measures are essential to safeguard user data and privacy. This entails implementing permissions for recording from the front-end webcam, encrypting user audio for storage, and ensuring the integrity of user passwords.

(3) Usability: The user interface should be intuitive and easy to navigate, even for individuals with limited technical expertise. This includes clear labels, contextual help options, and consistent design patterns throughout the application. Design considerations should also be made to be friendly to users with aphasia and disabilities.

(4) Documentation: Comprehensive documentation should be provided for both users and developers, including user guides, API documentation, and system architecture diagrams. This ensures that stakeholders have the necessary information to understand and use the application effectively.

3. Software design and implementation

This section outlines the software design and implementation process, focusing on minimalist UI design, system architecture, and coding details. It encompasses frontend-backend interaction, database management, and AI model integration for efficient user interaction and data processing.

3.1 Software Product Design

Due to the emphasis on catering to aphasia patients, the product adheres to a relatively minimalist design philosophy. In terms of user interface design (Figure 2), efforts are made to minimize navigation jumps, beyond necessary logins, to facilitate user interactions. Consequently, this design does not separate the user account interface; instead, it integrates account operations with recognition operations, thereby minimizing the frequency of jumps and navigations. To enhance readability, large and legible fonts with ample spacing are employed. The stark black-and-white contrast aids users in distinguishing actionable areas, while clearly delineated functional zones (User Centre and Dialogue) reduce potential errors. Additionally, consistency in layout, colour, and navigation is maintained throughout the entire application to assist aphasia users in memory and operation. Furthermore, multimodal interactions such as online recording and audio uploading cater to diverse user preferences.

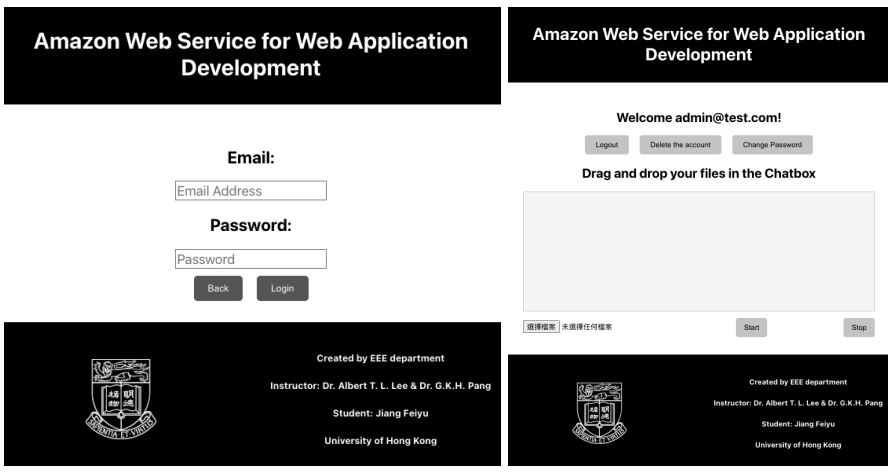

Figure 2a. UI design of login page and user center.

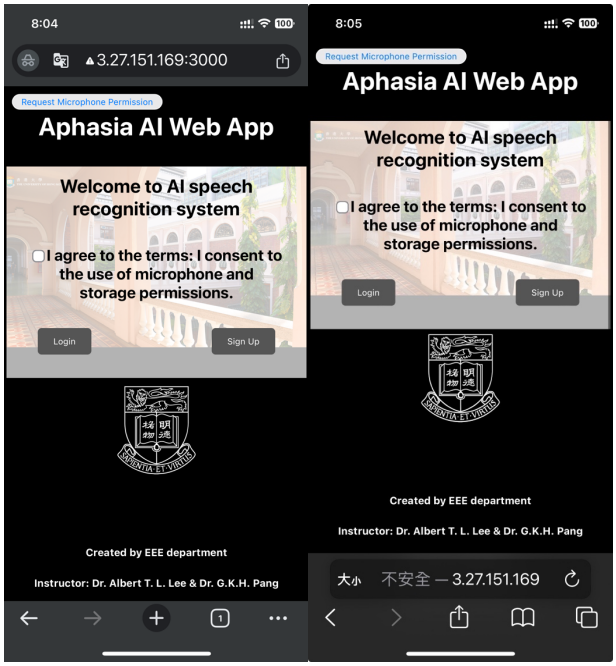

Figure 2b. Chrome and Safari mobile browser interfaces.

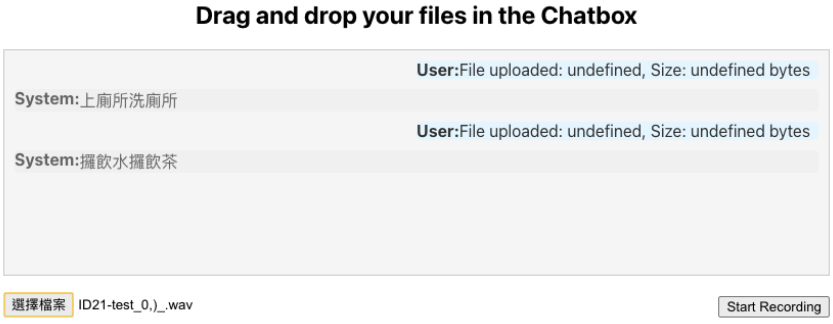

Figure 2c. Chat box interfaces after uploading audio.

3.2 System Architecture Design

Choosing the React+FastAPI technology stack for developing an AI-powered online chat application offers several advantages. React, a popular JavaScript library for building user interfaces, and FastAPI, a modern, fast web framework for building APIs with Python, complement each other to create a robust and efficient application. Here's a detailed description of why this technology stack is a suitable choice for developing such an application.

(1) **Rich and interactive user interface**: React's component-based architecture enables the creation of dynamic and responsive user interfaces. It provides a vast ecosystem of reusable components and libraries, making it easier to create a visually appealing

and interactive chat application. Users can have a seamless experience with real-time updates and smooth transitions.

(2) **Efficient backend development**: FastAPI, built on top of the high-performance Starlette framework, offers excellent performance and scalability. It leverages the asynchronous capabilities of Python to handle multiple concurrent requests efficiently. FastAPI's intuitive syntax and automatic API documentation generation make backend development faster and more maintainable.

(3) **Real-time communication**: React's integration with WebSocket libraries, such as Socket. io or WebSocket API, allows real-time bidirectional communication between the client and the server. This feature is crucial for a chat application, enabling instant message delivery and updates. FastAPI can handle WebSocket connections using libraries like fastapi-websocket, ensuring seamless integration with React.

(4) **AI integration**: React+FastAPI provides a solid foundation for integrating AI capabilities into the chat application. The backend powered by FastAPI can handle AI-related tasks, such as natural language processing, sentiment analysis, and chatbot interactions. Python's extensive libraries, like TensorFlow or PyTorch, can be utilized for machine learning and AI algorithms.

(5) **Scalability and performance**: React's virtual DOM and FastAPI's asynchronous nature contribute to a high-performing application. React efficiently updates only the necessary components, reducing unnecessary re-rendering. FastAPI's asynchronous design allows handling multiple simultaneous requests without blocking the server, resulting in improved scalability and responsiveness.

(6) **Community and support**: Both React and FastAPI have large and active communities, providing ample resources, tutorials, and support. Developers can find numerous online forums, documentation, and open-source projects related to these technologies. This ensures that any challenges faced during development can be easily addressed with the help of the community.

(7) **Code reusability**: React's component-based architecture and FastAPI's modular design promote code reusability. Components in React can be reused across different parts of the application, reducing redundancy and improving development efficiency. FastAPI's modular approach allows the creation of reusable API endpoints, making it easier to scale and extend the application in the future.

Taking into consideration the fundamental data flow, a technical architecture comprising frontend, backend, database, and AI model components is designed (Figure 3). As illustrated in the diagram, data such as user information and voice segments are transmitted between the frontend and backend via requests such as POST and FETCH. Meanwhile, data between the

backend, database, and speech model is handled through function calls and parameter passing for processing.

The React frontend is structured using a component-based approach, which enhances modularity and facilitates code reuse. Within the React application, distinct components are tasked with specific request functions, facilitating interaction with the backend. The FASTAPI backend, implemented in Python, defines routes that align with the anticipated functionalities of React components. Upon receiving requests from the React frontend, the backend efficiently processes data using FASTAPI's capabilities. Depending on the requested functionality, the backend conducts essential operations such as data processing, validation, or retrieval from the database. Furthermore, the backend establishes interfaces for invoking AI models, facilitating data transmission and model selection. Following the completion of AI model processing, the backend furnishes results to the frontend page, thereby reflecting changes in the user interface.

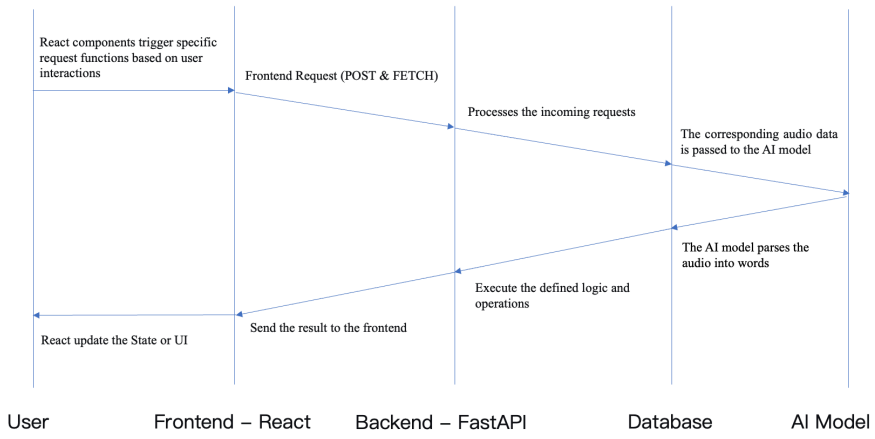

Figure 3. Data Flow.

3.3 Coding and Implementation

This section provides an explanation of the technical details of the frontend and backend implementation (Figure 4). As depicted in the figure, the direction of the arrows represents the sequence of request transmission and reception, while double arrows indicate bidirectional information exchange.

The frontend consists of several components with different functionalities. Each component corresponds to a request that interacts with the backend routes of FASTAPI. The frontend components can display changes in the interface. Among them, the Dialogue component is primarily responsible for audio uploading. It communicates with the backend routes /upload

and /upload-up to transmit audio and parsing information.

In the /upload and /upload-up routes, the program simultaneously saves the audio and reads the files in the database for preprocessing by the Interpret function. Subsequently, the Voice Detection Engine is invoked. The Voice Detection Engine serves as an engine to connect to the model interface, converting the audio files that need to be processed into binary files readable by AI models and returning the processing results. These results are then converted into understandable Chinese characters by the Interpret function and returned through the routes.

On the other hand, the frontend can perform account-related operations through routes such as /register, /change-password, /delete, etc. These operations can be directly performed on the database by the backend.

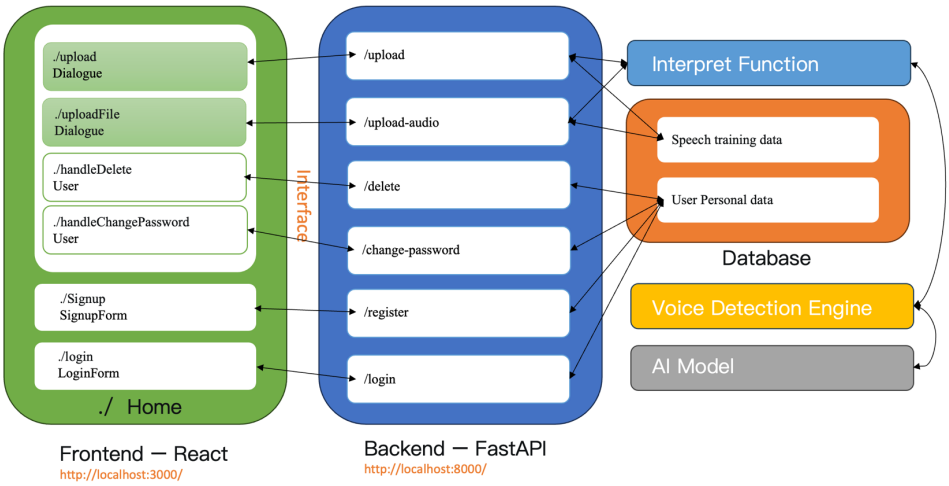

Figure 4. Project Framework Design.

3.3.1 Frontend Implementation

Figure 5 shows the implementation logic of frontend. When users log in for the first time, the system prompts them to choose between registration and login options. If users choose to register, they need to input their email (username) and password. Frontend performs regular expression checks to validate email addresses and ensure that passwords meet specified criteria for strength. For user convenience, an option for generating randomly strong passwords is provided. If the username already exists in the database, the system will alert the user.

Upon successful registration or login, users are directly taken to the user interface. In the user

interface, users can choose to log out or delete their accounts.

The frontend offers various methods for interacting with AI: Users can directly record audio on the webpage with corresponding permissions. Users can also drag and drop audio files into a dialog box or choose files to upload. The uploaded files are transferred to the backend, saved as WAV files, and users receive system prompts to confirm the upload. The frontend can display the parsed text on the interface.

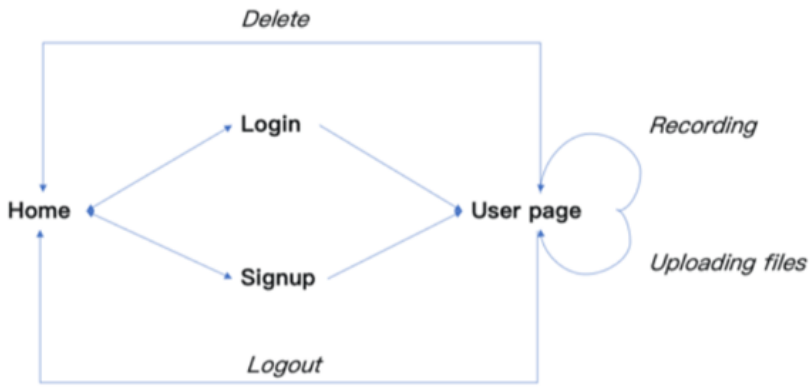

Figure 5. Implementation Logic of Frontend

The code utilizes React's functional components to create reusable UI elements. The chat box, login page, and navigator are implemented as functional components, highlighting the modular and efficient nature of React development.

State management is a crucial feature demonstrated in the code. The *useState* hook is used to manage component state within the functional components. For example, the Dialogue component maintains the dialogue state, which stores an array of messages exchanged in the chat. Similarly, the *AudioUpload* component manages the *selectedFile* state, representing the currently chosen audio file. This approach simplifies state management and enables dynamic updates within the components.

The frontend communicates with the backend server using the fetch API. This is exemplified in the *uploadFile* function within the Dialogue component and the *handleUpload* function in the *AudioUpload* component. These functions make POST requests to the backend server, sending the file data as *FormData*. The response is then handled accordingly, updating the dialogue or displaying appropriate alert messages.

Component interaction is also a notable feature in the codebase. For instance, the Dialogue component interacts with the Record component by passing the *updateDialogue* function and the username prop. This establishes seamless communication between components, allowing the Record component to update the dialogue within the Dialogue component.

Error handling is integrated throughout the code to provide a smoother user experience. The code includes error handling for file size limits and failed network requests. When the file size exceeds the limit, appropriate alerts or console logs are displayed. Additionally, error messages are logged for file upload or backend communication issues.

Below are some key technologies used in React:

- **Media Recording Function**: The *navigator.mediaDevices.getUserMedia* API is used to capture audio directly from the user's device, which utilizes the *MediaRecorder* API to manage audio recording, encompassing functionalities such as data availability management and cessation of recording events (Figure 6). This recording paradigm exhibits notable traits of being highly lightweight and user-friendly.
- **Audio Upload Function**: This code snippet (Figure 7) demonstrates the process of uploading recorded audio. It creates a *FormData* object to append the audio *blob* with a unique filename. The data is then sent to a server endpoint using the fetch API, enabling asynchronous file uploads and handling server responses for display in the application.
- **Microphone Permission Check Function**: The code (Figure 8) defines an async function, *checkMicrophonePermission*, using *navigator.permissions.query* to check microphone permission. Once checked, the result is stored in the result variable. State is updated using *setMicrophonePermission*. Additionally, a listener updates state upon permission changes. The empty array in *useEffect* ensures the code runs once on component mount.
- **Dynamic Render Function**: The dialogue function dynamically renders dialogue entries based on the contents of the dialogue state array (Figure 9). It includes conditional rendering for download buttons when a file URL is present in the entry.
- **Routing**: Routing (Figure 10) enables Single Page Applications (SPAs), where page sections update for a faster user experience compared to multi-page apps, permitting parameter exchange between routes and enabling dynamic content display and page interaction.

```
const startRecording = () => {
    navigator.mediaDevices.getUserMedia({ audio: true, video: false })
        .then(stream => {
            mediaRecorderRef.current = new MediaRecorder(stream);
            mediaRecorderRef.current.ondataavailable = (e) => {
                chunksRef.current.push(e.data);
            };
            chunksRef.current = [];
            mediaRecorderRef.current.start();
            setIsRecording(true);
            updateDialogue("Online Audio", "User"); // 添加 "User: Online Audio" 消息
        })
        .catch(err => console.error('Error accessing microphone:', err));
};


const stopRecording = () => {
    if (mediaRecorderRef.current && mediaRecorderRef.current.state === 'recording') {
        mediaRecorderRef.current.stop();
        setIsRecording(false);
        mediaRecorderRef.current.addEventListener('stop', () => {
            const audioBlob = new Blob(chunksRef.current, { type: 'audio/wav' });
            uploadAudio(audioBlob);
        });
    }
};
```

Figure 6. Media Recording Code.

```
const uploadAudio = async (audioBlob) => {
    const formData = new FormData();
    formData.append('audio', audioBlob, `${username}_recording.wav`);

    try {
        const response = await fetch('http://3.27.151.169:8000/record-audio', {
            method: 'POST',
            body: formData
        });

        if (!response.ok) {
            console.error('File upload failed');
            return;
        }

        const data = await response.json();
        updateDialogue(`File uploaded: ${data.filename}, Size: ${data.file_size} bytes`, "System");
        updateDialogue(data.answer, "System"); // 使用后端返回的消息
    } catch (error) {
        console.error('Error uploading audio:', error);
    }
};
```

Figure 7. Audio Upload Code.

```
useEffect(() => {
  const checkMicrophonePermission = async () => {
    try {
      const result = await navigator.permissions.query({ name: 'microphone' });
      setMicrophonePermission(result.state);
      result.onchange = () => {
        setMicrophonePermission(result.state);
      };
    } catch (error) {
      console.error('Error checking microphone permission:', error);
    }
  };

  checkMicrophonePermission();
}, []);
```

Figure 8. Microphone Permission Check Code.

```
{dialogue.map((entry, index) => (
  <div key={index} className={`dialogue-entry ${entry.user.toLowerCase()}`}>
    <strong>{entry.user}:</strong> {entry.message}
    {entry.fileUrl && (
      <button onClick={() => handleDownload(entry.fileUrl, entry.message)}>Download</button>
    )}
  </div>
))}
```

Figure 9. Dynamic Dialogue Rendering Code.

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/login" element={<LoginForm />} />
  <Route path="/signup" element={<SignupForm />} />
  <Route path="/user" element={<User />} />
</Routes>
```

Figure 10. Routing Code.

3.3.2 Backend Implementation

FastAPI is a modern Python framework used to build high-performance web applications and APIs. It comes with a range of powerful features that enable developers to quickly create reliable, scalable, and maintainable applications. The Backend FASTAPI program features various functionalities for audio processing and user management. It includes routes for audio recording, uploading, and translation, with file type detection and conversion capabilities. Additionally, it implements user registration, login, password change, and deletion functionalities with JSON data storage. Logging and exception handling ensure robustness and security. Below are some key technologies used in FASTAPI:

Firstly, FastAPI supports Cross-Origin Resource Sharing (CORS), which allows the application to accept cross-origin requests from different sources. This is useful for interacting with other domains or for debugging purposes in the development environment. Secondly, FastAPI provides the Trusted Host middleware, which handles the host header and prevents host header attacks. This helps ensure that the application communicates with the correct host and enhances security. Additionally, FastAPI offers flexible logging configuration. By using Python's logging module, log records could be viewed, which may use to track the application's runtime behavior and debug errors, including request methods, URLs, request headers, response status codes, and response headers.

In routes such as user registration, user login, and user deletion, the project uses the json module to read from and write to a JSON file for storing user data persistently. The recorded audio files are converted from the webm format to the wav format by invoking the FFmpeg command-line tool. This ensures proper audio processing and analysis in subsequent steps.

FastAPI simplifies data validation and automatic conversion of request data by using Pydantic models. This helps reduce data processing errors and improves the reliability of the application. FastAPI also provides functionality for handling file uploads and processing. Furthermore, FastAPI offers interfaces for integrating AI models.

- **Audio Conversion**: In this code snippet (Figure 11), the primary technology used is audio conversion. Specifically, it utilizes FFmpeg, a powerful multimedia framework, to extract audio from the uploaded webm file and convert it to the WAV format. This process involves running FFmpeg commands via subprocess to manipulate the audio data, ensuring compatibility with further processing steps. This capability enables seamless integration of different audio formats within the application, enhancing its versatility and functionality.
- **Password Changing**: This code snippet (Figure 12) implements a route for changing user passwords. It utilizes Python's FastAPI framework to define an HTTP POST endpoint ("/change-password"). Upon receiving a request with user credentials, it checks if the user exists in the user database. If the user is found, their password is updated with the new password provided in the request. The updated user data is then written back to a JSON file for persistence. Finally, a success message is returned to the client indicating that the password change was successful.

```
# Save the audio file
with open(file_path, "wb") as f:
    f.write(await audio.read())

# Determine file type
file_type = detect_file_type(file_path)
print(f"File type: {file_type}")

if file_type == "video/webm":
    print("Converting ...")
    # Convert webm to wav
    print(wav_file_path)
    try:
        # 使用 FFmpeg 提取音频并将其转换为 WAV 格式
        subprocess. run(['ffmpeg', '-i', file_path, '-vn', '-acodec', 'pcm_s16le', '-ar', '44100', '-ac', '2', wav_file_path])
        print("convert sucess")

    except ffmpeg.Error as e:
        print(f"ffmpeg error: {e.stderr}")
        return {"error": f"ffmpeg error: {e.stderr}"}

AIanswer = InterpretAI(file_counter)
print('APIHost AIanswer (interpret) = ',AIanswer)

if os.path.exists(wav_file_path):  # 检查文件是否存在
    os.remove(wav_file_path)  # 删除文件
    print(f"文件 {wav_file_path} 已成功删除")
else:
    print(f"文件 {wav_file_path} 不存在")

return {"answer": AIanswer}
```

Figure 11. Audio Conversion Code.

```python
@app.post("/change-password")
def change_password(user: User):
    existing_user = next((u for u in users_db if u["usn"] == user.usn), None)
    if not existing_user:
        raise HTTPException(status_code=404, detail="User not found")

    # 更新用户密码
    existing_user["pwd"] = user.pwd

    # 将更新后的用户数据写入 JSON 文件
    with open("users.json", "w") as file:
        json.dump(users_db, file, indent=2)

    return {"message": "Password changed successfully"}
```

Figure 12. Password Changing Code.

- **File Uploading**: In this code (Figure 13), the @app.post("/upload") decorator defines a POST endpoint at the URL path "/upload" that allows clients to upload files to the server. The file parameter of type *UploadFile* is used to receive the uploaded file data. This parameter is automatically handled by FastAPI, enabling easy access to the content, metadata, and other details of the uploaded file. The code uses the *save_path* variable to specify the path where the uploaded file will be saved. If the path doesn't exist, it creates the directory to ensure the target directory is available for file storage. The code determines the next available file number by calling the *find_next_available_number* function. It generates sequential file names using a file counter and the ".wav" extension. This facilitates organized storage of audio files and also serves as a reference credential for AI processing. If the uploaded file is not in WAV format, the code checks it using the *is_wav* function. If necessary, it converts the file to WAV format using the *AudioSegment* class from the *pydub* library. This ensures consistency in processing and compatibility with subsequent audio operations. After the file is processed, the code returns a JSON response containing detailed information about the uploaded file, such as the file name, file size, and the result of AI interpretation.
- **AI Audio Parsing**: After importing the Voice Detection Engine, the relevant code of the AI model is integrated into the FastAPI application. This allows us to utilize the functionality and methods provided by the AI model. The *InterpretAI* function acts as an interface for the AI model, specifically designed to handle audio processing. It receives an *audio_id* parameter as input, which serves as a unique identifier for the audio to be processed. The InterpretAI function invokes the *run_interpret_audio* method. *run_interpret_audio* takes on the role of the Voice Detect Engine interface, feeding the user ID into the main function, playing the role of encapsulation. For the function of Voice Detection Engine method will be further discussed in section 3.3.3, takes the *audio_id* as a parameter and is responsible for triggering the

execute_interpret function in the Voice Detection Engine for parsing. The *execute_interpret* function utilizes the AI model to perform the audio parsing operation, yielding the parsing result as output. By organizing the code in this way, a clear separation of concerns is established, allowing the *InterpretAI* function to serve as an intermediary between the FastAPI application and the Voice Detection Engine. This encapsulation enables efficient handling of audio-related tasks and ensures the appropriate utilization of the AI model's capabilities. Based on the parsing result returned by the AI model, the *InterpretAI* function performs different actions. If parsing fails, it returns "Parsing failed." If the upload is unsuccessful, it returns "Upload failed." Otherwise, it proceeds with character conversion and processing of the parsing result. In the *InterpretAI* function, each character in the parsing result is iterated over, and character conversion and processing are applied based on a predefined dictionary (Figure 1). Finally, the *InterpretAI* function returns the processed parsing result, which can be utilized by the FastAPI application.

```python
@app.post("/upload")
async def upload_file(file: UploadFile = File(...)):
    try:
        # Specify the path where you want to save the uploaded files
        save_path = "./Data"

        # Create the path if it doesn't exist
        os.makedirs(save_path, exist_ok=True)

        # Find the next available file number
        file_counter = find_next_available_number(save_path)

        # Generate the file name with sequential numbering
        file_name = f"{file_counter}.wav"
        file_path = os.path.join(save_path, file_name)

        # Save the file to the specified path
        with open(file_path, "wb") as f:
            shutil.copyfileobj(file.file, f)

        # Convert the file to WAV format
        if is_wav(file_path) == False:
            audio = AudioSegment.from_file(file_path, format=file.filename.split('.')[-1])
            audio.export(file_path, format="wav")

        print("file_counter", file_counter)

        # Process the uploaded file
        # For now, we will just return the file details
        AIanswer = InterpretAI(file_counter)
        print('APIHost AIanswer (interpret) = ',AIanswer)

        return {"filename": file_name, "file_size": os.path.getsize(file_path),"answer": AIanswer}
```

Figure 13. File Uploading.

```python
# 翻译音频
def InterpretAI(audio_id: int):
    print("audio_id", audio_id)
    AIanswer = run_interpret_audio(audio_id)
    if AIanswer == "...":
        return "Parsing failed"
    elif AIanswer == None:
        return "Upload failed"
    else:
        decoded_string = ''
        for char in AIanswer:
            if char in inter_dict:
                decoded_string += inter_dict[char]
        return decoded_string
```

Figure 14. AI Audio Parsing.

3.3.3 AI Recognition Methodology

The backend of FASTAPI initiates by tagging and storing audio files while concurrently converting them into WAV format, compatible with the Voice Detection Engine, along with the model specifications and reference files. These are then passed through the Interpret function to the Voice Detection Engine.

The Voice Detection Engine primarily leverages machine learning libraries such as TensorFlow and Keras. These libraries are employed for model loading and handling pickle data. Following the validation check for audio file integrity, the waveform data is segmented using built-in functions, and the resulting beat coordinate list is adjusted and processed. This detection relies on computing the inter-beat intervals, where smaller intervals correspond to closer beats. The function establishes a new coordinate list based on the number of detected beats. Error messages are printed if the number of beats falls outside the specified range or if the input coordinate list is empty. The obtained list of beats along with audio information serves as input data for the model, which eventually returns the list of audio files with the highest probabilities.

The parsing process is as follows (Figure 15): the code begins by loading the required models and pickle files, including the models 'cnn_yang1. h5' and 'model6n. h5', as well as the pickle files '*saveDBindex*', '*saveDBindex1*', '*saveDBprob*', '*saveDBprob1*', '*saveDBwordID*', and '*saveDBwordID1*'. Next, based on the provided file index, the code constructs the path to the audio file. The code then checks if the audio file path exists. If it does, the waveform of the audio file is displayed using the *showAudio* function. If the number of parsed audio segments is 3, further processing is performed for each segment. Each segment is saved as a temporary file. A new spectrogram is obtained for each segment. The spectrogram is converted to the format required by the models. The audio segment is predicted using the *cnn_yang1* and *model6n* models, resulting in predictions. The concatenated results, along with related data, are passed to the *newevaluateResult* function for further processing, resulting in a list of found commands and their probabilities. The processing results are logged in a log file. Based on the length of the found command list, the results are determined. If the list is empty, it indicates that the speech couldn't be fully recognized. If the number of parsed audio segments is not 3, an error message is returned. Otherwise, it may indicate one to three results, and the merged result will be returned.

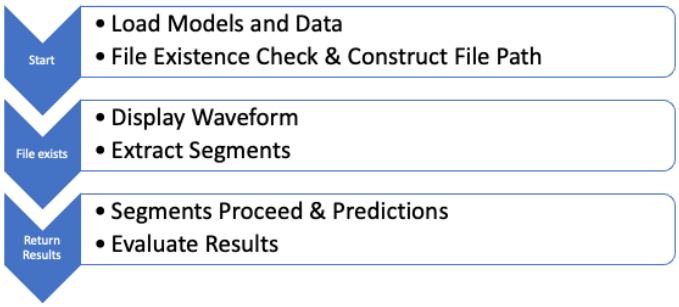

Figure 15. Audio Parsing with AI Models.

4. Optimization and Evaluation

This section evaluates the quality of the code and outlines multithreading optimization, using middleware to improve the difficulties encountered in code debugging, and AI speech recognition accuracy analysis, highlighting their crucial roles in enhancing efficiency and accuracy in AI-powered applications.

4.1 Multithreading optimization

The use of multithreading design in the Voice Detection Engine API has several advantages. Firstly, it allows for parallel processing of speech recognition operations. As multiple users send requests to the API simultaneously, each request can be assigned to a separate thread, enabling the system to handle multiple requests concurrently. This parallel processing capability significantly reduces the overall response time, ensuring a smoother and more efficient user experience.

Moreover, the multithreading design enhances system scalability. By efficiently utilizing system resources, such as CPU cores, the API can handle a higher volume of requests without sacrificing performance. This scalability is crucial in scenarios where the API is expected to handle a large number of concurrent users or when there is a sudden surge in request traffic.

The producer-consumer model employed in the Voice Detection Engine further optimizes the efficiency of the multithreading design. With this model, the producer threads deposit data into a shared buffer, while the consumer threads retrieve and process the data. By decoupling the data production and consumption processes, each thread can operate independently, allowing for efficient utilization of system resources. This design choice not only improves the overall performance of the system but also enhances its maintainability. It becomes easier to manage and troubleshoot issues related to specific threads without affecting the

functionality of the entire system.

In addition to improving efficiency, the multithreading design also enhances the system's stability and fault tolerance. Isolating the execution of time-consuming operations in separate threads ensures that if one thread encounters an error or crashes, it does not affect the execution of other threads. This fault isolation mechanism prevents the entire system from being disrupted due to a single thread failure, providing a more robust and reliable API.

The integration of multithreading design in the Voice Detection Engine API offers significant benefits in terms of efficiency, scalability, maintainability, and fault tolerance. By allowing parallel processing of speech recognition operations and decoupling the production and consumption of data, the API can handle a higher volume of requests, reduce wait time, and improve the overall responsiveness and performance of the system.

4.2 Concurrency in loading models

Concurrency in loading models and data refers to the ability to perform multiple loading tasks simultaneously. It offers several advantages:

- Improved Performance: Loading models and data can be time-consuming, especially when dealing with large files or numerous resources. By leveraging concurrency, multiple tasks can be executed in parallel, leading to faster overall loading times. This allows for better resource utilization and maximizes the available computing power.
- Efficient Resource Utilization: Concurrency enables the efficient utilization of system resources. While one task is waiting for I/O operations (such as reading files or network requests), other tasks can be executed. This prevents resource idle time and ensures that the system is fully utilized, leading to higher throughput and improved efficiency.
- Enhanced Responsiveness: Concurrency ensures that loading tasks do not block the execution of other parts of the program. By offloading the loading process to separate threads or processes, the main program can remain responsive and continue executing other operations concurrently. This is particularly important in scenarios where real-time responses or user interactions are required.

This code snippet (Figure 16) shows the concurrency in loading models. In the given code, concurrency is applied to load models and pickle data using *concurrent.futures.ThreadPoolExecutor*. The *load_models_and_pickles* function first creates a thread pool executor. It then uses the executor. map method to concurrently execute the

load_model_task function for each model path and the *load_pickle_file* function for each pickle path. By utilizing the thread pool executor, multiple models loading tasks and pickle loading tasks are executed concurrently. This approach maximizes the utilization of available CPU resources and reduces the overall loading time for models and data. The function returns two lists: models containing the loaded models and *pickle_data* containing the loaded pickle data. These lists represent the results of the concurrent loading tasks. This results in improved performance, efficient resource utilization, and enhanced program responsiveness, ultimately leading to a more efficient and responsive application.

```python
# 并发加载模型
def load_model_task(model_path):
    print("model loading from path:", model_path)
    return load_model(model_path)

# 并发加载数据
def load_pickle_file(pickle_path):
    with open(pickle_path, 'rb') as fp:
        return pickle.load(fp)

# 加载模型和数据
def load_models_and_pickles(model_paths, pickle_paths):
    # Load models
    with concurrent.futures.ThreadPoolExecutor() as executor:
        models = list(executor.map(load_model_task, model_paths))

    # Load pickles
    with concurrent.futures.ThreadPoolExecutor() as executor:
        pickle_data = list(executor.map(load_pickle_file, pickle_paths))

    return models, pickle_data
```

Figure 16. Concurrency in Loading Models

4.3 Code Debugging and Testing

Debugging is a crucial aspect of the development process, allowing developers to effectively identify and resolve issues within the codebase. FastAPI is a powerful framework for building web applications with great performance and scalability. However, one limitation of FastAPI is the lack of a built-in function for printing requests, which can be seen as a debugging flaw. Without this feature, developers may find it challenging to inspect and troubleshoot the request content, making it more difficult to identify and resolve issues within the codebase. While FastAPI offers custom middleware functions as a workaround, implementing and configuring these functions can be time-consuming and require additional effort. Having a dedicated debugging feature would streamline the debugging process and provide a more user-friendly experience for developers.

In FastAPI, there are various methods to use middleware for debugging purposes. One common approach is to create a custom middleware function that allows developers to log information about incoming requests and outgoing responses. Middleware plays a crucial role

in enhancing the debugging experience by providing insights into the request and response flow. The function can be used to analyze and print relevant details for debugging purposes. By leveraging the request object and the *call_next* function as parameters, developers can access and inspect the request content.

The function intercepts the request before it reaches the main application code, allowing developers to perform debugging operations. Within the middleware function, relevant information such as request headers, parameters, and request body can be logged. This provides insights into how the request is processed and helps identify any potential issues.

The code snippet (Figure 16) demonstrates logging incoming requests and outgoing responses in FastAPI using middleware. The *@app.middleware("http")* decorator indicates that this middleware function should be applied to all HTTP requests in the FastAPI application. The function accepts two parameters: request, representing the incoming request object, and *call_next*, which is a callable object representing the next middleware function or the main application endpoint. Within the function, developers can perform operations related to request logging. In this specific example, a logger object is used to log information about the request. The *logger.info()* method is called to log details such as the request method, URL, and request headers. This allows developers to track and analyze incoming requests. After logging the request information, the *call_next* function is invoked with the request parameter, allowing the execution to continue to the next middleware or the main application endpoint. The response from subsequent middleware or endpoints is then captured.

After executing the subsequent middleware or endpoint, the function continues to log information about the outgoing response. This includes the response status code and response headers. Developers can use this information to monitor the response being sent back to the client. Finally, the function returns the response, ensuring that the response is passed back through the middleware chain or delivered to the client.

```python
@app.middleware("http")
async def log_requests(request: Request, call_next):
    logger.info(f"Received request: {request.method} {request.url}")
    logger.info(f"Request headers: {request.headers}")

    response = await call_next(request)

    logger.info(f"Sent response: {response.status_code}")
    logger.info(f"Response headers: {response.headers}")

    return response
```

```
INFO:     127.0.0.1:60541 - "POST /upload HTTP/1.1" 200 OK
INFO:main:Received request: POST http://localhost:8000/upload
INFO:main:Request headers: Headers({'host': 'localhost:8000',
ntent-length': '125426', 'sec-ch-ua': '"Not_A Brand";v="8", "C
e";v="120"', 'sec-ch-ua-platform': '"macOS"', 'sec-ch-ua-mobil
a/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (
.0.0 Safari/537.36', 'content-type': 'multipart/form-data; bou
ZCGFqAhZhsVkl', 'accept': '*/*', 'origin': 'http://localhost:3
ite', 'sec-fetch-mode': 'cors', 'sec-fetch-dest': 'empty', 're
', 'accept-encoding': 'gzip, deflate, br', 'accept-language':
q=0.7,zh-CN;q=0.6'})
INFO:main:Sent response: 200
INFO:main:Response headers: MutableHeaders({'content-length':
ion/json', 'access-control-allow-origin': '*', 'access-control
```

Figure 17. Example of Debugging Log Output

4.4 AI Speech Recognition Accuracy Analysis

Using a dataset consisting of 48 commonly used Cantonese Chinese characters (Figure 1), the testing was conducted. This test dataset comprises Cantonese speech containing 1 to 3 recognizable Cantonese phrases. We conducted three sets of tests, each containing randomly selected characters, which were inputted into three different user models for comparison. The test is correct if one of the few results it returns contains the correct answer. The test results (Table 1) demonstrate an overall satisfactory performance, with the models achieving an aggregate accuracy of 97.9%. Although the specific recognition accuracy may vary depending on factors such as user pronunciation, the AI models exhibited consistent performance. This model demonstrates extremely high accuracy. However, due to the model being trained specifically on individual patients and lacking a comprehensive dataset with training and testing sets, it is difficult to determine if the model is overfitting. It is recommended to collect more data from different individuals with aphasia and include them in the training set. This can enhance the model's ability to generalize to different individuals, making it more applicable to inputs in general cases (since training a model individually for each person is costly). Additionally, data augmentation can be applied to the existing training data by introducing more variations and noise. Alternatively, techniques such as transfer learning and model optimization can be employed.

Model 1	15/16 tests passed	93% accuracy
Model 2	16/16 tests passed	100% accuracy
Model 3	16/16 tests passed	100% accuracy
Overall		97.9% accuracy

Table 1. AI Speech Recognition Accuracy Test Result.

There are various methods to test web response time. Modern browsers provide built-in network developer tools that can be used to monitor and analyze the webpage loading process. These tools typically include a network panel that displays time information for requests and responses, including DNS resolution time, connection time, sending time, receiving time, and more. In this project, since there is no specific requirement for response speed, the calculation focuses on the time it takes from sending the audio to receiving the parsed text. The API reflects good efficiency, with an average response time of less than 4 second with the help of multithreading.

5. DEPLOYMENT ON AWS

This section describes the AWS server deployment options, process, problems encountered, and solutions.

5.1 Reason for deployment on AWS EC2

Deployment is a crucial step that enables a web application to be accessed on the internet, facilitating user interaction and functionality. Deploying an application involves hosting it on a server or a cloud platform to make it accessible via the internet. Deployment allows users to access the application from anywhere through an internet connection. By making the application available online, it can reach a wider audience and cater to users in different locations. Additionally, deploying the application on a cloud platform enables easy scalability. As the user base grows or the application's demand increases, the cloud infrastructure can dynamically allocate resources to handle the load. This ensures that the application remains responsive and performs well even during peak usage periods. In the case of this project, this deployment strategy also helps validate the developed product's adaptability and reliability across different browsers. In this specific deployment scenario, choosing AWS EC2 as the server is primarily based on the robust infrastructure offered by AWS and cost considerations.

5.2 Deployment process

There are several steps of deployment. Figure 18 is a brief summary of the whole deployment process.

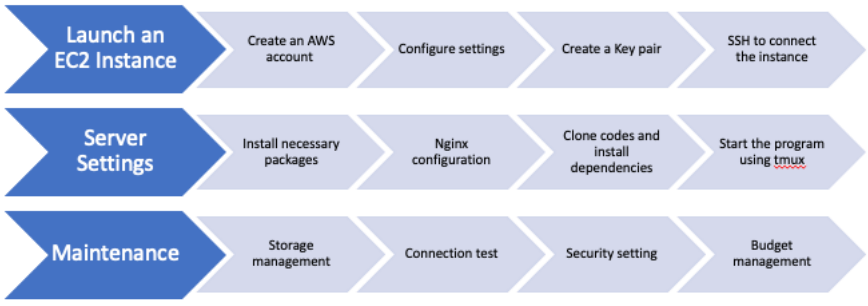

Figure 18. Deployment Process

- **Deploying an Application on AWS EC2**: To begin the deployment process, an AWS account needs to be created and configure the necessary settings. Once you have

registered an account, access the EC2 Dashboard. Configure the instance by selecting a suitable machine image, such as the latest long-term supported version of Ubuntu. Choose an appropriate instance type, such as *t2.micro*. Additionally, create a Key pair for SSH login, specifying the key pair name, type (RSA), and private key format (.pem). After generating the key pair, download the associated key file.

- **Server Settings**: Once the EC2 instance is up and running, the server environment for hosting the application need to be set up. Connect to the EC2 instance using SSH and install required packages using package management tools like *apt* or *yum*. Create a Nginx configuration file to define how incoming requests should be handled. Nginx is deployed as a reverse proxy server, driven by its reputation for high performance, efficient resource usage, and scalability. By acting as an intermediary between clients and servers, Nginx enhances security, load balancing, and the overall reliability of the system. This involves specifying the server's name, listening port, and proxy pass settings. The configuration file is usually stored under the directory /etc/nginx/sites-enabled/. Clone the application's source code repository to the EC2 instance using a version control system like *Git.* Once the code is in place, install any necessary dependencies using the appropriate package manager, such as pip for Python. To run the application, consider using a terminal multiplexer like *tmux*. This allows the program to continue running even when the SSH session is disconnected. Launch the application using the appropriate command.
- **Maintenance**: Once the application is deployed, ongoing maintenance is required to ensure its smooth operation. Monitor the storage usage of the EC2 instance regularly. If additional storage is needed, modify the volume associated with the instance by increasing its size. Perform periodic connection tests to verify that the application is accessible from the internet. This can be done using tools like *netcat* or *cURL* to check connectivity and ensure proper responses. Review and update the security settings of the EC2 instance as needed. This includes managing inbound and outbound traffic rules, configuring firewalls, and securing SSH access. Monitor resource usage, consider optimizing instance types, and leverage cost management tools provided by AWS to control expenses effectively.

5.3 Limitations and solutions

Due to budget constraints and server hardware limitations, there is insufficient disk space to store all audio content. In this situation, one approach is to use an overwrite mechanism in the database to handle audio content. Additionally, due to the lack of space to store all three speech models, only one of them can be used for demonstration. To address this limitation,

strategies such as data compression techniques can be considered to transfer unnecessary data to external storage services like Amazon S3 or optimize data storage to minimize disk space requirements. These measures help maximize available storage and effectively manage limited disk space.

Without an SSL certificate and a purchased domain name, the deployed application may be marked as insecure by popular browsers such as Chrome, Safari, and Firefox. Browsers automatically redirect HTTP requests to HTTPS, which may result in the server being unable to receive certain requests. One viable solution is to grant permission to the server's IP address in the Chrome browser settings. However, this approach is not entirely secure and should only be used when developers implicitly trust the IP address. To overcome this limitation, consider using an SSL certificate to enable HTTPS communication, ensuring secure data transmission between the server and clients. Additionally, purchasing a domain name and configuring DNS settings allow users to securely access the application using a recognizable and trusted URL.

During the deployment process, it is worth mentioning that the frontend application directly uses port 3000 without utilizing Nginx as a reverse proxy. To overcome this limitation, it is recommended to incorporate Nginx as a reverse proxy server. By configuring Nginx as a reverse proxy, requests on ports 3000 and 80 can be redirected to their respective backend and frontend servers, ensuring efficient traffic management and improving the overall security and performance of the entire application.

6. Conclusion

To address the current inadequacies in assistive applications for aphasia patients, this study endeavors to integrate full-stack development with AI speech recognition to provide convenience in daily life for aphasia patients. This study successfully developed a web application utilizing React as the frontend and FASTAPI as the backend as a platform for AI deployment. Key contributions include the development of a speech recognition platform capable of achieving a recognition accuracy of over 97% for 46 commonly used Cantonese Chinese characters and words, with platform response times shortened to under 4 second. Data were sourced from local simulated testing. Further research will focus on enhancing system stability, compatibility, and the security of transmitted information. Currently, this study emphasizes backend integration and development. At present, due to time and budget constraints, the deployment is still in the experimental stage. At the same time, there is no database system that can be stored on a large scale. The limitations of this study include a lack

of generalizability in patient interactions themselves. Since the AI model is trained based on individual cases, then deployed, and further accessed through a web application, the efficiency is relatively low. Although the platform has established mature model interfaces and templates that can be quickly deployed once trained, data collection and model training remain among the primary factors limiting user adoption. A feasible solution is to develop an online data collection and training interface in the future, realizing the integration of AI model application from data collection to training and deployment in a unified web application. At the same time, it is necessary to enhance the robustness of the server and establish additional database systems. This is expected to expand the platform's reach to serve more aphasia patients.

References

[1] van de Sandt-Koenderman, W. M. E. “Aphasia rehabilitation and the role of computer technology: Can we keep up with modern times?” in International journal of speech-language pathology, 2011, 13(1), pp. 21-27.

[2] Azevedo, N., Kehayia, E., Jarema, G., Le Dorze, G., Beaujard, C., & Yvon, M. “How artificial intelligence (AI) is used in aphasia rehabilitation: A scoping review.” in Aphasiology, 2024. 38(2), pp. 305-336.

[3] Adikari, A., Hernandez, N., Alahakoon, D., Rose, M. L., & Pierce, J. E. “From concept to practice: a scoping review of the application of AI to aphasia diagnosis and management.” in Disability and Rehabilitation, 2023.1-10.

[4] 錢志安. 粵語研究新資源:《香港二十世紀中期粵語語料庫》, 2013.

Appendixes

Appendix A: Tree of the project files, depth 3

Generated by: tree -L 3 > tree.txt

```
.
├── LICENSE
├── README.md
├── backend
│   ├── Data
│   │   └── data.txt
│   ├── README.md
│   ├── Request
│   │   ├── InterpretPostRequest.py
│   │   ├── MessagePostRequest.py
│   │   ├── SamplePostRequest.py
│   │   ├── __init__.py
│   │   └── __pycache__
│   ├── Utility
│   │   ├── ServerSettings.py
│   │   ├── __init__.py
│   │   └── __pycache__
│   ├── VD8_FullRecord.txt
│   ├── VoiceDetectionEngin.py
│   ├── __pycache__
│   │   ├── VoiceDetectionEngin.cpython-311.pyc
│   │   ├── VoiceDetectionEngine.cpython-311.pyc
│   │   └── main.cpython-311.pyc
│   ├── bin
│   │   ├── Activate.ps1
│   │   ├── activate
│   │   ├── activate.csh
│   │   ├── activate.fish
│   │   ├── dotenv
│   │   ├── pip
│   │   ├── pip3
│   │   ├── pip3.9
│   │   ├── python -> python3
│   │   ├── python3 -> /Applications/Xcode.app/Contents/Developer/usr/bin/python3
│   │   ├── python3.9 -> python3
│   │   ├── uvicorn
│   │   └── watchfiles
│   ├── dict.png
│   ├── lib
│   │   └── python3.9
│   ├── main.py
│   ├── models
│   │   ├── Models-LHo-20230111
│   │   ├── Models-P1-20230207
│   │   └── Models-P2-20230207
│   ├── pyvenv.cfg
│   ├── requirement.txt
│   └── users.json
├── package-lock.json
├── package.json
├── testAudio
│   ├── ID15-test_RCD_.wav
│   ├── ID16-est_S34_.wav
│   ├── ID17-test_VFZ_.wav
```

```
│   ├── ID18-test_VY0_.wav
│   ├── ID19-test_YJ2_.wav
│   ├── ID20-test_0,+_.wav
│   ├── ID21-test_0,)_.wav
│   ├── ID22-test_0,+_.wav
│   ├── ID23-test_!,+_.wav
│   ├── ID24-test_!(R_.wav
│   ├── ID25-test_(R0_.wav
│   └── ID26-test_0,)_.wav
└── tree.txt

16 directories, 48 files
```

Appendix B: main.py

```
def import_libraries():
    from fastapi import FastAPI, HTTPException, File, UploadFile
    from fastapi.middleware.cors import CORSMiddleware
    from pydantic import BaseModel
    from typing import List, Optional
    from fastapi.middleware.trustedhost import TrustedHostMiddleware
    import json
    import os
    import logging
    import requests
    import ffmpeg
    from fastapi import Request
    import shutil
    from pydub import AudioSegment
    import aiohttp
    from starlette.responses import JSONResponse
    import wave
    from pathlib import Path
    import magic
    import subprocess

    return FastAPI, HTTPException, File, UploadFile, CORSMiddleware, BaseModel, List, Optional, TrustedHostMiddleware, json, os, logging, requests, ffmpeg, Request, shutil, AudioSegment, aiohttp, JSONResponse, wave, Path, magic, subprocess
```

```
FastAPI, HTTPException, File, UploadFile, CORSMiddleware, BaseModel, List, Optional, TrustedHostMiddleware, json, os, logging, requests, ffmpeg, Request,
shutil, AudioSegment, aiohttp, JSONResponse, wave, Path, magic, subprocess = import_libraries()



from VoiceDetectionEngin import *



app = FastAPI()



class AudioURL(BaseModel):

    url: str



inter_dict = {'0':'我', '1':'要','2':'去', '3':'廁', '4':'所', '5':'返','6':'睡','7':'房', '8':'書', '9':'廚',

            'A':'刷', 'B':'牙','C':'洗', 'D':'面', 'E':'開', 'F':'電','G':'腦','H':'閂', 'I':'燈', 'J':'出',

            'K':'客', 'L':'廳','M':'睇', 'N':'視', 'O':'叫', 'P':'人','Q':'鐘','R':'想', 'S':'上', 'T':'床',

            'U':'落', 'V':'攞','W':'手', 'X':'機', 'Y':'畀', 'Z':'話',

            '!':'你', ',':'飲','+':'茶', '(':'唔', ')':'水', '$':'吃','%':'面','#':'飯', '@':'早', '&':'餐'

            }



# 添加 CORS 中间件

app.add_middleware(

    CORSMiddleware,

    allow_origins=["*"],  # 允许所有来源

    allow_credentials=True,

    allow_methods=["*"],  # 允许所有方法

    allow_headers=["*"],  # 允许所有头部

)



# 添加 TrustedHostMiddleware，确保正确处理主机头

app.add_middleware(TrustedHostMiddleware, allowed_hosts=["*"])



# 配置日志记录

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(__name__)
```

```
# 添加请求和响应日志中间


@app.middleware("http")
async def log_requests(request: Request, call_next):
    logger.info(f"Received request: {request.method} {request.url}")
    logger.info(f"Request headers: {request.headers}")

    response = await call_next(request)

    logger.info(f"Sent response: {response.status_code}")
    logger.info(f"Response headers: {response.headers}")

    return response


class User(BaseModel):
    usn: str
    pwd: Optional[str] = None


# 加载现有用户数据
try:
    with open("users.json", "r") as file:
        users_db = json.load(file)
except FileNotFoundError:
    print("not file path")
    users_db = []


@app.post("/register")
def register_user(user: User):
    existing_user = next((u for u in users_db if u["usn"] == user.usn), None)
    if existing_user:
        raise HTTPException(status_code=400, detail="Username already exists")
```

```
    users_db.append({"usn": user.usn, "pwd": user.pwd})

    # write users' data into JSON file
    with open("users.json", "w") as file:
        json.dump(users_db, file, indent=2)

    return {"message": "User registered successfully"}

@app.post("/login")
def login_user(user: User):
    existing_user = next((u for u in users_db if u["usn"] == user.usn), None)
    if not existing_user or existing_user["pwd"] != user.pwd:
        raise HTTPException(status_code=401, detail="Invalid credentials")

    return {"message": "Login successful"}

@app.post("/delete")
def delete_user(user: User):
    # 请你修改这个路由
    existing_user = next((u for u in users_db if u["usn"] == user.usn), None)
    if not existing_user:
        raise HTTPException(status_code=404, detail="User not found")

    users_db.remove(existing_user)

    # 将更新后的用户数据写入 JSON 文件
    with open("users.json", "w") as file:
        json.dump(users_db, file, indent=2)

    return {"message": "User deleted successfully"}

# 使用 magic 来检测文件类型
```

```
def detect_file_type(file_path):
    mime = magic.Magic(mime=True)
    file_type = mime.from_file(file_path)
    return file_type

# Maintain a counter to generate sequential file names
file_counter = 1

# Function to find the next available file number
def find_next_available_number(directory):
    file_names = os.listdir(directory)
    file_numbers = [int(name.split(".")[0]) for name in file_names if name.endswith(".wav") and name.split(".")[0].isdigit()]
    if file_numbers:
        return max(file_numbers) + 1
    else:
        return 1

# check whether wav file
def is_wav(file_path):
    try:
        with wave.open(file_path, 'rb') as f:
            print(file_path, "is a wav file")
            # Check if the file is a WAV file
            return f.getnchannels() > 0
    except wave.Error:
        return False

# 在线录制文件路由
@app.post("/record-audio")
async def record_audio(audio: UploadFile = File(...)):
    try:
```

```
# Specify the path where you want to save the uploaded files

save_path = "./Data"

# Ensure the save path exists, if not, create it

Path(save_path).mkdir(parents=True, exist_ok=True)


# Find the next available file number

file_counter = find_next_available_number(save_path)


# Generate the temp file name

file_name = f"temp.webm"

file_path = os.path.join(save_path, file_name)


# Generate the file name with sequential numbering

wav_file_name = f"{file_counter}.wav"

wav_file_path  = os.path.join(save_path, wav_file_name)


# Save the audio file

with open(file_path, "wb") as f:

    f.write(await audio.read())


# Determine file type

file_type = detect_file_type(file_path)

print(f"File type: {file_type}")


if file_type == "video/webm":

    print("Converting ...")

    # Convert webm to wav

    print(wav_file_path)

    try:

        # 使用 FFmpeg 提取音频并将其转换为 WAV 格式

        subprocess. run(['ffmpeg', '-i', file_path, '-vn', '-acodec', 'pcm_s16le', '-ar', '44100', '-ac', '2', wav_file_path])

        print("convert sucess")
```

```
            except ffmpeg.Error as e:
                print(f"ffmpeg error: {e.stderr}")
                return {"error": f"ffmpeg error: {e.stderr}"}

        AIanswer = InterpretAI(file_counter)
        print('APIHost AIanswer (interpret) = ',AIanswer)

        if os.path.exists(wav_file_path):  # 检查文件是否存在
            os.remove(wav_file_path)  # 删除文件
            print(f"文件 {wav_file_path} 已成功删除")
        else:
            print(f"文件 {wav_file_path} 不存在")

        return {"answer": AIanswer}


    except Exception as e:
        return {"error": f"Error uploading audio: {e}"}

# 拖拽/上传文件
@app.post("/upload")
async def upload_file(file: UploadFile = File(...)):
    try:
        # Specify the path where you want to save the uploaded files
        save_path = "./Data"

        # Create the path if it doesn't exist
        os.makedirs(save_path, exist_ok=True)

        # Find the next available file number
        file_counter = find_next_available_number(save_path)
```

```
    # Generate the file name with sequential numbering
    file_name = f"{file_counter}.wav"
    file_path = os.path.join(save_path, file_name)

    # Save the file to the specified path
    with open(file_path, "wb") as f:
        shutil.copyfileobj(file.file, f)

    # Convert the file to WAV format
    if is_wav(file_path) == False:
        audio = AudioSegment.from_file(file_path, format=file.filename.split('.')[-1])
        audio.export(file_path, format="wav")

    print("file_counter", file_counter)

    # Process the uploaded file
    # For now, we will just return the file details
    AIanswer = InterpretAI(file_counter)
    print('APIHost AIanswer (interpret) = ',AIanswer)

    if os.path.exists(file_path):  # 检查文件是否存在
        os.remove(file_path)  # 删除文件
        print(f"文件 {file_path} 已成功删除")
    else:
        print(f"文件 {file_path} 不存在")

    return {"answer": AIanswer}
except Exception as e:
    # 使用日志记录详细的错误信息
    logging.error("An error occurred while uploading the file:", exc_info=True)
    raise HTTPException(status_code=500, detail="An error occurred while uploading the file. Please check the server logs for more details.")
```

```
# 修改密码路由
@app.post("/change-password")
def change_password(user: User):
    existing_user = next((u for u in users_db if u["usn"] == user.usn), None)
    if not existing_user:
        raise HTTPException(status_code=404, detail="User not found")

    # 更新用户密码
    existing_user["pwd"] = user.pwd

    # 将更新后的用户数据写入 JSON 文件
    with open("users.json", "w") as file:
        json.dump(users_db, file, indent=2)

    return {"message": "Password changed successfully"}

# 翻译音频
def InterpretAI(audio_id: int):
    print("audio_id", audio_id)
    AIanswer = run_interpret_audio(audio_id)
    if AIanswer == "...":
        return "Parsing failed"
    elif AIanswer == None:
        return "Upload failed"
    else:
        decoded_string = ''
        for char in AIanswer:
            if char in inter_dict:
                decoded_string += inter_dict[char]
        return decoded_string
```

```
# 根页面

@app.get("/")

async def root():

    return {"message":"Welcome to my final year project app! You may explore more to the docs page : ) -- Bob Jiang"}
```

Appendix C: Dependency list

absl-py==2.1.0

aiohttp==3.9.3

aiosignal==1.3.1

annotated-types==0.6.0

anyio==4.2.0

asttokens==2.4.1

astunparse==1.6.3

attrs==23.2.0

audioread==3.0.1

bcrypt==4.0.1

cachetools==5.3.2

capstone==5.0.1

certifi==2023.7.22

cffi==1.16.0

charset-normalizer==3.3.2

click==8.1.7

colorama==0.4.6

colored-traceback==0.3.0

contourpy==1.2.0

crypto==1.4.1

cycler==0.12.1

decorator==5.1.1

executing==2.0.1

fastapi==0.108.0

ffmpeg==1.4

flatbuffers==23.5.26

fonttools==4.47.2

frozenlist==1.4.1

gast==0.5.4

google-auth==2.27.0

google-auth-oauthlib==1.2.0

google-pasta==0.2.0

grpcio==1.60.0

h11==0.14.0

h5py==3.10.0

idna==3.4

imageio==2.34.0

intervaltree==3.1.0

ipython==8.20.0

jedi==0.19.1

joblib==1.3.2

keras==2.15.0

kiwisolver==1.4.5

lazy_loader==0.3

libclang==16.0.6

librosa==0.10.1

llvmlite==0.41.1

Mako==1.3.0

Markdown==3.5.2

MarkupSafe==2.1.3

matplotlib==3.8.2

matplotlib-inline==0.1.6

ml-dtypes==0.2.0

msgpack==1.0.7

multidict==6.0.5

Naked==0.1.32

natsort==8.4.0

networkx==3.2.1

numba==0.58.1

numpy==1.26.1

oauthlib==3.2.2

opencv-python==4.8.1.78

openvino==2023.1.0

opt-einsum==3.3.0

packaging==23.2

paramiko==3.3.1

parso==0.8.3

pexpect==4.9.0

Pillow==10.1.0

platformdirs==4.1.0

plumbum==1.8.2

pooch==1.8.0

prompt-toolkit==3.0.43

protobuf==4.23.4

psutil==5.9.6

ptyprocess==0.7.0

pure-eval==0.2.2

pwntools==4.11.0

pyasn1==0.5.1

pyasn1-modules==0.3.0

pycparser==2.21

pydantic==2.5.3

pydantic_core==2.14.6

pydub==0.25.1

pyelftools==0.30

pygame==2.5.2

Pygments==2.16.1

PyNaCl==1.5.0

pyparsing==3.1.1

PyQt3D==5.15.5

PyQt5==5.15.7

PyQt5-sip==12.11.0

PyQtChart==5.15.6

PyQtDataVisualization==5.15.5

PyQtNetworkAuth==5.15.5

PyQtPurchasing==5.15.5

PyQtWebEngine==5.15.6

pyserial==3.5

PySocks==1.7.1

python-dateutil==2.8.2

python-magic==0.4.27

python-multipart==0.0.6

pywatchman==1.4.1

PyYAML==6.0.1

requests==2.31.0

requests-oauthlib==1.3.1

ROPGadget==7.4

rpyc==5.3.1

rsa==4.9

scikit-image==0.22.0

scikit-learn==1.4.0

scipy==1.12.0

sdaxen-python-utilities==0.1.5

shellescape==3.8.1

six==1.16.0

sniffio==1.3.0

sortedcontainers==2.4.0

sounddevice==0.4.6

soundfile==0.12.1

soxr==0.3.7

stack-data==0.6.3

starlette==0.32.0.post1

stego-lsb==1.5.4

TBB==2021.11.0

tensorboard==2.15.1

tensorboard-data-server==0.7.2

tensorflow==2.15.0

tensorflow-estimator==2.15.0

tensorflow-io-gcs-filesystem==0.34.0

tensorflow-macos==2.15.0

termcolor==2.4.0

threadpoolctl==3.2.0

tifffile==2024.2.12

traitlets==5.14.1

typing_extensions==4.9.0

unicorn==2.0.1.post1

urllib3==2.0.7

utility==1.0

uvicorn==0.25.0

wcwidth==0.2.13

Werkzeug==3.0.1

wrapt==1.14.1

xlwt==1.3.0

yarl==1.9.4