Development of Multilingual and Code-Mixed Translation App for Malaysian

By

OI JIA ZHUN

A REPORT
SUBMITTED TO
Universiti Tunku Abdul Rahman
in partial fulfillment of the requirements
for the degree of
BACHELOR OF COMPUTER SCIENCE (HONOURS)
Faculty of Information and Communication Technology
(Kampar Campus)

FEB 2026

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Cik Nurul Syafidah Binti Jamil who has given me this bright opportunity to engage in a mobile application project. It is my first step to establish a career in the mobile development field. A million thanks to you.

To a very special person in my life, Liew Ji Wen, for her patience, unconditional support and love, and for standing by my side during hard times. Finally, I must say thanks to my parents and my family for their love, support and continuous encouragement throughout the course.

ABSTRACT

This project focuses on developing a multilingual and code-mixed translation mobile application to solve the communication challenges faced in Malaysia's multilingual society. Most existing translation apps are not able to recognize or translate mixed-language speech well, especially when users switch between Malay, Chinese and English within the same sentence. To address this problem, a fine-tuned SeamlessM4T ASR model is used to improve the accuracy of recognizing code-mixed audio. The recognized text is then processed by Google Gemini to generate translations. Google Cloud Text-to-Speech is used to produce natural-sounding voice results. The whole system is implemented as a mobile application using React Native with Firebase for backend management. The aim of this project is to provide a more accurate, user-friendly and practical translation tool that supports real-time code-mixed communication in daily life.

Area of Study: Natural Language Processing, Mobile Application Development

Keyword: Code-Mixed Translation, Automatic Speech Recognition (ASR), Multilingual Processing, Large Language Models (LLM), Mobile Application, Text-to-Speech (TTS), Natural Language Processing (NLP)

TABLE OF CONTENT

LIST OF FIGURE

LIST OF TABLE

CHAPTER 1 – PROJECT BACKGROUND

1.1 Introduction

Malaysia is a multicultural country. At here, people are talking variant type of language. For example, Malay, Chinese and English. Although Malay serves as the official language in Malaysia, but still a lot of other languages and dialects are integrated into the daily lives [1]. This shows the rich culture in Malaysia, but it also leads to significant communication barriers to people.

However, current translation application often fails to provide accurate results in these daily communication scenarios. Most commercial application such as Google Translate, Microsoft Translator, DeepL, Papago and so on, are only designed under the assumption of monolingual input. Hence, the code-mixing, which mean a sentence including more than one language, often failed to process and resulting inaccurate or nonsensical output, making these application impractical for Malaysian.

Furthermore, the demand for a specialized code-mixing translation tool is becoming more urgent due to the growing population in Malaysia. A recent data shows that applications from international students increased by approximately 25% in 2024 [2]. Besides, as if 2024, there are 2.4 million documented migrant workers active in the domestic labour market [3]. If this issue still being neglected, the communication barriers will grow exponentially and reduce service efficiency. This could even hinder the socio-economic development of the entire country.

To avoid the disasters listed above, this project proposes to develop a mobile translator application specifically designed for daily code-mixed communication. In this project, translator application integrates fine-tuned multilingual Automatic Speech Recognition (ASR) model, Large Language Model (LLM) as translation engines and real-time voice syntheses. The application focuses on daily practical scenarios such as retail transactions, food services and navigation. This application also support voice input and output, allowing users to engage in natural dialogue and converting spoken input into the listener's preferred language for immediate playback.

Through this application, the communication barrier will be resolved and foster better the understanding between different cultural groups. Thereby reduce the misunderstandings and enhance user confidence during cross-linguistic conversation. Ultimately, this application promote social integration within Malaysia's multicultural society.

1.2 Problem Statement and Motivation

The effectiveness of these commercial applications is hindered by several technical and environmental factors.

Problem 1: Standard Machine Translation (MT) Systems Unable Handle Semantic Ambiguity and Rare Word Senses

The models often generate incorrect output when they encounter semantic ambiguity and rare word senses [4] [5] [6]. Research shows that even advanced MT system translates the sentences that containing ambiguous terms, it only achieves about 50-60% accuracy [4]. This is because parallel training corpora often bias models toward high-frequency word meanings instead of depending on the whole sentence. For example, in the phrase “a blaze between its eyes”, DeepL will incorrectly translates “blaze” as “flame”. On the other hand, LLM able to identify it as “a white stripe”. This error indicates that the current MT systems are outdated and inadequate for performing translation, especially when dealing with semantic ambiguity and rare word senses.

Problem 2: Current ASR Systems in Translation Applications Are Unable to Handle Multilingual Code-Mixed Communication

Although many translation applications are available, but few of them are suitable for multilingual code-mixed communication [7] [8]. This is because most ASR systems that implemented in these translation applications are only designed for single language input. However, in Malaysia, users are frequently switch between Malay, Chinese and English in a single sentence. Besides, code-mixed speech leads to more complex pronunciation patterns, acoustic variability and vocabulary recognition [9]. Therefore, the ASR systems often recognize the only primary language and ignores the other languages. This leads the final result deviated from the user’s original meaning.

Problem 3: Standard LLM Interface Lack a Structured Method for Storing Conversation History with Speaker Distinction

Although users can directly use LLM for translation, LLM interfaces typically record conversations in a simple chronological list that lacks speaker distinction [10]. These interfaces are designed for general conversations and unable to maintain the natural flow of bilingual dialogue. Therefore, without a structured conversation history, users find it difficult to track which speaker said each translated sentence or to refer to specific content in long conversations.

1.3 Project Objectives

Objective 1: To develop a mobile translation application capable of processing multilingual communication

This is to solve the first problem regarding the weakness of standard MT systems in handling semantic ambiguity and rare word senses. To address this, the project utilizes a LLM as the core translation engine instead of traditional MT systems [4] [6]. This approach leverages the model's contextual reasoning to improve translation accuracy for semantic ambiguity and rare word senses. By analyzing the entire sentence context, the application provides more reliable outputs.

Objective 2: To Improve Recognition Accuracy for Multilingual Code-Mixed Communication Through ASR Fine-Tuning

This is to solve the second problem regarding the current ASR systems unable to handle multilingual code-mixed communication. The project focuses on fine-tuning an ASR model to enhance its ability. By using this approach, the system able to identify language boundaries when users switch between Malay, Chinese and English within a single sentence [7]. By improving recognition accuracy, the application will ensure that it always responds accurately to user intent.

Objective 3: To Implement a Structure Conversation History Using a Segregated Chat-Style Interface

This is to solve the third problem regarding the standard LLM interfaces lack a structured method for storing conversation history. To solve this problem, this project refers to the design scheme of T3 and the system of Thomas Kasakowskij et al. [11]. This application adopts a messenger-style interface, which allows for easy tracking and confirmation of conversation history by using chat bubbles [12]. Instead of simple chronological list, this design ensures clear distinction between speakers.

1.4 Project Scope

This project focuses on developing a mobile translation system integrating speech recognition and language modeling technologies. A fine-tuned Meta SeamlessM4T model is used as the ASR component for recognizing speech input. Integrating the fine-tuned ASR ensures accuracy in speech-to-text conversion, especially in environments where users frequently switch languages.

Furthermore, the application utilizes the Google Gemini API as its translation engine. This LLM features advanced contextual reasoning capabilities and performance when handling mixed code input. Besides, the system implements a structured chat flow to organize communication. This allows user A and user B to be visually separated on the interface and clearly identify roles. Next, to enable accessible communication, Google Cloud Text-to-Speech (TTS) is used to convert translated text into speech.

Firebase then serves as the backend for data management. Firebase handles user authentication, translation history and user input. This synchronizes data across devices and maintains reliability. Overall, the project combines these technologies and provides a real-time and contextually accurate translations mobile application.

The scope of this project is subject to several constraints to maintain focus on the core objectives. Firstly, development is limited to the Android environment only. Secondly, the system is designed for conversational text and speech. It does not support the translation of full documents such as PDF, Word, PowerPoint and so on. Lastly, the application does not provide offline translation functionality. Since there are some component rely on cloud-based services.

1.5 Contribution

This study provides several significant contributions to the fields of LLM and multilingual communication in Malaysia. Firstly, it integrates LLM into the translation pipeline rather than relying on MT system. By using this approach, the application can more accurately capture the contextual meanings and reduce the errors that related to linguistic complexity.

Secondly, this project involves the fine-tuning of pre-trained ASR models to enhance ethe recognition of code-mixed speech. This technical contribution ensures more reliable and stable transcriptions are being produced even users frequently switch between different languages. It provides a higher quality of input for the translation engine.

Thirdly, the development of this application contributes a practical solution for real-world communication by supporting real-time voice-to-voice interaction. By focusing on accessibility through a chat-style interface and voice interaction, the tool promotes smoother cross-cultural interactions in daily life

1.6 Report Organization

This report is divided into six main chapters. Chapter 1, the Introduction, provides an overview of the project including the problem statement, motivation, project scope, objectives and contributions. Chapter 2, the Literature Review, evaluates and compares the existing LLM & ASR and analyse existing mobile translation application. Chapter 3, System Methodology and Design, explains the development approach in the project. It details the requirements, system flowchart, use case diagram and description. Chapter 4, System Implementation, focuses on the technical development of the mobile application. Chapter 5, System Evaluation and Discussion, presents the testing results and performance of the application. Chapter 6, Conclusion and Recommendation, summarize the contributions of the project and provides suggestions for future research.

CHAPTER 2 – LITERATURE REVIEW

2.1 Previous Work on Neural Machine Translation and Large Language Models

Traditional MT systems (E.g. statistical machine translation) face limitations in capturing contextual information and often rely on word-by-word or sentence-by-sentence translation. However, with the advancement of deep learning, neural machine translation (NMT) become the mainstream approach, especially models based on the Transformer architecture. Based on this, large-scale multilingual models and optimized NMT frameworks have become widely used in recent years. These models help to improve the translation quality, especially in code-mixed scenarios

MobileNMT is a Transformer-based NMT model designed for real-time translation on mobile devices [13]. By applying pruning and quantization, the model size has been compressed to approximately 15 to 20 MB while still maintaining high translation performance. Since its size has been compressed, the model can be directly deployed on smartphones, allowing for fast translation and offline usage. However, MobileNMT has limited capability in contextual modeling and shows weaker adaptability to code-mixed inputs compared to large-scale multilingual models.

mBART is a multilingual sequence-to-sequence pre-trained model that proposed by Facebook [14]. Its encoder–decoder architecture allows the model to leverages context more effectively, including document-level and conversational inputs. Studies have shown that mBART performs particularly well on low-resource language translation tasks, achieving improvements of up to 12 BLEU points over baseline systems [15]. However, the model is not optimized for code-mixed text and only suitable for single-language translation. Besides, its 611M parameter size is relatively large and makes local deployment become challenge.

Google Gemini is a LLM based on the Transformer architecture [16]. In translation tasks, it integrates multimodal capabilities and enhances contextual modeling, thereby improving semantic coherence and multi-turn dialogue translation performance. The model supports both specifying and automatically detecting the source language. Although its large size makes local deployment unfeasible and necessitates API-based access, Google provides a generative AI tier for developers that is extremely low-cost or even entirely free [18]. Specifically, the Gemini 3 Flash model is optimized for high-volume production, with pricing as low as $0.50 (RM1.97) per 1M

input tokens and $3.00 (RM11.79) per 1M output tokens [17]. This results in cost-effective and highly accessible solution for developing and testing translation prototypes without incurring significant computational expenses. However, consistent operation still relies on a stable internet connection.

ChatGPT demonstrates the potential of general-purpose LLMs in translation tasks [19]. Its strength lies in its strong contextual understanding and outperforms traditional translation, especially in multi-turn dialogue and code-mixed scenarios [20] [21]. However, as the ChatGPT model consists of tens of billions of parameters, it is not feasible for local deployment and can only be used via API calls. Specifically for the latest ChatGPT 5.2 model, its API costs are significantly higher than those of its competitors, with approximately $1.75 (RM6.88) per 1M input tokens and $14.00 (RM55.02) per 1M output tokens [17]. Besides, this also results in dependency on network access and increased computational costs.

Model	Architecture / Type	Advantages	Disadvantages	Deployment
MobileNMT	Transformer / NMT	• Lightweight • Offline • Fast translation	• Limited context modeling • Weak on code-mixed inputs	On-device (smartphones)
mBART	Multilingual Seq2Seq / NMT	• Strong in low-resource translation • Document & dialogue context	• No support for code-mixing • Large size, difficult to local deployment	Server-based only
Gemini	Transformer / LLM	• Strong contextual understanding • Good for code-mixing • Lower cost	• Large size, difficult to local deployment	API only (cloud-based)
ChatGPT	Transformer / LLM	• Strong contextual understanding • Good for code-mixing	• Higher cost • Large size, difficult to local deployment	API only (cloud-based)

Table 2.1: Advantages and Disadvantages of Translation Models

2.2 Previous Work on Automatic Speech Recognition (ASR) for Multilingual and Code-Mixing Speech

With the development of large-scale multilingual models, ASR systems have also achieved significant progress. However, recognizing code-mixing speech remains a challenging task. In Malaysia, code-mixing is very common in daily communication. Therefore, ASR systems capable of robustly handling multilingual and code-mixed inputs are essential. If an ASR system fails to accurately recognize and process multiple languages, it will severely impact the accuracy of subsequent translation.

Whisper is a multilingual ASR model released by OpenAI and it supports around 100 languages [7] [22]. It demonstrates high accuracy in monolingual recognition but performs less effectively on code-mixing speech. Research indicates that Whisper struggles with mixed-language inputs without adaptation but adding language prompts before decoding or applying fine-tuning can reduce error rates. For instance, in Malay–English mixed speech, fine-tuning has been shown to reduce word error rates by approximately 9% [23]. Whisper is open-source and can be deployed locally or accessed via API.

Amazon Transcribe is a cloud-based ASR service provided by AWS and it supports automatic language detection and multilingual speech recognition [24]. It currently supports 31 languages, including Malay, English, and Chinese. Amazon Transcribe can process mixed-language input [25]. It will generate concatenated transcriptions containing recognized text in multiple languages, which can then be passed to external translation systems. However, it only runs on the cloud, requiring continuous internet connectivity and incurring API costs. Moreover, while it handles inter-sentence or inter-utterance switching well, its recognition accuracy for fine-grained intra-sentence code-switching is limited.

Massively Multilingual Speech (MMS) is a multilingual ASR model released by Meta, covering more than 1100 languages [7] [26]. Although it provides impressive language coverage, its default training is based on monolingual utterances. As a result, for code-switched audio, the raw model often misrecognizes or enforces a single-language transcription [7]. MMS is open-source and supports high-performance local deployment. Therefore, it is an attractive option for building customized code-switching ASR systems.

SeamlessM4T is another multilingual ASR model developed by Meta. It was trained beyond 200 languages [7] [27]. Although the base model is not specifically optimized for code-switching, research has shown that incorporating code-switched training data can significantly improve recognition accuracy. For example, in Malay–English speech, fine-tuned SeamlessM4T reduced error rates by as much as 83% compared to the base model [7]. Furthermore, SeamlessM4T is fully open-source and can be deployed locally or on servers. It provides high flexibility for both research and real-world applications.

Microsoft Azure Speech provides multilingual recognition and language identification (LID) functionalities [28]. It allows dynamic switching of input languages within the same session without restarting the recognition pipeline [29]. However, Azure's continuous multilingual recognition currently only captures language switches at the phrase or utterance level, but not at the word level [30]. This causes intra-sentence code-switching transcription less accurate. Azure's strength lies in its enterprise integration and built-in translation pipelines. However, since it is a cloud-based service, it is less optimized for highly dynamic multilingual conversations such as Chinese–Malay–English speech in Malaysia.

System	Type	Malay Support	Chinese Support	English Support	Code-Switch Handling	Deployment
OpenAI Whisper	Open-source model	✓	✓	✓	Weak	Local / API
Amazon Transcribe	Commercial API	✓	✓	✓	Moderate	Cloud only
Meta MMS	Open-source model	✓	✓	✓	Weak	Local
Meta SeamlessM4T	Open-source model	✓	✓	✓	Strong	Local / Server
Microsoft Azure Speech	Commercial API	✓	✓	✓	Limited	Cloud only

Table 2.2: Language Support and Deployment Comparison of ASR

2.3 Review the Existing Application

2.3.1 Google Translate

Voice Input

The Google Translate mobile app fully supports voice input. It provides a microphone dictation feature and even a real-time Conversation Mode for bilingual dialogues [31] [32]. In Conversation Mode, users select two languages first. After that, the app will automatically detects which one is being spoken and translate both sides of the dialogue in real time. This allows users to engage in bilingual exchanges without manually switching input modes.

Code-Mixed Input

As shown in Figure 2.1, Google Translate does not officially support code-mixed input. The system can automatically detect one main language in the input but cannot process multiple languages simultaneously, regardless of whether the input is in text or speech form. For example, if the input mixes Malay and English, Google Translate will only identify one language and partially translate the other, but may misinterpret or ignore the non-primary language [33]. Although Conversation Mode can support bilingual communication, it still assumes that the input is one language at a time instead mixed simultaneously [32]. Therefore, Google Translate does not support handling multi-language mixed input.

Figure 2.1: Google Translate Fail to Handle Code-Mix Input

Translation Accuracy

Google Translate is widely recognized for its extensive language coverage. It supports more than 100 languages through NMT. In general use, the system provides quick and readable transaltions for most common phrases [34]. However, a limitation is that its output can be overly literal and it results in less natural phrasing for long or complex sentences [35]. Furthermore, it also occur deviations when handling colloquial expressions or formal tones. Overall, Google Translate's accuracy in daily use is enough high and suitable for quickly grasping the general meaning.

Language Support

Google Translate supports more than 100 languages, including Malay, Chinese, and English [36]. All three are fully supported for both text and voice translation.

2.3.2 DeepL Translator

Voice Input

DeepL was originally known for text translation, but it has now added voice features to its mobile app. The DeepL mobile app allows users to speak into their phone and receive translations [37] [38]. This means that casual users can translate via voice input. By 2024, DeepL introduced the *"DeepL Voice"* feature which provides real-time speech translation for face-to-face communication [39]. As of mid-2025, DeepL's voice input supports a growing list of languages, including English, German, Japanese, Korean, French, and Mandarin Chinese. Overall, DeepL's mobile app does indeed support voice input in multiple major languages. It allow users to speak and get translations displayed instantly.

Code-Mixed Input

As shown in Figure 2.2, DeepL does not have a dedicated feature for code-mixed input. It expects only one language at a time like Google Translate. Users must select or auto-detect a single source language for each translation, regardless of whether the input is in text or speech form. If you input a sentence mixing other languages, DeepL will likely misidentify the language. Currently, there is no evidence that DeepL can handle multiple languages simultaneously within a single sentence and it does not offer any "mixed-language mode." Therefore, spontaneous in-sentence code-switching is beyond DeepL's capabilities. In summary, DeepL does not support code-mixed input.

Figure 2.2: Deepl Fail to Handle Code-Mix Input

Translation Accuracy

DeepL's translate accuracy is very high and frequently praised for the languages it supports. Its neural translation engine is renowned for its deep understanding of context and producing natural, fluent output [35]. In comparative tests, DeepL often preserves nuances better and sounds closer to human translation. It performs particularly well with complex sentence structures and formal expressions. In daily use, DeepL's translations are typically clear and fluent. It is less likely to sound like awkward literal translations compared to Google Translate. However, the only drawback is its language coverage. DeepL currently only supports about 30 languages, especially on European and a few Asian languages. Overall, DeepL delivers very high-quality translations within its supported language range, but its coverage is limited and it lacks regional colloquial support.

Language Support

As of 2025, although DeepL has expanded more languages, it still does not support Malay [40]. Therefore, DeepL is an excellent high-quality choice for English↔Chinese translation, but it cannot be used at all for Malay content.

2.3.3 Microsoft Translator

Voice Input

Microsoft Translator's app offers comprehensive support for voice input. Users able to speak directly into the microphone and translate their speech instantly over 100 languages [41] [42]. The app interface provides a dedicated voice mode. In addition, Microsoft offers a unique Multi-Device Conversation feature that supports up to 250 people joining the same conversation, each participating on their own device with real-time speech translation. While this function may be excessive for casual users, it demonstrates the robustness of the app's voice features. For ordinary users, Microsoft Translator makes bilingual conversations or group conversations easy to manage. Voice translation is fully supported, including audio playback of translated results.

Code-Mixed Input

As shown in Figure 2.3, it is similar to Google Translate and Microsoft Translator. It requires one language at a time and cannot translate sentences that mix multiple languages in a single input. However, if the voice input only contains Chinese and English, the program is still able to distinguish between them. But if Malay is mixed in, the system will be unable to identify the languages. The app can automatically detect the language of the input (voice or text), but it only identifies one language at a time. If you mix languages in a single sentence, the app will typically choose one language as the main input and treat the other as unknown or leaving it untranslated. However, Microsoft provides a solution for bilingual communication through its "Conversation Mode". Users only need to set Language A and Language B, and the app automatically detects which of the two is being spoken and translates it into the other language [42]. This mode works very effectively for turn-based code-switching, but it cannot handle in-sentence language mixing. On the other hand, the multi-device conversation feature also requires each participant to set a fixed language. In summary, Microsoft Translator has no special support for mixed-language sentences. The engine performs well in bilingual tasks but lacks the capability for simultaneous multi-language processing. Users must select one specific language at a time and making it less efficient.

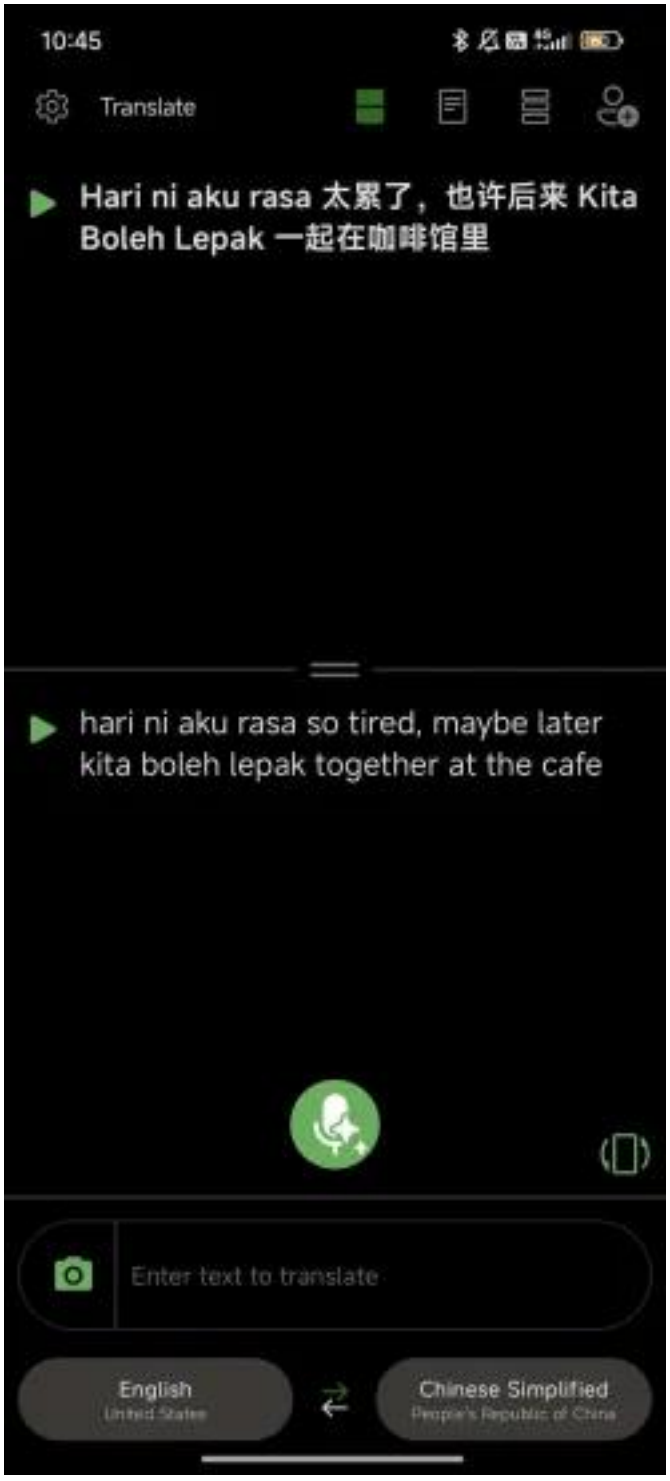


Figure 2.3: Microsoft Translator Fail to Handle Code-Mix Input

Translation Accuracy

Microsoft Translate offers reliable accuracy. In terms of quality, it is positioned between Google Translate and DeepL. It can convey the meanings of the sentences in daily communication and maintain extensive vocabulary support [34]. Evaluation results show that the application generates accurate text translations without serious errors. There are indications that the translation engine tends to generate results on a word-by-word basis. Moreover, speech recognition is also recognized. This is because the system is able to accurately capture and translate input in real-world usage scenarios. Overall, translations may require revision, but Microsoft's accuracy is sufficient for general needs.

Language Support

Microsoft Translator supports over 100 languages including Malay, Chinese and English [42]. In practice, users can freely translate between these languages. For example, Malay ↔ Chinese, Chinese ↔ English, Malay ↔ English.

2.3.4 iTranslate

Voice Input

iTranslate supports voice input, but some functions require a paid upgrade. The app highlights a "Voice Translation" mode and even released a standalone *iTranslate Voice* app. In the main app, users can tap the microphone to speak, the app will recognize the speech and translate it. According to the developer, iTranslate supports voice conversations in over 40 languages including Chinese, English and Malay. The voice chat interface is designed to be relatively simple and efficient. However, it should be noted that in the free version, some functions may be limited such as unlimited voice conversations or speech recognition for all languages. Reviews and documentation indicate that offline voice and photo translation require a subscription to the "Pro" version [34]. However, iTranslate does allow users to use voice input in major languages for general online use. It also supports text-to-speech playback. This means translations can be read aloud.

Code-Mixed Input

As shown in Figure 2.4, iTranslate does not have a dedicated function for handling code-mixed input. It supports only one language at a time. The app includes an automatic language detection feature [43], but the detection result will treat the entire sentence as a single language. Furthermore, iTranslate's voice input functionality is restricted to a premium subscription, meaning that non-paying users are limited to textual input only for translation. This creates an additional barrier for users who rely on verbal communication. For example, if input "Saya makan lunch already" (Malay + English), the app might identify it as Malay and attempt a translation, but it will likely mistranslate the English part. Currently, there is no evidence that iTranslate can intelligently parse and translate mixed-language sentences. Besides, iTranslate does not provide bilingual conversation mode like Google Translate, it is more oriented toward "one language at a time" translations. As a result, in-sentence code-switching will confuse iTranslate. In summary, iTranslate does not support code-mixed input.

Figure 2.4: iTranslate Fail to Handle Code-Mix Input

Translation Accuracy

Among commonly used translation apps, iTranslate is generally regarded as the weakest in terms of accuracy. In independent tests, its translation quality was clearly lower than Google or Microsoft. Experts pointed out that iTranslate has a relatively limited vocabulary, sometimes producing incorrect word choices or garbled grammar structures [34]. A specific technical issue is the inconsistency in handling grammatical details such as gender agreement. Hence, the output may become ambiguous or deviate from the original meaning. Due to these limitations, iTranslate is not recommended use for professional or important situations. The application can be used for daily communication, but users must verify the result manually. Furthermore, many features are restricted to the paid version. But even the Pro version is noted to lag behind free alternatives in terms of NMT quality. Overall, iTranslate is suitable for casual use remains more susceptible to producing incorrect results.

Language Support

iTranslate supports text translation for over 100 languages including Malay, Chinese and English [44]. However, there is a technical gap between its text and voice capabilities. While text translation covers the full range of supported languages, the speech recognition and speech synthesis features are limited to around 40 languages only [43]. Overall, although iTransalte covers the primary languages required for this study, but some advanced features such as offline mode and unlimited voice translation, are only accessible through a paid subscription [34]. While basic text translation for Malay, Chinese and English is available for free, but voice-based translation remains restricted in the free version.

2.3.5 Naver Papago

Voice Input

A core feature of the Papago application is its voice translation capability [45]. The system allows users to speak directly and their voice will be converted into text or spoken output in real time. The application also provides a "Conversation Mode" designed for two-way communication. In this mode, two users can select their respective languages and press the microphone button to begin conversation. It requires an internet connection or download offline language packs. In actual use, Papago performs well with voice input. For instance, a Korean speaker and an English speaker can have a voice conversation through Papago, it will translate the speech sentence by sentence. The app uses the microphone to capture speech and outputs translations with relatively natural-sounding voices. Since Papago was developed by a Korean company, its speech recognition is especially strong for Korean and adapts well to Asian accents. Overall, ordinary users can easily use Papago's mobile app for voice queries and bilingual conversations, making it very convenient for on-the-go use.

Code-Mixed Input

As shown in Figure 2.5, Papago cannot translate mixed-language sentences in one input. If multiple languages are mixed in a single sentence, Papago does not automatically identify and separately translate each part. The app does provide an automatic language detection feature which can determine which one language from its supported set the input belongs to [46]. Moreover, when utilizing voice input, the system remains unable to distinguish between multiple languages within the same audio stream. However, this detection is limited to selecting one language from its 15 supported options. For example, it can detect whether a sentence is Korean, Thai, or English, but it cannot assign different parts of one sentence to different languages. In a mixed-language voice scenario, Papago will usually recognize only one of the languages, while ignoring or misinterpreting the words from the other. Papago's conversation mode is designed for two languages, but also requires turn-taking instead simultaneous mixing. In summary, Papago does not support code-mixed input.

Figure 2.5: Papago Fail to Handle Code-Mix Input

Translation Accuracy

Papago is highly rated for its accuracy in specific language pairs, especially does well with Asian languages. It often produces more natural results for Korean, Japanese and Chinese compared to general-purpose translator produces [47]. Many Korean prefer to use it because it handles linguistic nuances more accurately such as Korean honorifics [48]. For example, Papago identifies the correct politeness levels in Korean, but other applications unable identify and produce inappropriate informal expressions. While Papago is best with short phrases and daily sentences, its accuracy decreases for European languages due to a smaller training dataset. Overall, Papago's focus on Asian linguistic contexts makes it a reliable choice for casual translation in these languages.

Language Support

Papago only supports 14 languages and Malay is not included [45]. Therefore, Papago cannot recognize and process Malay text and speech. While it supports both Chinese and English, user in Malaysian context are required to utilize alternative applications to handle Malay-related tasks. Among the three target languages identified for this study, Papago only provides support for Chinese and English.

App	Google Translate	DeepL Translator	Microsoft Translator	iTranslate	Naver Papago
Voice Input	✓	✓	✓	✓	✓
Code-Mixed Input Handling (Text)	✕	✕	✕	✕	✕
Code-Mixed Input Handling (Voice)	✕	✕	Partial (EN/CN only)	Paid	✕
Translation Accuracy	Moderate	High	Moderate	Low	Moderate
Support Malay	✓	✕	✓	✓	✕
Support Chinese	✓	✓	✓	✓	✓
Support English	✓	✓	✓	✓	✓

Table 2.3: Functional and Language Support Comparison of Existing Translation Applications

2.4 Proposed Solution

To address the limitations of current translation tools, this study proposes a solution that combines a fine-tuned Meta SeamlessM4T model with Gemini as translation engine. In this system, SeamlessM4T will first undergo fine-tuning using code-mixed audio data in order to enhance the model's ability to recognize Malay, Chinese and English. By training on mixed-language data, the system can reduce transcription errors and provide a more stable foundation for the translation stage.

The system utilizes Google Gemini as the translation engine. After speech recognition, the system passes the transcription to Google Gemini and process the text. Gemini is selected because Gemini handles code-mixed input well and reasons about context betters. Besides, the reason that Google Gemini is selected is also because it provides free tier to developers and allows the developers to use it without any cost.

This project emphasizes a structured chat flow to enable role identification. To solve the ambiguity of "who said what" problem, this project implements a segregated chat-style interface. By aligning messages position from User A and User B to opposite sides and tagging them with unique ID, the system keeps the conversation coherent and makes translation history browsable in context.

The entire system is developed as a mobile application that supports voice input and output. The application leverages cloud-based processing to perform heavy processing without exhausting mobile hardware resources. By leveraging cloud-based processing, the application provides a responsive user experience.

CHAPTER 3 – SYSTEM METHODOLOGY AND DESIGN

3.1 System Development Methodology

The project uses Agile Scrum methodology as shown in Figure 3.1. The framework is iterative and built around continuous delivery [49]. Under this methodology, the development process is divided into short and time-boxed iterations called sprints. These usually last between 1 and 2 weeks. Each sprint aims to produce a potentially deployable product increment in order to perform regular assessment and adjustments.

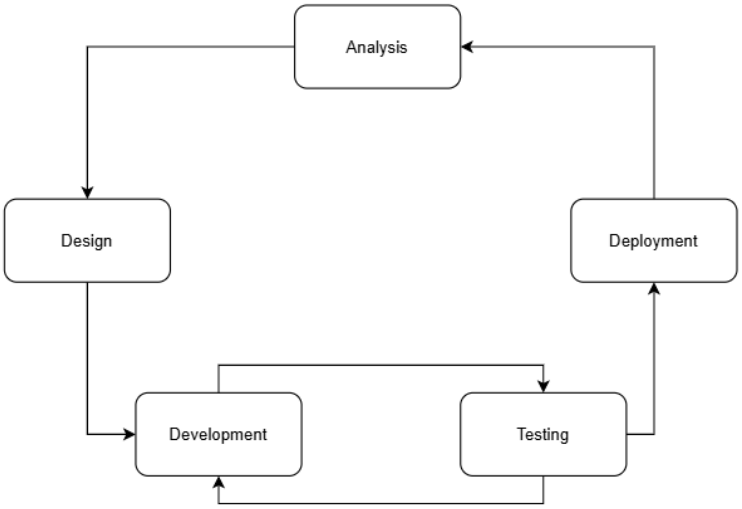


Figure 3.1: Agile Scrum Methodology Workflow

1. Analysis Phase

The analysis phase focuses on identifying the core problem and establish the project scope. During this stage, the project objectives are clearly established. It is used to guide the technical direction and prevent the project from veering of course. In this phase, the outcome is defined problem statement and a set of project objectives for the translation application.

2. Design Phase

The design phase focuses on formulating the overall system architecture and design based on the identified objectives. This process includes identifying the necessary computing resources, development environments and third-party APIs required for implementation. The feasibility of the system structure is verified before any coding begins. In this phase, the outcome is the finalized system architecture, user interface designs, use case diagrams and so on.

3. Development Phase

The development phase focuses on the actual implementation of the system based on the design specifications. This stage involves coding the mobile interface, establishing database connectivity, integrating the translation engine & ASR and so on. The application is built in functional increments through an iterative process. In this phase, the outcome is a functional version of the multilingual translation application with integrated API modules and a complete codebase.

4. Testing Phase

The testing phase focuses on evaluates the performance and accuracy of the system. This process specifically measures the ASR accuracy, the translation precision, noise testing and so on. In this phase, the outcome is a system evaluation report that documenting the performance and translation accuracy of the application.

5. Deployment Phase

The deployment phase focuses on installing the completed application on physical mobile devices. The objective is to verify its performance under usage conditions. In this phase, the outcome is the successful installation of the application on different mobile devices and the verification of its functions during real-world use.

3.2 System Requirements

3.2.1 Hardware Requirements

The project runs on two devices which is desktop and an Android mobile device. The detailed specification are listed in Table 3.1 and Table 3.2

The desktop serves as the main development environment. It is used to handle the implementation of both fronted and backend modules. It is configured with high-performance CPU and sufficient RAM to handle resource-heavy tasks. For example, integrated development environments (IDE), fine-tuned ASR model, backend services and so on.

The Android mobile device is used to handle testing and deployment in a live environment. This stage is necessary to verify the system's performance, especially the interface responsiveness and the voice translation accuracy.

Components	**Specifications**
Processor	AMD Ryzen 7 5700X 8-Core Processor
Operating System	Windows 11
Graphic	NVIDIA GeForce RTX 5070
Memory	32GB DDR4 RAM
Storage	Kingston NV3 1TB M.2 NVMe SSD

Table 3.1: Specifications of Desktop

Components	**Specifications**
Model	POCO F7 Pro
Processor	Qualcomm Snapdragon 8 + Gen 3
Operating System	Android 15, HyperOS 3
Graphic	Adreno 750
Memory	12 GB
Storage	256 GB internal storage

Table 3.2: Specifications of Android Mobile Device

3.2.2 Software Requirements

The project uses several platforms and tools for development, environment management and integrating services.

Anaconda handles environment management and dependency isolation. It creates a isolated sandbox for the project and ensures that complex libraries do not conflict with system-level packages. This maintains the stability of the experimental environment across different machines.

Visual Studio is the primary IDE. This is because its lightweight nature and its support for Python and JavaScript. Key features also provided such as the integrated terminal, Git support and intelligent code completion. These features help to manage both frontend and backend codebases efficiently.

Lastly, React Native is the core framework used for mobile application development. It enables the creation of a responsive user interface for Android. For backend and cloud services, Firebase is integrated to handle user authentication and store conversation history. Furthermore, the Google Cloud Console is used to mange and integrate the Gemini API and the Google Cloud Text-to-Speech (TTS) service into the application

3.3 User Interface Design

The user interface is designed to facilitate efficient multilingual communication through a structed and intuitive layout. Figure 3.2 shows the Home Page which serves as the central hub of the system. The “Create a New Chat” button that shown in the interface is to initiate a new translation session. A drawer menu icon that located at the top-left corner is used to show/hide the sidebar. The sidebar allow users to create new translation session, review the past conversation history and accessing account settings.

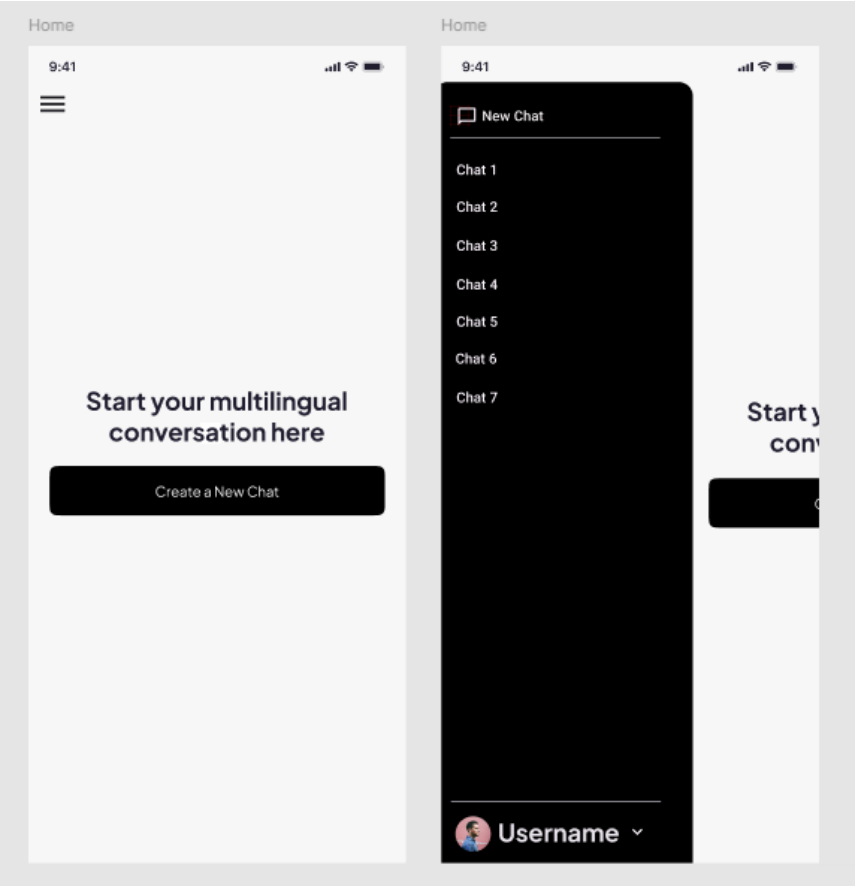


Figure 3.2: Home Page UI Design

Figure 3.3 shows the profile page of the application. It provides users with management capabilities for their personal accounts. This page displays user information, including profile picture, username and email address. Besides, interactive elements allow users to update their profile picture or username via “Edit Profile” button. A “Logout” button is also provided to allow user log out the account or switch account.

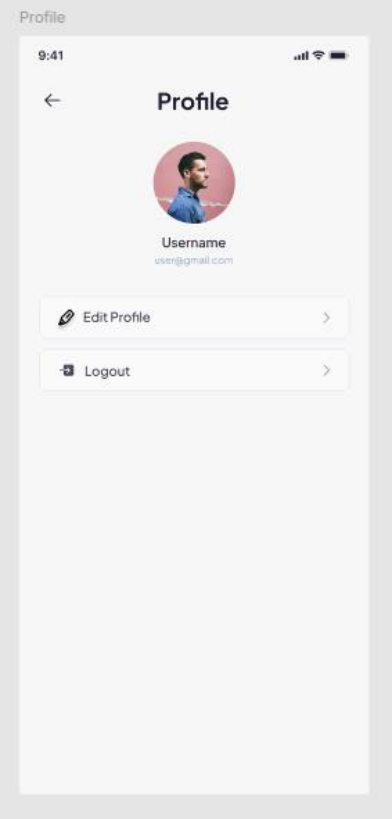


Figure 3.3: Profile Page UI Design

Figure 3.4 shows the Chat Page of the application. Upon starting a new conversation, the system prompts users to select the type of conversation room and the source and target languages for both participants. The interface utilizes a segregated chat-style layout where the speaker 1's messages are aligned to the left (orange) and the speaker 2's messages are aligned to the right (green). This visual layout helps with role identification and improve the readability of the conversation history. By separating the messages, users can easily refer back to previous parts of the dialogue. Besides, a "more" menu that located at the top-right corner provides management options such as renaming or deleting the chat room.

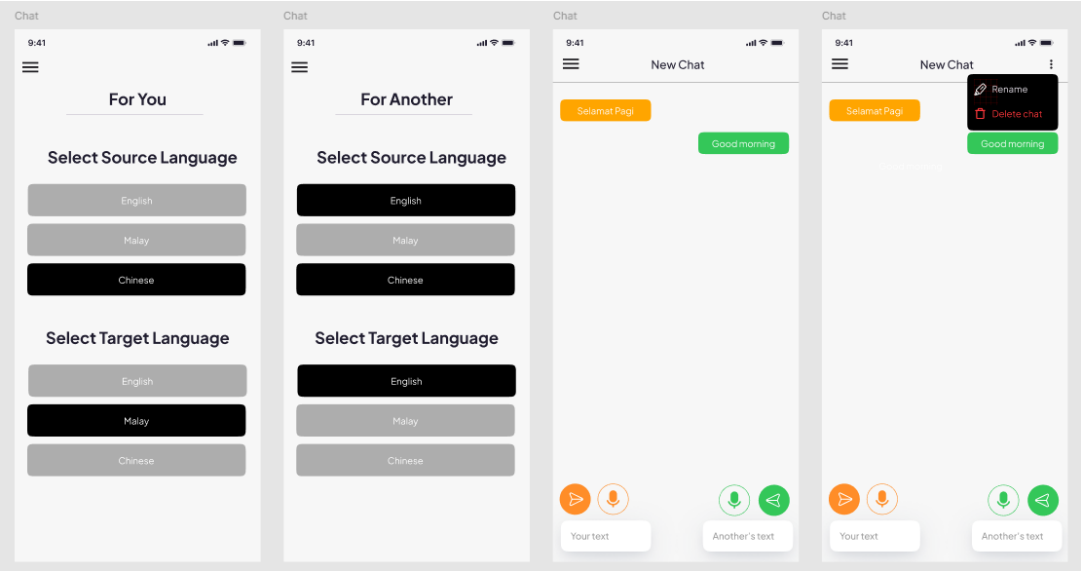


Figure 3.4: Chat Page UI Design

As shown in Figure 3.5, the Login Page is the main interface. Registered users need to enter their credentials to access the application. For new users, they need to create a new account before access the application. The interface also includes a “Reset Password” feature to assist users find back their account. Besides, the application supports third-party authentication which is Google Account login. It provides a more convenient and faster sign-in process for users.

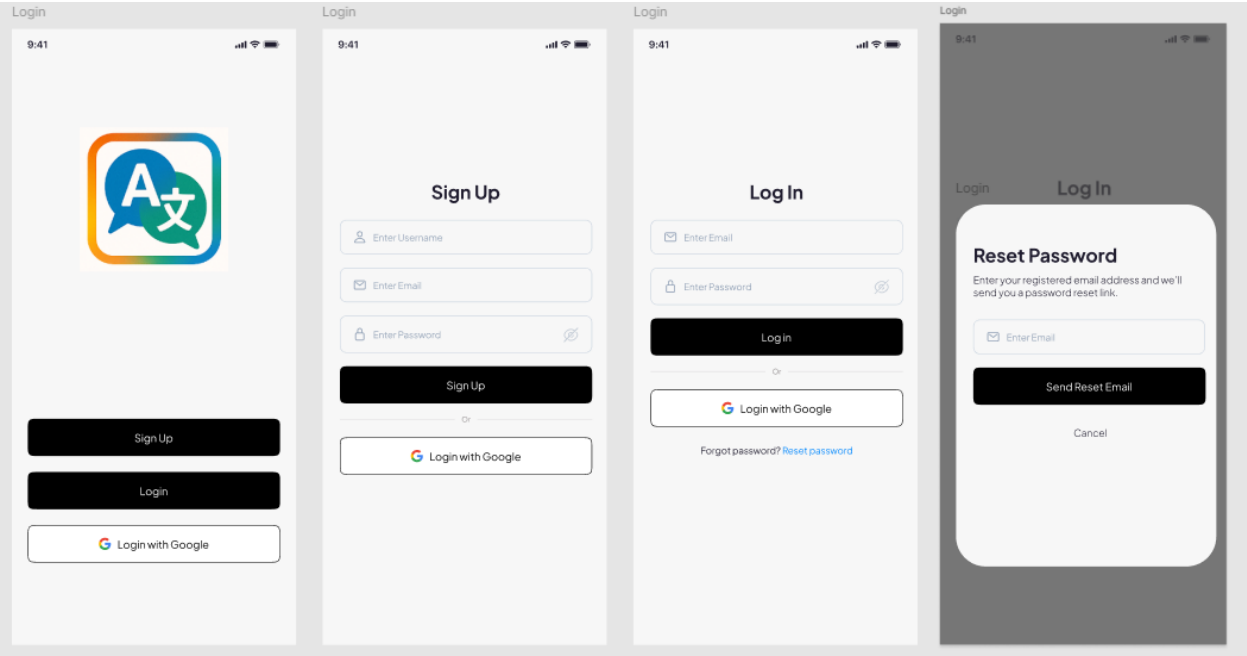


Figure 3.5: Login Page UI Design

3.4 Use Case Diagram & Description

Figure 3.6 illustrates the use case for the system. It outlines the interactions between the user and the application. The diagram is categorizes the system's functionalities into four main modules which is account management, profile management, translation services and chat history management.

Account management consists of several core processes including account registration, authentication, password recovery and logging out. During registration, the system will implement an email verification process. It helps to ensure the validity of user data. For authentication, users are required to enter their credentials to access the application. In the profile management section, users can view their current account information and update details such as username, profile picture and email account.

The translation services allow users to process inputs by texting or voice. These functions are extended by a voice playback feature that allows users to receive the output with audio. Furthermore, the chat management module enables users to review their conversation history, rename chat, delete records and acquire the account information of another speaker. Overall, these use cases represent the core interactions and features in the applications.

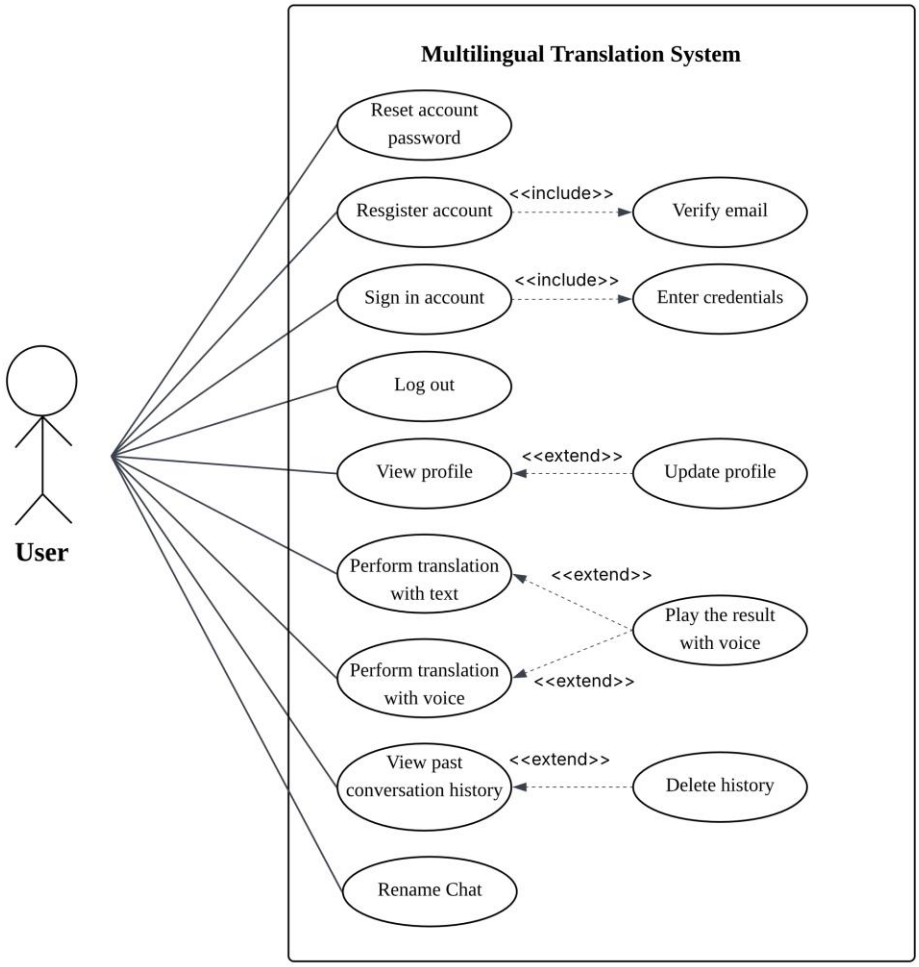


Figure 3.6 Use Case Diagram

System	Multilingual Translation Mobile Application
Use Case Name	Register Account
ID	1
Description	The user fills in registration details to create a new account. During registration, the system includes an email verification step to ensure account authenticity
Primary Actor	User
Trigger	The user selects the “Sign Up” option from the login page
Precondition	The user has not registered
Normal Flow of Event	1. The user taps the “Sign Up” button. 2. The system displays the registration form. 3. The user enters an usertname, email and password. 4. The system validates the entered data. 5. The system sends a OTP to the provided address. 6. The user receive the OTP from email and enter the OTP. 7. The system confirms verification and creates the account successfully.
Sub Flow	S1: Validate Input Data 1. The system checks whether the email format is correct S2: Store User Information 1. The system saves user credentials into the database
Alternate /Exceptional Flow	A1: Invalid email format 1. System displays an error message: “Please enter a valid email address.” A2: Password field is empty 1. System displays: “Password cannot be empty.” A3: Verification email not received

	1. System displays a message: “Didn’t receive the email? Click to resend.”

Table 3.3: Register Account Use Case Description

System	Multilingual Translation Mobile Application
Use Case Name	Log In Account
ID	2
Description	The user logs into the application using valid credentials. If the user forgets the password, the use case can extend to the “Reset Password” process.
Primary Actor	User
Trigger	The user selects the “Log In” option from the login page
Precondition	The user has not logged in
Normal Flow of Event	1. The user taps the “Login” button. 2. The system displays the login form. 3. The user enters a valid email address and password. 4. The system validates the entered credentials against the database. 5. If the credentials are correct, the system authenticates the user. 6. The system redirects the user to the main chat interface of the application.
Sub Flow	S1: Authenticate credentials 1. The system compares the entered credentials with stored user data.
Alternate/Exceptional Flow	A1: Incorrect email or password 1. System displays: “Incorrect email or password. Please try again.” A2: Empty fields 1. System displays: “Email and password fields cannot be empty.” A3: Forgot password

	1. The user taps “Reset Password” and the use case extends to Reset Password.

Table 3.4: Log In Account Use Case Description

System	Multilingual Translation Mobile Application
Use Case Name	Reset Password
ID	3
Description	The user resets the account password if forgotten. The system sends a password reset link to the registered email address.
Primary Actor	User
Trigger	The user selects “Forgot Password” from the login screen.
Precondition	
Normal Flow of Event	1. The user taps “Reset Password.” 2. The system prompts for the registered email address. 3. The user enters their email and request OTP. 4. The system sends a OTP to that email. 5. The user enter OTP and reset a new password. 6. The system updates the password in the database.
Sub Flow	-
Alternate/Exceptional Flow	A1: Invalid email format 1. System display: “Invalid email format.” A2: Unregistered email 1. System display: “Email not found. ”

Table 3.5: Reset Password Use Case Description

System	Multilingual Translation Mobile Application
Use Case Name	Perform Translation
ID	4
Description	The user selects source and target languages, then provides input by text or voice, the translation engine generates translated text or voice output.
Primary Actor	User
Trigger	The user enters text or records voice in the chat interface.
Precondition	The user is logged in and has an active session
Normal Flow of Event	1. The user opens a new chat. 2. The system displays the type of conversation room. 3. The user selects the room. 4. The system displays available language options. 5. The user selects the source language and target language. 6. The system confirms the selected languages. 7. The user chooses an input method (text or voice) 8. If text is selected, the user types a message in the input field. 9. If voice is selected, the user records speech through the microphone. 10. The original text or transcribed text is forwarded to the translation engine 11. The translated output is displayed in text form. 12. The user can optionally request voice output.
Sub Flow	S1: Generate Output 1. The translated text is returned to the application. If voice output is requested, TTS converts it into speech.
Alternate/Exceptional Flow	-

Table 3.6: Perform Translation Use Case Description

System	Multilingual Translation Mobile Application
Use Case Name	View Past Conversation History
ID	5
Description	The user views past translation sessions
Primary Actor	User
Trigger	The user selects the chat that want to view.
Precondition	The user is logged in and has existing chat history.
Normal Flow of Event	1. The user select the chat 2. The system retrieves and displays previous translations.
Sub Flow	S1: Load Chat Content 1. The system retrieves all messages under the selected chat.
Alternate/Exceptional Flow	A1: Database or network error 1. System displays: “Unable to load chat history. Please try again later.”

Table 3.7: View Past Conversation History Use Case Description

System	Multilingual Translation Mobile Application
Use Case Name	Delete Chat
ID	6
Description	The user deletes an entire chat session. The system permanently removes the selected chat after confirmation.
Primary Actor	User
Trigger	The user taps the red trash bin icon at the top-right corner of the chat
Precondition	The user is logged in and has at least one saved chat in history
Normal Flow of Event	1. The user navigates to the Chat History section. 2. The system displays all previously saved chat sessions. 3. The user selects a specific chat. 4. The user taps the trash bin icon to delete the selected chat. 5. The system displays a confirmation message. 6. The user confirms the deletion. 7. The system removes the chat record permanently from the database. 8. The system displays a success message.
Sub Flow	S1: Database Update 1. Once confirmed to delete, the chat record is removed from the database.
Alternate/Exceptional Flow	A1: User cancels the confirmation 1. The deletion process stops and the chat remains in history.

Table 3.8: Delete Chat Use Case Description

System	Multilingual Translation Mobile Application
Use Case Name	View Profile
ID	7
Description	The user views personal account information
Primary Actor	User
Trigger	The user taps profile avatar on the top-right corner
Precondition	The user is logged in
Normal Flow of Event	1. The user taps the profile avatar. 2. The system retrieves the user's profile data from the database. 3. The system displays the user's information, including avatar, name and email. 4. The user reviews their profile information.
Sub Flow	S1: Data Retrieval 1. The system fetches stored profile details from the database.
Alternate/Exceptional Flow	-

Table 3.9: View Profile Use Case Description

System	Multilingual Translation Mobile Application
Use Case Name	Update Profile Avatar
ID	8
Description	The user updates profile avatar
Primary Actor	User
Trigger	The user taps the profile avatar to change the picture
Precondition	The user is logged in and currently viewing their profile page.
Normal Flow of Event	1. The user taps the avatar to update the profile picture. 2. The system opens the image picker 3. The user selects or captures a new photo. 4. The system uploads and updates the avatar. 5. The profile page refreshes to show the updated information.
Sub Flow	S1: Image Upload Process 1. The system allows the user to choose an image and uploads it to the server.
Alternate/Exceptional Flow	A1: User cancels image selection 1. The system closes the image picker without making changes.

Table 3.10: Update Profile Avatar Use Case Description

System	Multilingual Translation Mobile Application
Use Case Name	Update Profile Name
ID	9
Description	The user updates profile name
Primary Actor	User
Trigger	The user taps the pencil icon next to the name to edit it.
Precondition	The user updates profile avatar
Normal Flow of Event	1. The user taps the pencil icon beside the display name. 2. The system displays a text field for name editing. 3. The user enters the new name and confirms. 4. The system updates and saves the new name. 5. The profile page refreshes to show the updated information.
Sub Flow	S1: Name Update Process 1. The system validates and saves the new display name.
Alternate/Exceptional Flow	A1: Empty name field 1. The system displays: “Name cannot be empty.”

Table 3.11: Update Profile Name Use Case Description

System	Multilingual Translation Mobile Application
Use Case Name	Logout
ID	10
Description	The user logs out of the application
Primary Actor	User
Trigger	The user taps “Logout” button.
Precondition	The user is logged in and currently viewing their profile page.
Normal Flow of Event	1. The user selects the logout option. 2. The system terminates the current session. 3. The user is redirected to the login screen.
Sub Flow	-
Alternate/Exceptional Flow	-

Table 3.12: Logout Use Case Description

System	Multilingual Translation Mobile Application
Use Case Name	Rename Chat
ID	10
Description	The user renames an existing chat session. The system updates and saves the new chat name in the database.
Primary Actor	User
Trigger	The user taps the “Rename Chat” button at the top-right corner of the chat
Precondition	The user is logged in and has at least one existing chat session saved in history.
Normal Flow of Event	1. The user selects a specific chat. 2. The user taps the “Rename Chat” button. 3. The system displays a text input field for entering the new chat name. 4. The user types a new chat name and confirms. 5. The system validates and saves the new name in the database. 6. The chat list refreshes to display the updated name.
Sub Flow	S1: Update Chat Name in Database 1. The system validates the new name, updates it in the database and refreshes the user interface.
Alternate/Exceptional Flow	A1: Empty name field 1. System displays: “Chat name cannot be empty.”

Table 3.13: Rename Chat Use Case Description

3.5 System Flowchart

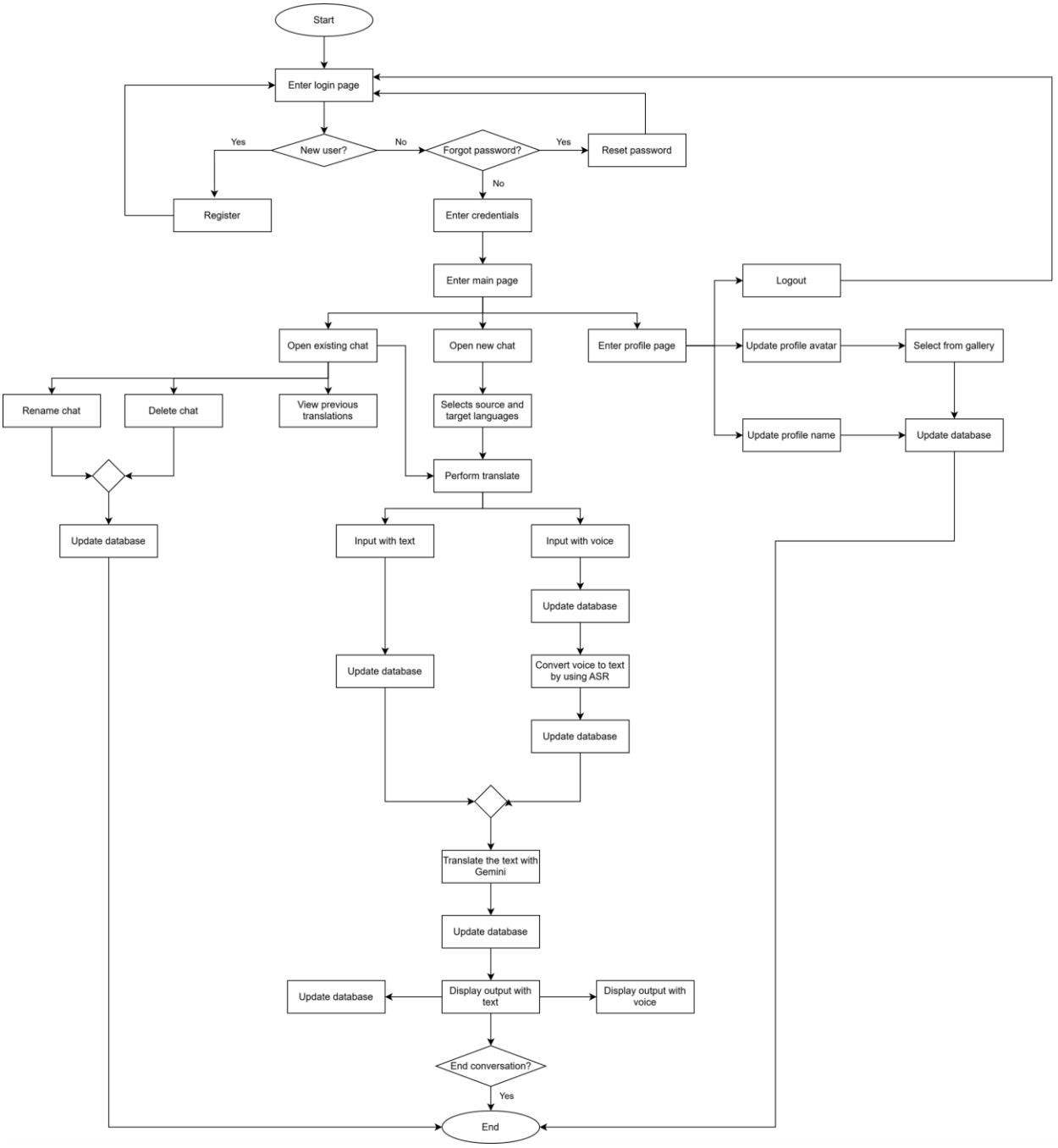


Figure 3.7: Overall flowchart for mobile application

3.6 Framework and Technologies

3.6.1 Development Framework

The application is developed by using React Native. It is a robust framework for building cross-platform mobile applications. React Native was selected because of its extensive ecosystem. It helps to handles the heavy lifting for integrating external APIs and libraries. This framework also makes the UI development smoother and keeping the codebase maangeable.

3.6.2 Backend and Data Management

Firebase handles authentication and data storage. All translation records and chat histories are stored in a structured text format within Firebase. It enables users to review past conversation across sessions. It also allows for the secure management of API keys. It helps to ensure real-time synchronization and effectively reducing the need for extensive server-side infrastructure.

3.6.3 Speech Recognition (ASR)

The speech recognition component is implemented using a fine-tuned Meta SeamlessM4t model that built for multilingual audio in Malay, Chinese and English. Unlike the conventional ASR that are typically limited to monolingual inputs, fine-tuned SeamlessM4t model able to identify language boundaries and processing mixed-language input. This enhancement reduces transcription errors and provides higher-quality text input for the translation engine.

3.6.4 Translation Engine

In this project, Google Gemini API is selected as translation engine. Gemini was selected due to its advanced contextual reasoning and its strong performance in handling code-mixed linguistic patterns. These capabilities are essential for the multilingual nature of this project. It ensures the translations remain semantically accurate.

3.6.5 Text-to-Speech (TTS)

To provide audio output, the application also integrates Google Could Text-to-Speech. This service generates high-quality and natural sounding voice outputs in multiple languages. It will adapt its pitch to the selected language by users and ensures the voice sounds alive instead of robotic. Besides, by providing audio output, it enhances the convenience and effectiveness of users during conversations.

3.7 System Architecture and Workflow

The system adapts a Client-Server Architecture to optimize performance and handle the computational demand of the integrated AI models. By using this architecture, the mobile application serves as a client for user interaction. Backend server hosts the fine-tuned SeamlessM4T model for ASR processing. Translation and voice output are processed via cloud-based APIs.

As shown in Figure 4.8, the workflow of the system is structured into 4 primary stages:

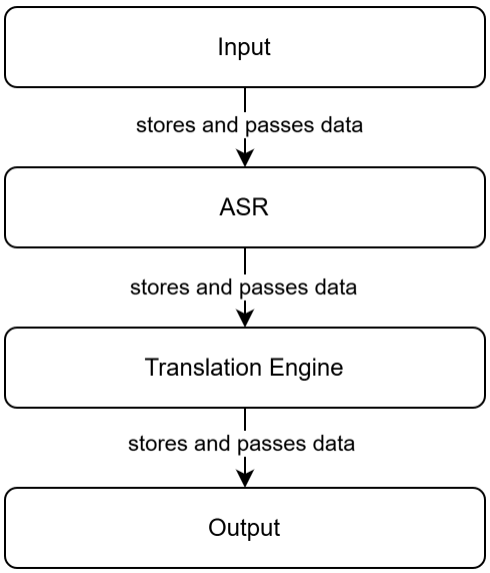


Figure 3.8: Workflow of the system

1. Input Phase:

The mobile application captures audio or text input from the users. This data is initially saved to the database and subsequently forwarded to backend server for further processing.

2. ASR Processing:

The server processes the audio input by using the fine-tuned SeamlessM4T model. It transcribes the speech into text accurately. It will identify language boundaries in code-mixed utterances. After that, the transcribed text will be updated to database.

3. Translation Phase:

The system retrieves the transcribed text from the database and passes it to the Google Gemini Api. The translation engine utilizes contextual reasoning and semantic understanding to generate fluent translations. It ensures accuracy even in complex code-mixed scenarios. Once the translation is completed, the result will be updated to database.

4. Output Phase:

The translated results display on the mobile app. Conversation logs also will be saved to the database and allow users to review history whenever they want. The translated text can be converted into audio through the Text-to-Speech (TTS) feature.

CHAPTER 4 – SYSTEM IMPLEMENTATION

4.1 Fine-tuning ASR Engine

This section covers the fine-tuning process for SeamlessM4T-v2-large model. The objective of fine-tuning is to improve the model's ability to recognize and transcribe code-mixed speech. The implementation process is divided into four stages which is environment setup, data pipeline construction, model configuration and training execution.

4.1.1 Development Environment Setup

The development environment was set up by using Anaconda to keep dependencies isolated and the system stable. Since the project runs on a NVIDIA RTX 5070, some extra configuration was needed to keep things compatible. A virtual environment named "cs_asr" was created with Python 3.10 to serve as a sandbox for model training.

During the setup, it was identified that standard PyTorch did not support the CUDA 12.8 drivers. To address this problem, the fine-tuning process had to switch to PyTorch. Without this specific versioning, the GPU backend would fail to initialize and the model could not accelerate. Within this isolated environment, essential libraries like *transformers*, *accelerate*, *soundfile* were installed to handle training and data processing.

4.1.2 Data Pipeline and Windows Compatibility

A custom data processing pipeline was developed to address both the linguistic requirements and specific operating system constraints. The training data came from merging two datasets. Specifically, the ASCEND dataset (Mandarin_English audio) and Malay-Speech-3k dataset were merged together using the *concatenate_datesets* function. This approach forces the model to learn language boundaries and transitions during training. It is more effective for code-mixed scenarios than processing each language as a separate task.

However, there is a technical challenge during the data preparation phase. The challenge is about the compatibility issue with the Windows operating system. The standard HuggingFace

Audio (decode=True) method failed because the required *torchcodec* backend was not fully supported in the local Windows environment. To address this problem, a manual decoding logic was implemented which is illustrated in Figure 4.1. The dataset was loaded with automatic decoding disabled to preserve the raw byte stream. A custom preprocessing function was develop to read these bytes using *io.BytesIO* and the *soundfile* library. Furthermore, all audio inputs were standardized by resampling them to 16kHz and converting them to a mono-channel format in order to match the pre-training specifications of the SeamlessM4T model.

```
def preprocess_batch(batch):
    audio_dicts = batch["audio"]
    texts = batch[text_key]

    audio_arrays = []
    clean_texts = []

    for audio_dict, txt in zip(audio_dicts, texts):
        with io.BytesIO(audio_dict["bytes"]) as f:
            wav, sr = sf.read(f)

        if wav.ndim > 1:
            wav = wav[:, 0]

        if sr != TARGET_SR:
            wav = librosa.resample(
                wav.astype(np.float32),
                orig_sr=sr,
                target_sr=TARGET_SR
            )
        else:
            wav = wav.astype(np.float32)

        audio_arrays.append(wav)
        clean_texts.append(str(txt).strip())

    batch["input_values"] = audio_arrays
    batch["labels"] = clean_texts
    return batch
```

Figure 4.1: Implementation of Custom Audio Preprocessing

4.1.3 Model Configuration and Data Collation

After the data is prepared, the SeamlessM4T-v2-leage model was configured to adapt from its default Speech-to-Text Translation (S2TT) mode to ASR mode. This transition involved configuring the processor to treat the input audio and output text as part of the same language family. By focusing strictly on speech recognition, the system ensures that the fine-tuning process is targeted at improving transcription accuracy instead of translation performance.

To handle the variable-length audio inputs inherent in speech data, a custom *DataCollator* was implemented to mange batching. As shown in Figure 4.2, collator is the masking of padding tokens in the text labels with the value -100. This configuration is necessary because the Cross-Entropy loss function used during training is designed to ignore tokens with this specific value during backpropagation. This ensures that gradient updates are focused solely on meaningful text tokens. By using this approach, the training efficiency and model convergence can be improved.

```
pad_token_id = tokenizer.pad_token_id
labels[labels == pad_token_id] = -100

inputs["labels"] = labels
return inputs
```

Figure 4.2: Implementation of Label Padding Masking within the Custom DataCollator

4.1.4 Fine-Tuning Execution

Lastly, the final stage is execute the fine-tuning process by using the *Seq2SeqTrainer*. Since the VRAM is limited, some hyperparameters are optimized during training. The hyperparameters details was shown in Figure 4.3 below. The training duration was set to 5 epochs. Although preliminary tests with a single epoch showed a noticeable improvement over the baseline model, but extending the training duration to 5 epochs yielded superior convergence and transcription accuracy.

To manage memory usage, the batch size was set to 1 per device. To compensate for this, gradient accumulation was implemented to simulate a larger effective batch size. Gradient checkpointing was enabled to trade computational speed for VRAM space. *Adafactor* replaced

AdamW as the optimizer because it uses less memory. It is critical when training large models on limited hardware. This setup successfully produced specialized model weights capable of transcribing mixed-code audio with high precision.

```
# ---- 3.9 TrainingArguments ----
training_args = Seq2SeqTrainingArguments(
    output_dir="./seamlessm4t_cs_zh_en_ms_e5",
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    gradient_accumulation_steps=1,
    num_train_epochs=1,
    learning_rate=1e-5,
    warmup_ratio=0.2,
    logging_steps=50,
    eval_steps=200,
    evaluation_strategy="steps",
    save_steps=500,
    save_total_limit=2,
    fp16=False,
    bf16=False,
    optim="adafactor",
    gradient_checkpointing=True,
    remove_unused_columns=False,
)
```

Figure 4.3: Configuration of Training Hyperparameters

4.2 Real-time Data Management

The application uses Firebase Cloud Firestore as a NoSQL database for an event-driven architecture. By using Firestore as a Single Source of Truth (SSOT), the application avoids the complexity of traditional state management libraries and the need for constant HTTP polling. The frontend subscribes directly to database updates by using listeners instead of manually requesting data. By using this approach, it ensures high responsiveness and allows the application to handle asynchronous. For example, system allows users continue using the application without waiting for a translation response from the cloud

4.2.1 Cross-Screen State Synchronization

To improve code maintainability, the application decouples data logic from the UI components. When the user metadata or conversation states are updated, these changes are committed directly to the corresponding Firestore document. Firestore SDK will detects these changes automatically and triggers the *onSnapshot* method to update all client-side listeners which is shown in Figure 4.4. This mechanism is particularly useful for updating independent UI component such as the side drawer or the user profile page. Instead of manual “state lifting” to pass data, these screens re-render automatically whenever the underlying database changes. This maintained the data integrity across the entire application even when multiple screens are open at once.

```
const unsubscribe = onSnapshot(q, (snapshot) => {
    const chats = snapshot.docs.map(doc => ({
        id: doc.id,
        ...doc.data()
    }))
        .filter(chat => !chat.deletedAt)
        .sort((a, b) => {
            // Sort by createdAt descending in JS (no Firestore composite index needed)
            const aTime = a.createdAt?.seconds ?? 0;
            const bTime = b.createdAt?.seconds ?? 0;
            return bTime - aTime;
        });
    setChatHistory(chats);
}, (error) => {
    console.error('[Firestore] Chat query error:', error.message);
});
return () => unsubscribe();
```

Figure 4.4: Implementing listener in the side drawer

4.2.2 The Database as a Reactive Processing Hub

Firestore's real-time capability serves as the core infrastructure for the application's translation features. It prevent the blocking data flow between the UI and the backend. The backend handles the complex AI workflow by first running the SeamlessM4T to do the transcription and then the Gemini perform the translation. In order to keep everything streamlined, it writes the final output straight back to the corresponding Firestore documents.

This mechanism effectively separate heavy computational logic from the mobile client. It allows the device to remain responsive and the server processes complex data asynchronously. Hence, once the AI workflow is complete, the mobile app receives the final translation instantly via its listeners which is shown in Figure 4.5.

```
if source_langs and target_lang:
    translation = run_translation(text, source_langs, target_lang)
    if translation:
        msg_doc.reference.update({'translation': translation})
        print(f"    [+] Text Message Translated: {translation}")
```

Figure 4.5: Backend worker updating Firestore to trigger reactive UI update

4.3 User Interface Design Implementation

The presentation layer of the application was developed by using React Native. It ensures a responsive and consistent user experience. The user interface architecture adopts a modular design to enhance maintainability of the application. By using this approach, the system remains responsive and can be easily scaled. This methodology ensures that the application layout maintains consistency across various mobile devices with different screen density and aspect ratio.

4.3.1 Component-Driven Architecture and Styling Modularization

To avoid monolithic code, interface elements were decoupled into standalone components. For instance, the conversation interface is managed through a reusable MessageBubble.js component instead of hardcoding everything into ChatScreen.js. As illustrated in Figure 4.6, this component dynamically adjusts its layout alignment and colour scheme based on the props received. This modular approach also keeps sub-elements manageable. For example, localized text strings and the audio duration indicator for voice message.

Separation of concerns is maintained through styling modularization. As shown in Figure 4.7, inline styles are replaced with dedicated styling modules such as CustomDrawerContentStyles.js and SidebarContainerStyles.js. By using this approach, it centrally manages the visual layout and colour palette. This ensures that the UI styling independent of business logic and making the codebase easier to maintain and update.

```
const containerStyle = isPrimary ? styles.leftContainer : styles.rightContainer;
const bubbleStyle = isPrimary ? styles.orangeBubble : styles.greenBubble;
const textColor = '■#FFF';

return (
    <View style={[styles.container, containerStyle]}>
        <View style={styles.messageRow}>
            {isSecondary && renderTtsButton()}
            <View style={[styles.bubble, bubbleStyle]}>
```

Figure 4.6: Dynamic style control within the MessageBubble.js component

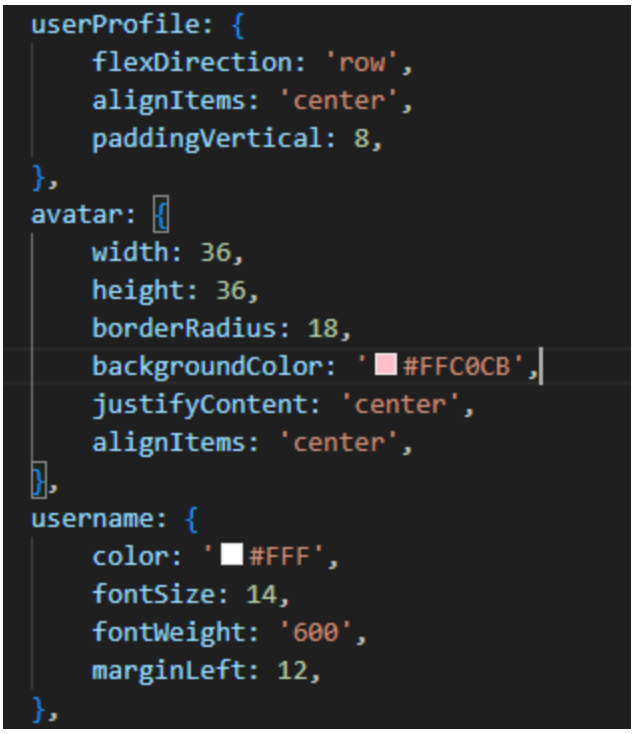


Figure 4.7: Examples of styling modularization

4.3.2 User Guideline and Feedback Mechanisms

The application also implements feedback mechanism to make the interaction flow smoother. To assist new users, the system includes “Coach-mark” tutorials that provide contextual guidance for complex features as shown in Figure 4.8. These walkthroughs help users understand the system’s functionality. Custom success modals were developed to provide clear confirmation when user successfully perform somethings. For example, custom success modal will pop out when users successfully updated their profile. These integrated micro-interactions help reduce the learning curve of the application.

```
{/* Step 1 Coach Mark: point to Create a New Chat button */}
{tourStep === 1 && (
    <CoachMark
        step="Step 1 of 4"
        arrowDir="up"
        text={"👆 Tap \"Create a New Chat\" to start your first c
        onSkip={skipTour}
    />
)}
```

Figure 4.8: Conditional invocation of Coach-marks for fluid onboarding

4.4 Final Application Overview

4.4.1 Sign up and Login Module Implementation

Figure 4.9 shows the Sign up and Login screen, which is the authentication entry point for the application. To improve user convenience, the application provides 2 methods to access into application. First method is standard email-based registration and second method is third-party authentication via Google Sign-In.

For the Sign Up functionality (Figure 4.10), the interface provides 3 input fields which is "Username", "E-mail" and "Password". The application utilizes React Native *useState* hooks to capture and manage the string values that entered by the user. When the user triggers the registration process, the system performs a client-side validation check to ensure all fields are filled. After that, the system executes the Firebase *createUserWithEmailAndPassword* API to register a new account.

The Login screen allows users to access the app using their registered "E-mail" and "Password" (Figure 4.11). This is handled by the *signInWithEmailAndPassword* function from the Firebase Auth library. Furthermore, Google Sign-In option is provided by using the *@react-native-gogle-signin/google-signin* library. When user chooses this method, the application will requests an ID token from the user's Google account and pass this credential to Firebase through *GoogleAuthProvider.credential* to complete the authentication proves without a manual password.

Besides, if users provide incorrect credentials or leave fields empty, the system triggers a *CustomAlert* component to display error messages to users. Next, the application also provides "Forgot Password" functionality to allow users find back their accounts.

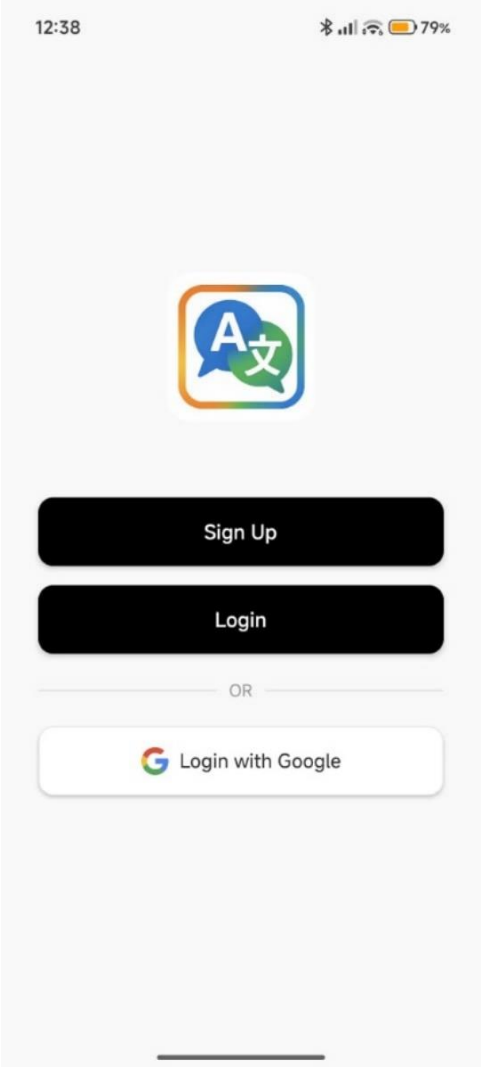


Figure 4.9: Application Entry Interface

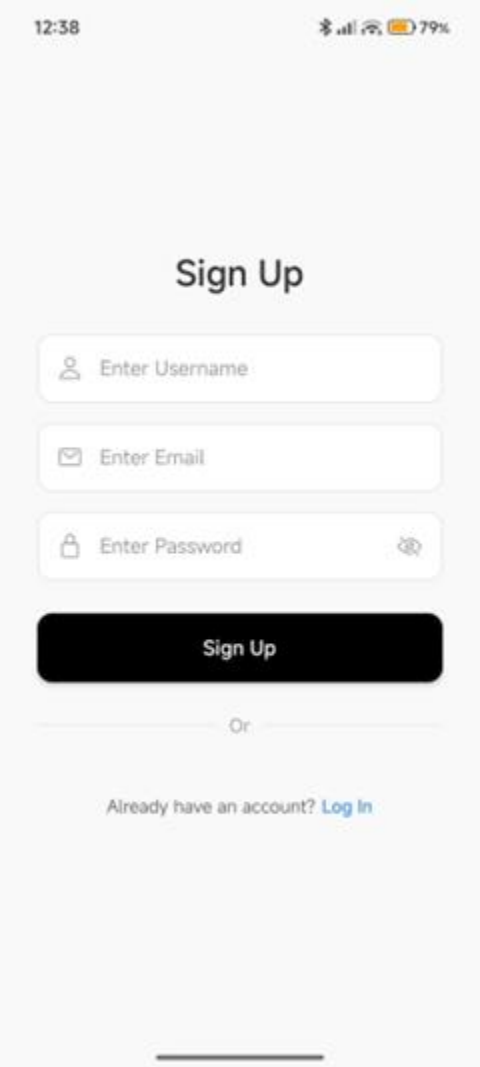


Figure 4.10: Sign Up Interface

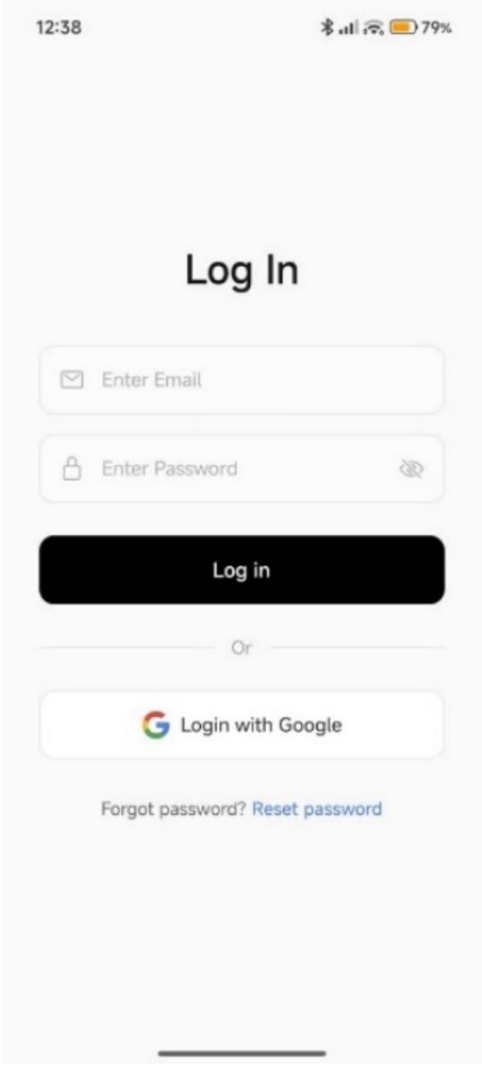


Figure 4.11: Login Interface

4.4.2 Interactive Application Tutorial Module Implementation

The Application Tutorial module is implemented as an interactive *CoachMark* system. It is designed to provided guidance to new users. Unlike traditional static sliders, this module utilizes a transparent overlay mechanism to highlight specific functional components. This approach ensures that users can learn the usage of application while interacting with the actual UI elements.

Technically, the tutorial consists of a five-step walkthrough process (Figure 4.12 to Figure 4.16). Each step renders a floating bubble (*CoachMark*) positioned relative to the target UI element. Each bubble contains a short instruction and a “Next” button. Besides, a “Skip” button is provided to allow users exit the tutorial at any state.

To ensure that the tutorial is only displayed to new users, the application integrates with Firebase Firestore. Upon the completion of the final step or when the skip option is selected, the system will trigger an update a completion flag to the user’s profile in the database.

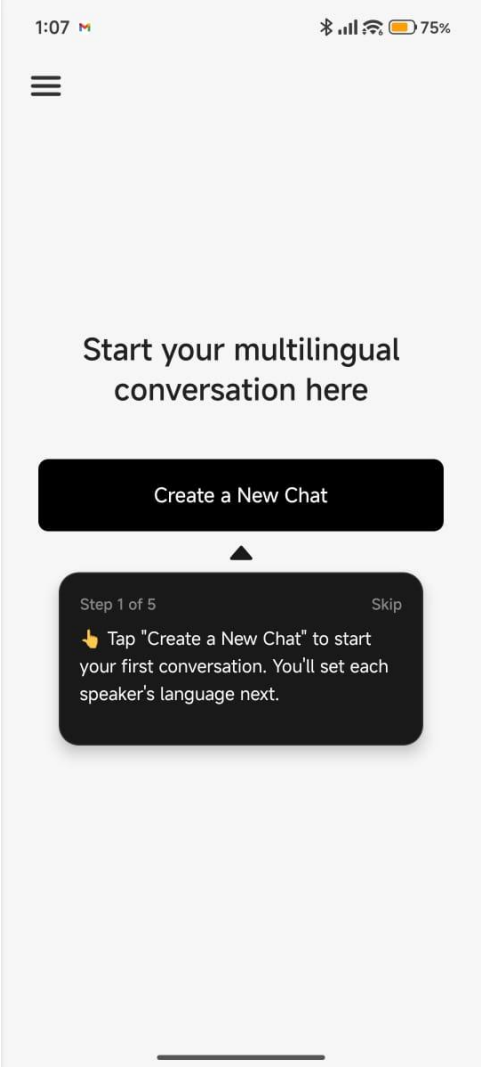


Figure 4.12: Tutorial Step 1 – Create New Chat

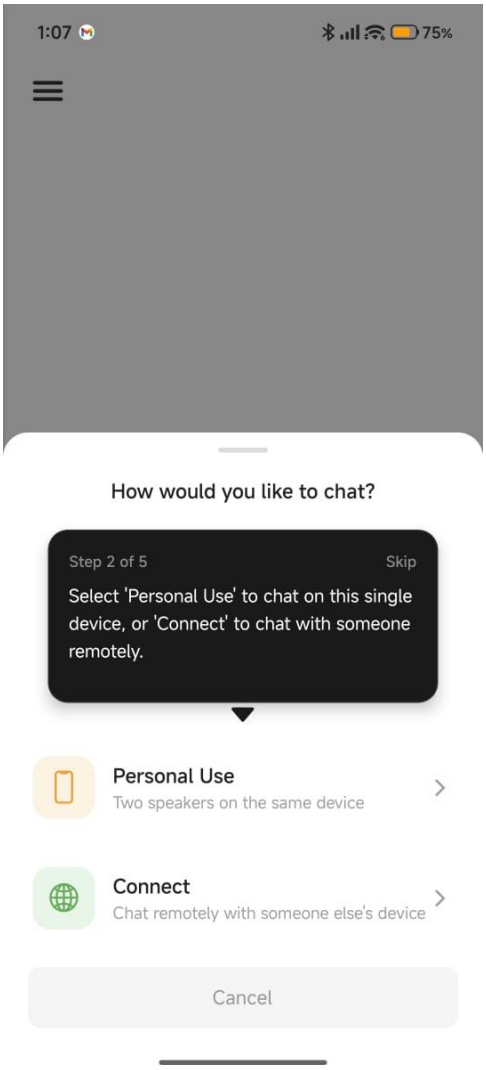


Figure 4.13: Tutorial Step 2 – Chat Mode Selection

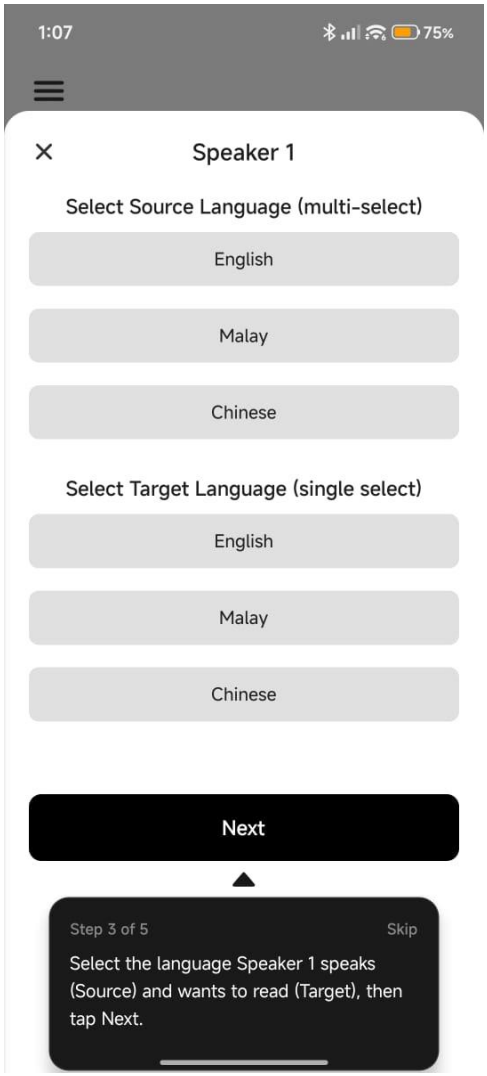


Figure 4.14: Tutorial Step 3 – Speaker 1 Language Configuration

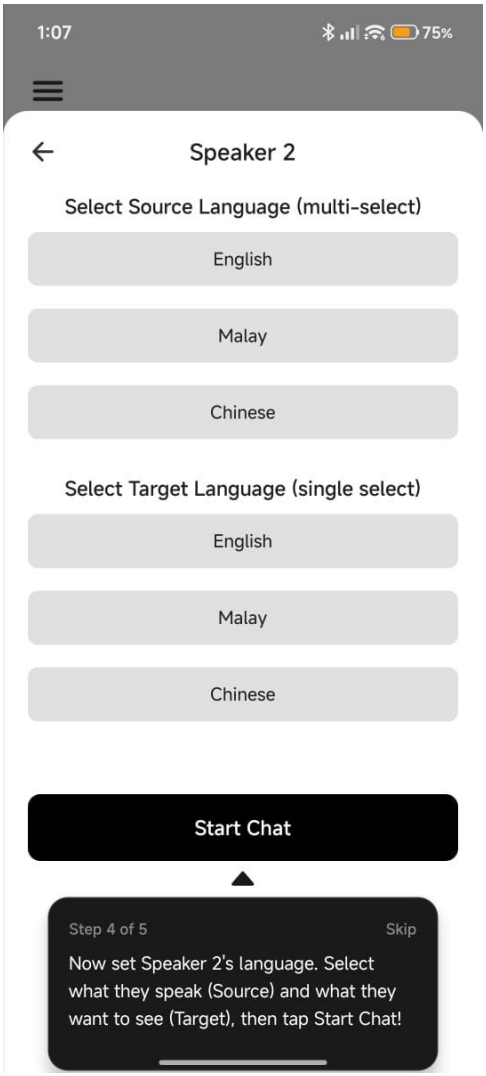


Figure 4.15: Tutorial Step 4 – Speaker 2 Language Configuration

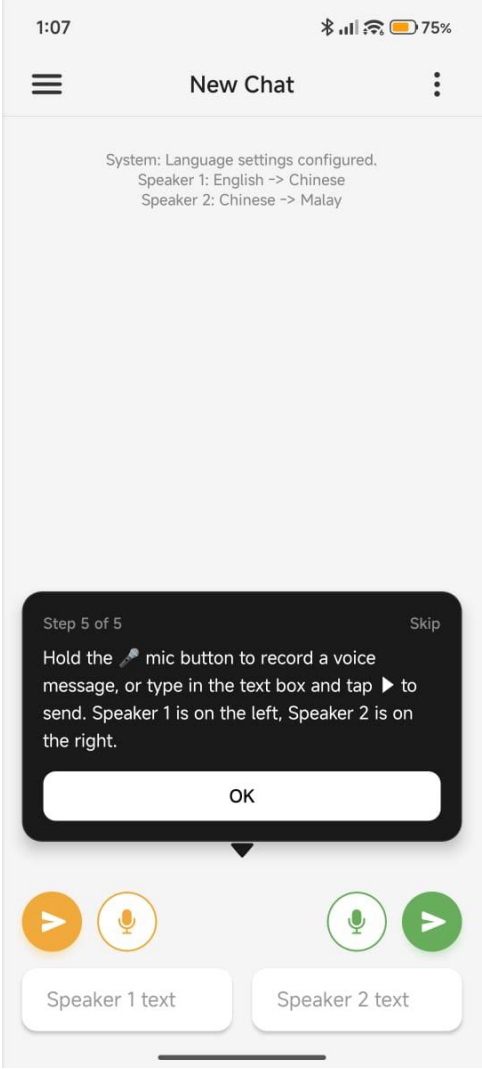


Figure 4.16: Tutorial Step 5 – Chat Interface Guidance

4.4.3 Translation Chat Interface Implementation

As shown in Figure 4.17, the Translation Chat module is developed with a structured layout to support real-time multilingual communication. To provide visual clarity and distinguish between different participants, the module uses a colour-coded bubble system that managed with a *FlatList* component. By using this component, messages from Speaker 1 are identified with an orange background and messages from Speaker 2 are displayed in green.

Beyond the visual arrangement, the interface includes 2 audio features to improve accessibility and interaction of users. Each translated message is paired with a speaker icon. It is used to triggers a TTS engine via an *onPress* event listener. This allows users to hear the pronunciation of the translated text. Next, a voice input functionality is provided through another microphone button at the bottom of the screen. This functionality utilizes a long-press gesture logic to record speech from the users.

Figure 4.17: Translation Chat Interface

4.4.3.1 More Options

To provide users with better control over conversation management, a "More Options" menu is integrated into the top-right corner of the interface. This menu provides management functions including renaming the chat, deleting the chat and changing the translation languages.

When the rename function is selected (Figure 4.18), the application will displays a text input field to allow the user to modify the conversation title. Once the user updated the new title, the system synchronizes the input with the Firestore database to ensure the change is persisted across sessions. For the delete functionality (Figure 4.19), the application implements a "soft delete" logic instead of permanent data removal. Rather than executing the *deleteDoc()* function, the system adds a *deletedAt* timestamp as a flag of deletion. The application also allows users reconfigure their source and target languages at any time as shown in Figure 4.20.

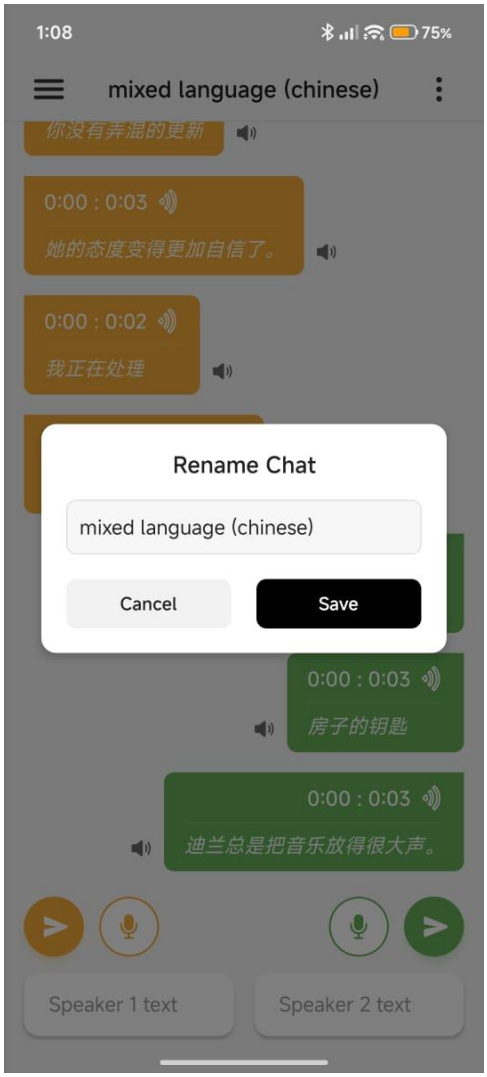


Figure 4.18: Rename Chat Functionality

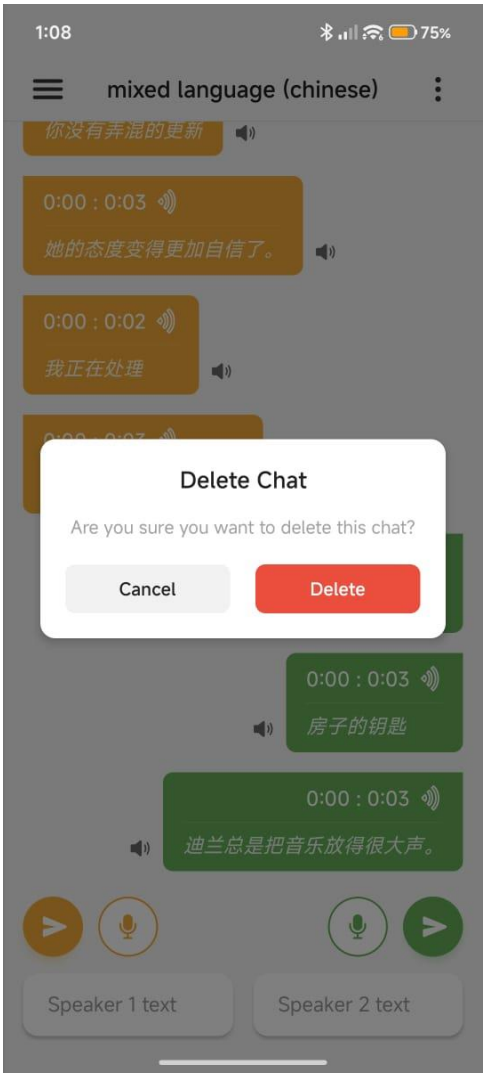


Figure 4.19: Delete Chat Confirmation Dialog

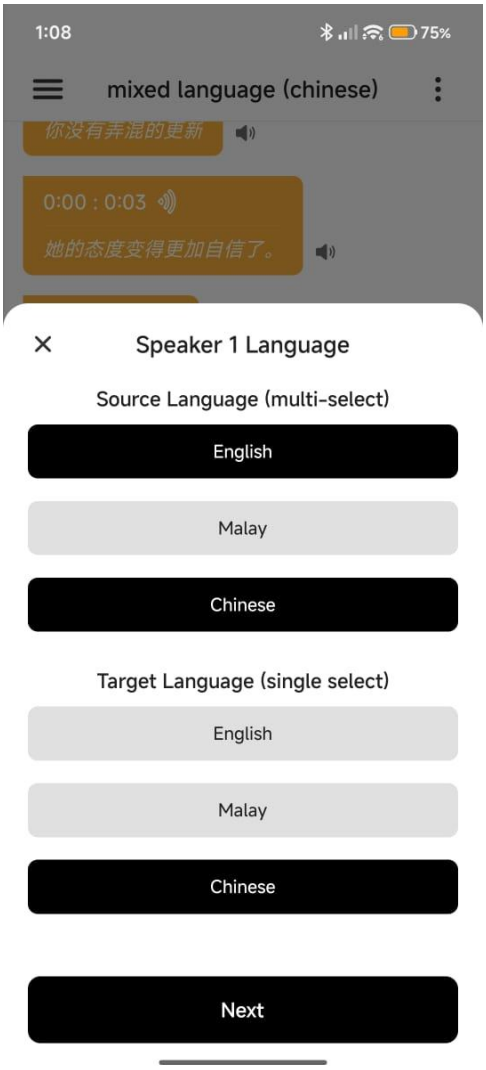


Figure 4.20: Change Language Settings Interface

4.4.4 User Profile Implementation

The User Profile module is a central hub for account management. As shown in Figure 4.21, this module shows the user's identity and a set of functional options in a structured layout.

The profile screen retrieves the user's name and email and shows them at the top beside a profile icon. Below the user information, a list of navigational items is implemented including "Edit Profile", "Dark/Light mode" and "Guideline". Each one is a touchable element that will navigate user to specialized screen or toggles a system settings.

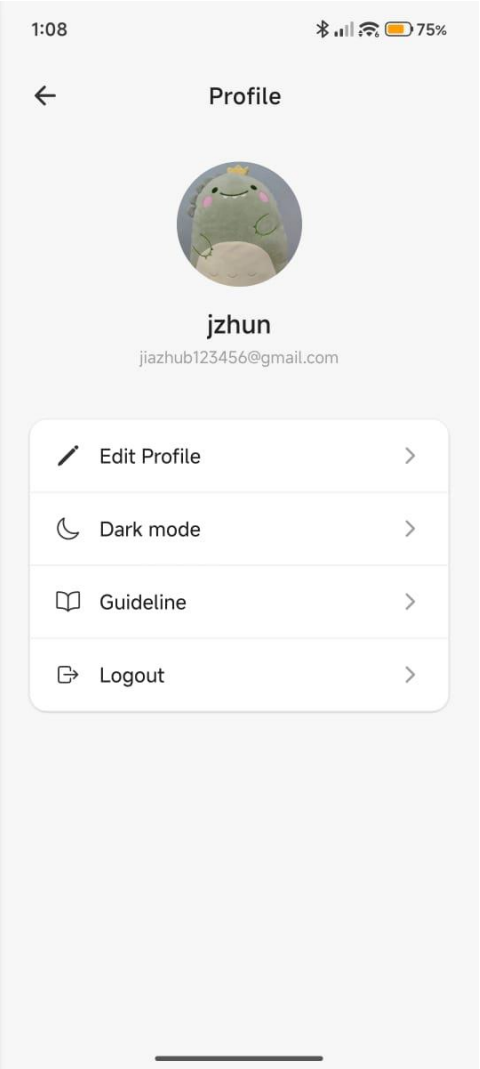


Figure 4.21: User Profile Interface

4.3.4.1 User Profile Edit

The Edit Profile module enables users to change their account details (Figure 4.22). It has a text field for updating the username and a read-only field showing the email address. This design ensures that the user's primary identifier remains consistent while still allowing for personal customization. When user saves their changes, the application will update to the Firestore database and ensures the new username is reflected across the entire application.

Technically, the module utilize the *useState* hooks of React Native to manage the local state of the username input during the editing process. Furthermore, an profile picture editing section is provided for profile picture interactions. When the "Save Changes" button is triggered, the application will execute an asynchronous call using the *updateProfile* API from Firebase Authentication and update the user's data. Upon a successful update, a feedback message is displayed and the new information will immediately reflected across the application state (Figure 4.23).

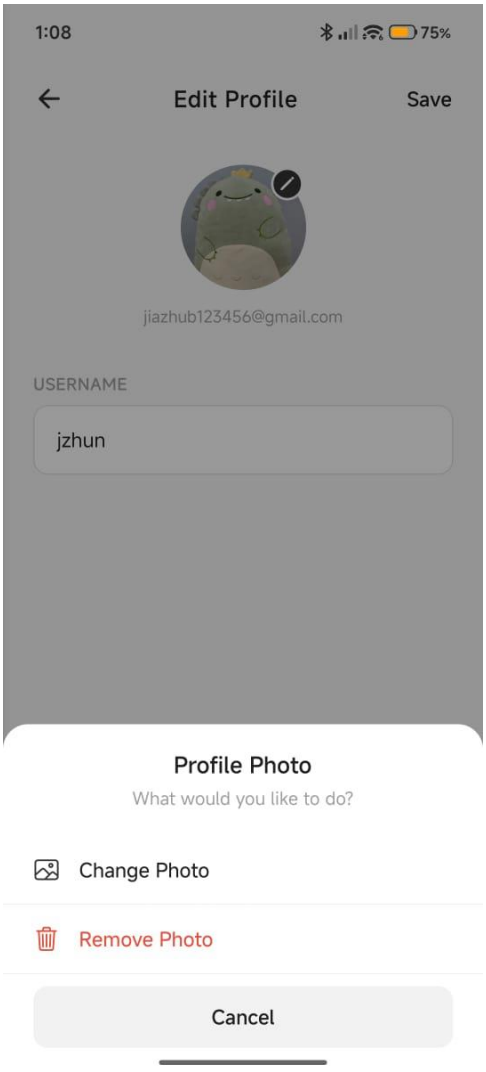


Figure 4.22: Edit Profile Interface

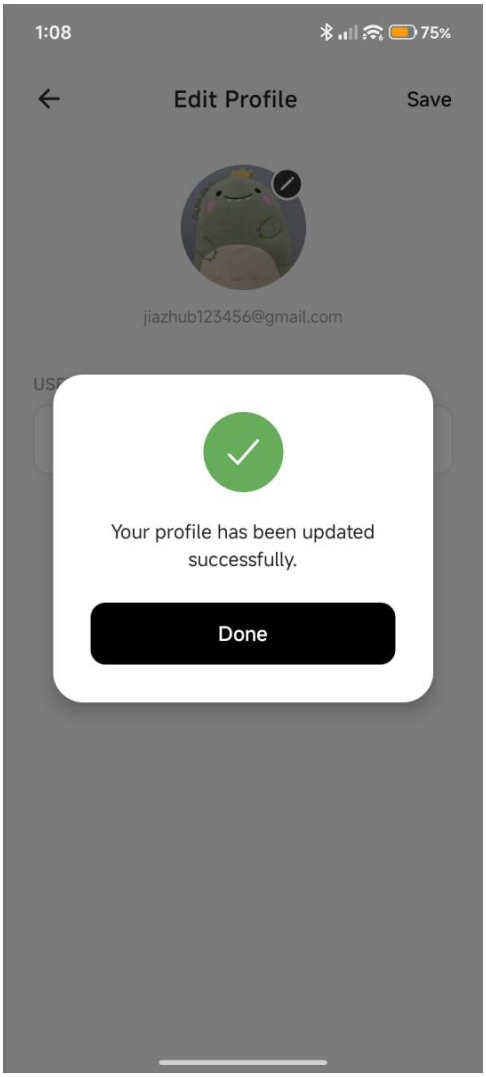


Figure 4.23: Profile Update Confirmation Dialog

4.3.3.2 Dark Mode Implementation

The Dark Mode sub-module provides users with an alternative visual theme for improved comfort when they are using the application. This feature is implemented using a global theme provider that wraps the entire application. It distributes dynamic colour variables to internal modules such as the chat interface and profile settings (Figure 4.24 and Figure 4.25). When the "Dark mode" toggle is activated, the background of internal screens shifts to a dark palette while text and icons are inverted.

Figure 4.24: Dark Mode Chat Interface

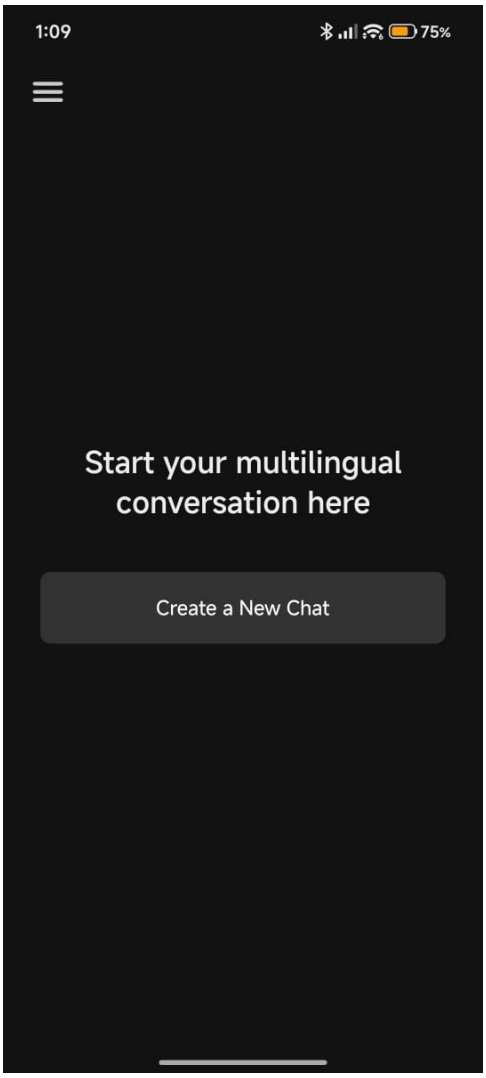


Figure 4.25: Dark Mode Application Entry Interface

4.4.5 Remote Translation Chatroom Implementation

The Remote Translation Chatroom module is implemented to facilitate communication between users in different geographic locations. Two users can join a translation session by using a unique code without the consideration of distance.

The module operates using room-based connection logic. One user initiates a new session by selecting the "Create a New Chat" button, then choose the remote connection mode. It will trigger a function to generate a unique alphanumeric invitation code (Figure 2.26). This code is stored in the Firestore database to mark the session as active. Another user can then join the session by entering the code (Figure 2.27). At this stage, the application performs an asynchronous query to verify the validity and status of the code in the Firestore backend.

Once connected, both users are redirected to shared chat interface. To ensure a low-latency experience, the application uses Firestore's *onSnapshot* function to listen for updates in real time. This listener ensures that any message or translation is immediately updated and displayed on the other person's screen.

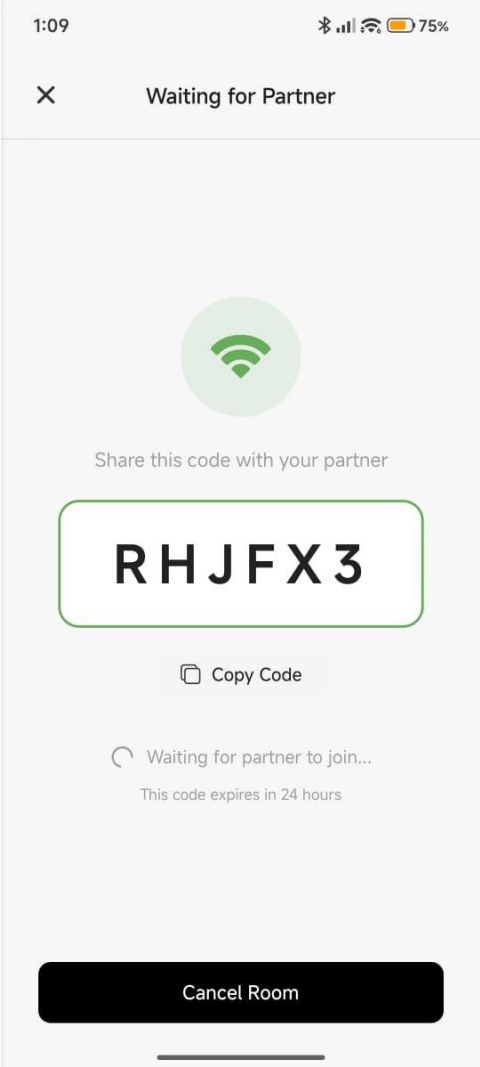


Figure 2.26: Waiting for Partner Interface

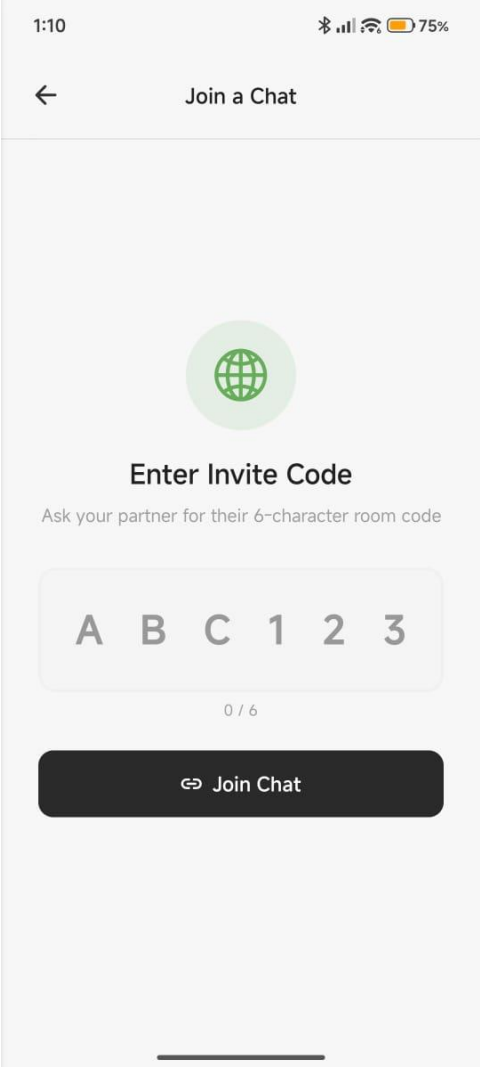


Figure 2.27: Join Chat Interface

CHAPTER 5 – SYSTEM EVALUATION AND DISCUSSION

5.1 Evaluation of Fine-tuned ASR Engine

The evaluation of the fine-tuned ASR engine focuses on its performance when handling Malaysian code-mixed speech.

5.1.1 Evaluation Metrics

Three metrics were uses to evaluate the model across Mandarin and English/Malay scripts. These metrics are Word Error Rate (WER), Character Error Rate (CER) and Mixed Error Rate (MER) [7].

5.1.2 Experimental Setup

The experimental evaluation used a combined test set built from unseen portions of the training data. This evaluation set was created by concatenating the held-out splits from the ASCEND dataset and the Malay-Speech-3k dataset. Three model configurations were benchmarked to track the performance changed during training. These model are pre-trained base (SeamlessM4T-v2-large), the intermediate model (representing Epoch 1) and the final model (representing Epoch 5).

5.1.3 Quantitative Performance Analysis

Table 5.1 summarizes the results. The data show a clear improvement after fine-tuning on domain-specific data.

Model	WER	CER	MER
Base Model	1	1	1
Intermediate (Epoch 1)	0.2577	0.1048	0.2207
Final (Epoch 5)	0.2448	0.0982	0.2111

Table 5.1 Comparative Performance of ASR Models

As shown in Table 5.1, the base model recorded an error rate of 100% across all metrics. Qualitative inspection of the outputs revealed that the pre-trained SeamlessM4T-v2 model defaulted to Speech-to-Text Translation (S2TT) mode rather than the required ASR mode. For instance, when provided with mixed-language input, the base model will translate Mandarin segments into English while maintain semantic meaning as shown in Figure 5.1.

In contrast, the intermediate model (Epoch 1) exhibited a reduction in error rates after the initial training pass. This shift confirms that the model successfully adapted its modality from translation to transcription by learning the acoustic and linguistic boundaries as shown in Figure 5.2. The final model (Epoch 5) achieved the highest accuracy. It got the lowest values across WER, CER and MER. Although the performance gap between Epoch 1 and Epoch 5 is narrower, but this phase represents critical model convergence. Extended training enabled the model to better resolve ambiguities in code-switching boundaries and make the model have a higher precision for specialized Malaysian vocabulary.

```
ers.integrations
  warnings.warn(
W1127 21:06:05.828000 30884 site-packages\torch\distributed\elastic\multiprocessing\redirects.py:29] NOTE: Redirects are
 currently not supported in Windows or MacOs.
Loading checkpoint shards: 100%|████████████████████████████████████████████████████| 2/2 [00:01<00:00,  1.14it/s]
C:\Users\User\miniconda3\envs\cs_asr\lib\site-packages\huggingface_hub\file_download.py:942: FutureWarning: `resume_down
load` is deprecated and will be removed in version 1.0.0. Downloads always resume when possible. If you want to force a
new download, use `force_download=True`.
  warnings.warn(
Loaded audio, length: 8.52 seconds
You must either specify a `tgt_lang` or pass a correct `text_decoder_input_ids` to get
                a correct generation, otherwise the generation will probably make no sense.

=== Transcription ===
In order for our children to receive the best education, we have to adjust the financial plan of the family.
====================
```

Figure 5.1: Translation Result generated by the Base Model

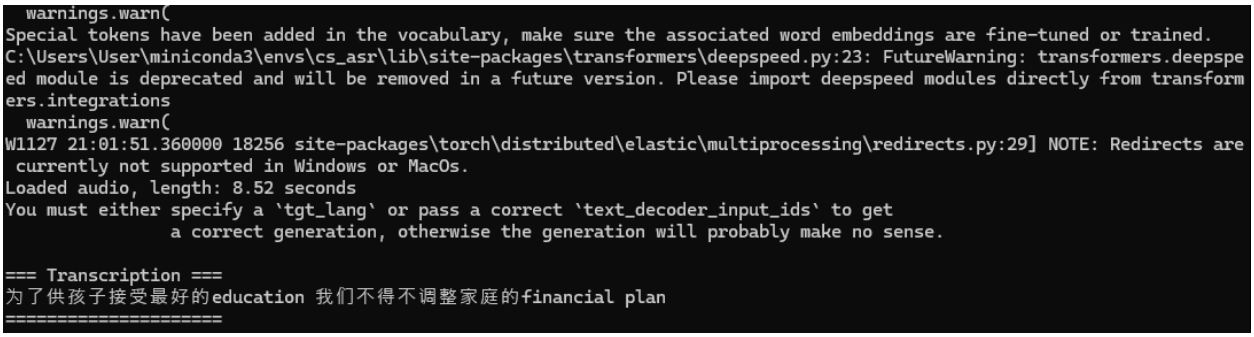


Figure 5.2: Translation Result generated by the Final Model

5.2 Translation Accuracy Evaluation

The translation functionality of the application was evaluated to measure its effectiveness in real-world communication. The objective of this evaluation was to assess the system's accuracy in translating single-language speech and the code-mixed speech.

Each test scenario was run 10 times with voice input under acoustic conditions. The success rate was calculated by counting correct translations against total attempts. By using this approach, it helps identify specific failure points in the translation engine when processing diverse linguistic structures.

5.2.1 Single Language Translation

The first phase tested accuracy on single-language input to confirm Gemini handles standard linguistic input reliably. Results for six translation directions are in Table 5.2.

Source Language	Target Language	Correctness valiation	Accuracy
Malay	English	8/10	80%
Chinese	English	10/10	100%
English	Chinese	9/10	90%
Malay	Chinese	10/10	100%
English	Malay	10/10	100%
Chinese	Malay	10/10	100%

Table 5.2: Accuracy of Single Language Translation

Across all single-language tests, accuracy landed between 80% and 100%. Four out of six directions scored perfectly. Malay-to-English came in at 80% and English-to-Chinese at 90%. The system handles standard linguistic input without much trouble.

5.2.2 Mixed Language Translation

The second phase tested accuracy on code-mixed speech where two or three languages are combined in a single sentence. The results are categorized based on the target translation language as shown in Table 5.3, Table 5.4 and Table 5.5

Source Language	Target	Correct	Accuracy
Malay + English	English	9/10	90%
Chinese + English	English	10/10	100%
Malay + Chinese	English	6/10	60%
Malay + English + Chinese	English	5/10	50%

Table 5.3: Accuracy of Mixed Language to English Translation

Source Language	Target	Correct	Accuracy
English + Malay	Malay	7/10	70%
Chinese + Malay	Malay	5/10	50%
English + Chinese	Malay	8/10	80%
Malay + English + Chinese	Malay	5/10	50%

Table 5.4: Accuracy of Mixed Language to Malay Translation

Source Language	Target	Correct	Accuracy
English + Chinese	Chinese	7/10	70%
Malay + Chinese	Chinese	5/10	50%
English + Malay	Chinese	7/10	70%
Malay + English + Chinese	Chinese	6/10	60%

Table 5.5: Accuracy of Mixed Language to Chinese Translation

The experimental results show that the system performs reliably when translating bilingual sentences. The accuracy rates are ranging between 70% and 100%. High performance is particularly noted in English-Chinese and English-Malay combinations. However, a decline in accuracy is observed when Malay and Chinese are mixed in the same sentence. The accuracy rates are dropped to around 50% to 60%.

The lowest performance levels were recorded in trilingual scenarios. The accuracy rates are ranging from 50% to 60%. this degradation is likely attributed to the increased complexity of the acoustic boundaries and the high probability of semantic overlap between the different scripts and phonemes.

5.3 Robustness Test For Incorrect Source Language Selection

This test evaluates the system's ability to handle user errors during language selection. In traditional translation applications, if user selects "English" as the source language but speaks in "Chinese", the translation application usually fails to generate a correct output. However, since our translate engine is LLM, it able to guess and infer the meaning of the user and generate appropriate output. A total of 15 sample sentences were used for this evaluation, with 5 sentences selected for each language (English, Malay and Chinese). The results are summarized in Table 5.6 below.

Actual Input Language	Source Language Setting	Success Rate	Overall Accuracy
Chinese	English	5/5	100%
Malay	Chinese	4/5	80%
English	Malay	5/5	100%

Table 5.6: Accuracy of Translation with Incorrect Source Language Selection

The results show that the application is highly robust against incorrect language selection. For English and Chinese, the system achieved a 100% success rate. Even though the selected source language is incorrect, the application still successfully recognized the Chinese and English speech and provided the correct translation.

In the Malay language test, the accuracy was 80%. The is because the recognition rate for the Malay language is naturally lower compared to English and Chinese. This makes occasional errors unavoidable. However, during this test, the application still managed to correctly identify and accurately translate most of the Malay input. This performance proves that the application is much more flexible and it can provide correct results even when users select the wrong source language.

5.4 Noise Testing (Environmental Evaluation)

As the application allows user to input by using voice, the robustness of the ASR engine against environmental noise is essential. Therefore, the assessment of robustness of the ASR engine is conducted. This evaluation utilized a test set of 15 sample sentences which is 5 sentences for each language (English, Malay and Chinese). The tests were conducted across 2 distinct environments. The first environment is a quiet indoor setting and the second environment is a noisy restaurant environment. The results are summarized in Table 5.7 below.

Environment	**English**	**Malay**	**Chinese**	**Overall Accuracy**
Indoor (Quiet)	5 / 5	3 / 5	5 / 5	86.67%
Restaurant (Loud)	5 / 5	2 / 5	4 / 5	73.33%

Table 5.7: Voice Recognition Results in Different Environments

The finding from the noise testing indicate that the system is relatively resilient in real-world scenarios. Even in the noisy environment, the overall accuracy remained at 73.33%. This indicated that background noise does not severely compromise the engine's core functionality. English and Chinese exhibited the highest stability. English maintained a 100% success rate in both environment and Chinese showed only a minor decrease in the noisy environment.

However, the results show that Malay language recognition is significantly less accurate compared to English and Chinese. Even in the quiet indoor environment, Malay recognition accuracy was only 60%. This discrepancy may be attributed to lower optimization levels for the Malay language compared to other languages.

5.5 Testing Application with Different Devices

To ensure the application remains stable across different mobile environments, cross-device compatibility testing was performed. Since Android devices have a wide variety of OS version and screen size, it is necessary to verify the UI remains consistent across them. Table 5.7 below lists the specific Android devices that being used in this evaluation.

The testing methodology involved deploying the application on several physical devices with different specifications. During this process, the focus was on the layouts responded properly and the interactive elements such as the microphone button, were easy to use.

No.	Device Model	Operating System	Screen Size
1	Xiaomi 15T Pro	Android 14	2712 x 1220
2	Xiaomi 11 Lite 5G NE	Android 13	2400 x 1080
3	Redmi Note 11	Android 12	2400 x 1080

Table 5.8: Type of Android Devices for Testing

5.5.1 Test Cases

User Authentication & Authorization

ID	Scenario	Steps	Expected Result	Status		
				Xiaomi 15T Pro	Xiaomi 11 Lite 5G NE	Redmi Note 11
1	Successful New Registration	1. Open Registration page 2. Enter valid email/password 3. Tap Register	User registers successfully; redirected to home/tutorial. Firebase Auth creates the user.	✅	✅	✅
2	Register with Existing Email	1. Attempt to register with an existing email	Displays "Email already in use" error; remains on page without crashing.	✅	✅	✅
3	Successful Email Login	1. Enter registered email/password 2. Tap Login	Login succeeds, loads chat history, and redirects to the home screen.	✅	✅	✅
4	Login with Incorrect Password	1. Enter correct email, wrong password	Displays an "Invalid credentials" error message.	✅	✅	✅
5	Forgot Password Functionality	1. Tap "Forgot Password" 2. Enter email and submit	Prompts "Password reset email sent"; user receives Firebase reset link.	✅	✅	✅

Table 5.9: Test Cases for User Authentication & Authorization

Profile & User Settings

ID	Scenario	Steps	Expected Result	Status		
				Xiaomi 15T Pro	Xiaomi 11 Lite 5G NE	Redmi Note 11
1	View Profile Page	1. Tap avatar icon in sidebar or navigate to Profile	Avatar, email, and statistics are displayed correctly.	✅	✅	✅
2	Edit Username	1. Tap "Edit Profile" 2. Modify username and click Save	Pop-up: "Your profile has been updated successfully." New name reflects immediately.	✅	✅	✅
3	Upload/Modify Avatar	1. Tap 'pencil' icon on avatar 2. Pick gallery photo and save	Photo updates immediately; image uploaded successfully to Firebase Storage.	✅	✅	✅
4	Toggle Dark Mode	1. Toggle Dark/Light mode switch	Backgrounds, components, and fonts invert seamlessly into the selected mode.	✅	✅	✅
5	Logout Function	1. Tap Logout in Profile or Sidebar	Sidebar cache clears, session ends, and app routes back to Login Screen.	✅	✅	✅

Table 5.10: Test Cases for Profile & User Settings

Personal Chat

ID	Scenario	Steps	Expected Result	Status		
				Xiaomi 15T Pro	Xiaomi 11 Lite 5G NE	Redmi Note 11
1	Create New Local Chat	1. Create New Chat 2. Choose Personal Chat 3. Select both languages	Navigates to ChatScreen with language config shown at the top.	✅	✅	✅
2	Send Text Message (Speaker 1)	1. Type message in bottom-left 2. Tap Send	Speaker 1's bubble appears with source text and target translation.	✅	✅	✅
3	Voice Input (STT)	1. Long press microphone icon 2. Speak and release	Recording overlay appears; voice converts to text, sends, and translates.	✅	✅	✅
4	TTS Audio Playback	1. Tap speaker icon next to bubble	App plays back the translated sentence in the specified language.	✅	✅	✅
5	Change Chat Languages	1. Tap Header Menu 2. Tap 'Change Language' button 3. Update language	System message: "Language settings updated"; applies to new messages.	✅	✅	✅

6	Rename Chat Room	1. Tap Header Menu 2. Tap ‘Rename’ button 2. Modify name and Save	Chat renamed via modal; header title updates instantly.	✅	✅	✅

Table 5.11: Test Cases for Personal Chat

Remote Chat

ID	Scenario	Steps	Expected Result	Status		
				Xiaomi 15T Pro	Xiaomi 11 Lite 5G NE	Redmi Note 11
1	Host: Creating a Room	1. Create New Chat 2. Choose connect room	Navigates to WaitingRoomScreen; presents 6-digit Invite Code.	✅	✅	✅
2	Guest: Joining a Room	1. Enter the 6-digit Invite Code	Guest joins; Host's screen transitions to synced RemoteChatScreen.	✅	✅	✅
3	Guest: Invalid Invite Code	1. Enter non-existent/expired code	App throws "Room not found or expired" alert.	✅	✅	✅
4	Remote Chat Sync	1. Host sends a text message 2. Observe Guest device	Guest receives bubble immediately with translation applied for their language.	✅	✅	✅
5	Host: Cancel Waiting Room	1. Host clicks Cancel in waiting room	Room is destroyed, Firebase docs deleted, user routes to Home.	✅	✅	✅

Table 5.12: Test Cases for Remote Chat

Sidebar & Chat History

ID	Scenario	Steps	Expected Result	Status		
				Xiaomi 15T Pro	Xiaomi 11 Lite 5G NE	Redmi Note 11
1	Categorized List Generation	1. Open Sidebar from Home	List partitioned into Remote Chat and Personal Chat sections.	✅	✅	✅
2	History Search & Filter	1. Enter query in Sidebar Search bar	List dynamically re-renders to show matching history labels.	✅	✅	✅
3	Seamless Resume	1. Tap any prior chat list node	App recovers session; loads exact history log in ChatScreen.	✅	✅	✅
4	Permanent Chat Deletion	1. Choose Chat 2. Tap Menu 3. Delete Chat	Wipes Firebase documents; routes home; Sidebar access removed.	✅	✅	✅

Table 5.13: Test Cases for Sidebar & Chat History

Application Onboarding & Tutorial

ID	Scenario	Steps	Expected Result	Status		
				Xiaomi 15T Pro	Xiaomi 11 Lite 5G NE	Redmi Note 11
1	Initial First-Time Display	1. Register new user 2. Log in	First coach mark prompt appears correctly over UI without clipping.	✅	✅	✅
2	Structural Integrity Check	1. Complete the step sequence	Steps lock onto focal elements (Buttons, Inputs) with clear tooltips.	✅	✅	✅
3	Manual Re-activation	1. Tap 'App Guideline' in Profile	Onboarding progression re-triggers successfully in current theme (Dark/Light).	✅	✅	✅

Table 5.14: Test Cases for Application Onboarding & Tutorial

All test devices passed the functional tests, covering user authentication (Table 5.9), profile settings (Table 5.10), chat logic (Tables 5.11 and 5.12), sidebar history (Table 5.13), and the onboarding tutorial (Table 5.14). The application adapted to different screen resolutions without layout issues or overlapping elements.

CHAPTER 6 – CONCLUSION AND RECOMMENDATION

6.1 Project Summary

This project built a multilingual translation app designed for the Malaysian. By integrating the Google Gemini API, the system effectively handles the complexities of code-mixed language that frequently encountered in daily conversations. The implementation on the Android platform shows AI models can deliver context-aware translations on mobile. These tools produce context-aware translations that fit how Malaysians actually communicate across Malay, Chinese, and English.

6.2 Achievement of Objectives

The objectives defined in the Chapter 1 have been successfully achieved. The result is a fully functional Android application that brings together fine-tuned ASR, LLM and TTS modules. Benchmarking results show that the system handles Malaysian linguistic nuances more effectively compared to other translation application. The successful fine-tuning of the SeamlessM4T engine and the integration of the Gemini API provided the necessary precision to resolve multilingual boundaries and makes the project achieves the goals.

6.3 Limitations and Challenges

Several technical limitations and challenges were encountered during the development process. First, the precision of the ASR engine remains inconsistent when processing Malay dialects. This performance gap is primarily due to the lack of high-quality datasets. This affected the model's ability to recognize informal speech accurately.

Second, the full fine-tuning of the large-scale SeamlessM4T-v2-large model imposed a heavy computational burden. Due to the 112GB VRAM constraints of the RTX5070 hardware, training sessions had to be extended longer periods to avoid memory errors. These hardware limitations restricted the number of hyperparameter tuning sessions and training iterations that could be performed within the project timeline.

Finally, due to the limited time to complete the project, the current version of the application lacks some advanced features. Although the core translation are developed, but the further features may enhance the user experience of applications.

6.4 Future Recommendations

To further improve the application's performance, several directions for future research are recommended. First, it is suggested to curate a larger and more diverse dataset of Malay audio recordings. These speech could further improve the ASR precision.

Future research should explore more efficient training strategies to mitigate the high resource requirements. Teachniques such as Parameter Efficient Fine-Tuning (PERT) or Low-Rank Adaptation (LoRA) should be investigated [50] [51]. LoRA cuts down VRAM usage by training low-rank matrices instead of the full weight set. This makes it possible to adapt the model on consumer-grade GPUs without adding inference latency or bloating storage costs.

Future updates should focus on expanding the application's functionality. In future, the application could support the offline translation, so that allows users to do translation without an internet connection.

References

[1] "MyGovernment - Government of Malaysia's Official Portal," *Malaysia.gov.my*, 2026. https://www.malaysia.gov.my/en/government/kenali-malaysia/official-language

[2] *Studytravel.network*, 2025. https://studytravel.network/magazine/news/0/31256

[3] R. RAHIM, M. CARVALHO, and J. IBRAHIM, "Over 2.4 million foreign workers in the country as of Sept, Parliament hears," *The Star*, Dec. 05, 2024. https://www.thestar.com.my/news/nation/2024/12/05/over-24-million-foreign-workers-in-the-country-as-of-sept-parliament-hears

[4] V. Iyer, P. Chen, and A. Birch, "Towards Effective Disambiguation for Machine Translation with Large Language Models," *ACLWeb*, Dec. 01, 2023. https://aclanthology.org/2023.wmt-1.44/

[5] F. Martelli *et al.*, "DiBiMT: A Gold Evaluation Benchmark for Studying Lexical Ambiguity in Machine Translation," *Computational Linguistics*, pp. 1–79, Sep. 2024, doi: https://doi.org/10.1162/coli_a_00541.

[6] M. Faisal, "Translation Technology Trends 2025: Why LLM Surpasses NMT," *AD VERBUM AI+HUMAN TRANSLATIONS*, Dec. 10, 2025. https://www.adverbum.com/post/translation-technology-trends-2025 (accessed Apr. 10, 2026).

[7] T. Nguyen and H.-D. Tran, "Can we train ASR systems on Code-switch without real code-switch data? Case study for Singapore's languages," *arXiv preprint arXiv:2506.14177*, 2025. https://arxiv.org/abs/2506.14177.

[8] H. Liu, X. Zhang, L. P. Garcia, Andy, C. E. Siong, and S. Watanabe, "Aligning Speech to Languages to Enhance Code-switching Speech Recognition," *arXiv.org*, 2024. https://arxiv.org/abs/2403.05887

[9] "ASR challenge with code-switching in multilingual nations," *Idiap Research Institute, Artificial Intelligence for Society*, 2026. https://www.idiap.ch/en/scientific-research/speech-and-audio-processing/speech-group-news/asr-challenge-with-code-switching-in-multilingual-nations (accessed Apr. 10, 2026).

[10] "Chat Completions Overview," *OpenAI API Reference*, 2026. https://developers.openai.com/api/reference/chat-completions/overview

[11] T. Kasakowskij and J. M. Haake, "T3 Talk2Text – A model for near real-time voice transcription in virtual group meetings," *Discover Education*, vol. 4, no. 1, Jun. 2025, doi: https://doi.org/10.1007/s44217-025-00568-6.

[12] ""Ultimate Guide to UI/UX Best Practices for Chat App Design":, *Cometchat.com*, 2024. https://www.cometchat.com/blog/chat-app-design-best-practices

[13] Y. Lin, X. Wang, Z. Zhang, M. Wang, T. Xiao, and J. Zhu, "MobileNMT: Enabling Translation in 15MB and 30ms," *arXiv preprint*, arXiv:2306.04235, Jun 2023. https://arxiv.org/abs/2306.04235.

[14] "facebook/mbart-large-50-many-to-many-mmt · Hugging Face," *huggingface.co*. https://huggingface.co/facebook/mbart-large-50-many-to-many-mmt.

[15] Y. Liu, J. Gu, N. Goyal, X. Li, S. Edunov, M. Ghazvininejad, M. Lewis, and L. Zettlemoyer, "Multilingual Denoising Pre-training for Neural Machine Translation," *arXiv preprint arXiv:2001.08210*, Jan 2020. https://arxiv.org/abs/2001.08210.

[16] "Introducing Gemini: our largest and most capable AI model," *Google*, Dec. 06, 2023. https://blog.google/technology/ai/google-gemini-ai/#scalable-efficient.

[17] A. Laurent, "AI API Pricing Comparison (2025): Grok, Gemini, ChatGPT & Claude," *IntuitionLabs,* Dec. 03, 2025. https://intuitionlabs.ai/articles/ai-api-pricing-comparison-grok-gemini-openai-claude

[18] Google, "Gemini Developer API Pricing," *Google AI for Developers*, 2025. https://ai.google.dev/gemini-api/docs/pricing

[19] OpenAI, "Introducing ChatGPT," *OpenAI*, Nov. 30, 2022. https://openai.com/index/chatgpt/.

[20] S. Chen and Y. Lin, "A multidimensional comparison of ChatGPT, Google Translate, and DeepL in Chinese tourism texts translation: fidelity, fluency, cultural sensitivity, and persuasiveness," *Frontiers in Artificial Intelligence*, vol. 8, Art. no. 1619489, Jul. 2025. https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2025.1619489/full.

[21] Muhamad Huzaifah, W. Zheng, Nattapol Chanpaisit, and K. Wu, "Evaluating Code-Switching Translation with Large Language Models," in *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING)*, Torino, Italy, May 2024, pp. 6381–6394. https://aclanthology.org/2024.lrec-main.565/.

[22] OpenAI, "Introducing Whisper," *Openai.com*, 2022. https://openai.com/index/whisper/.

[23] T. Nguyen and H.-D. Tran, "AsyncSwitch: Asynchronous Text-Speech Adaptation for Code-Switched ASR," *arXiv preprint arXiv:2506.14190*, 2025. https://arxiv.org/abs/2506.14190#:~:text=We%20introduce%20AsyncSwitch%2C%20a%20novel,on%20paired%20speech%2Dtext%20corpora.

[24] "Amazon Transcribe Features," *Amazon Web Services, Inc.*, Sep. 02, 2025. https://aws.amazon.com/transcribe/features/#topic-0.

[25] "Supported languages and language-specific features - Amazon Transcribe," *docs.aws.amazon.com*. https://docs.aws.amazon.com/transcribe/latest/dg/supported-languages.html.

[26] "Preserving the World's Language Diversity Through AI," *Meta*, May 22, 2023. https://about.fb.com/news/2023/05/ai-massively-multilingual-speech-technology/

[27] "SeamlessM4T—Massively Multilingual & Multimodal Machine Translation | Meta AI Research," *ai.meta.com*. https://ai.meta.com/research/publications/seamlessm4t-massively-multilingual-multimodal-machine-translation/

[28] "Azure AI Speech | Microsoft Azure," *Microsoft.com*, 2025. https://azure.microsoft.com/en-us/products/ai-services/ai-speech#Related-products.

[29] eric-urban, "Speech translation overview - Speech service - Azure AI services," *Microsoft.com*, Nov. 19, 2024. https://learn.microsoft.com/en-us/azure/ai-services/speech-service/speech-translation

[30] PatrickFarley, "Implement language identification - Speech service - Azure AI services," *Microsoft.com*, May 25, 2025. https://learn.microsoft.com/en-us/azure/ai-services/speech-service/language-identification?tabs=once&pivots=programming-language-csharp.

[31] “Google Translate - Apps on Google Play,” *play.google.com*. https://play.google.com/store/apps/details?id=com.google.android.apps.translate

[32] “Translate a bilingual conversation - Android - Google Translate Help,” *Google.com*. https://support.google.com/translate/answer/6142474?hl=en&co=GENIE.Platform%3DAndroid

[33] “translate simultaneously in three or more languages - Google Translate Community,” *Google.com*, 2023. https://support.google.com/translate/thread/219286962/translate-simultaneously-in-three-or-more-languages?hl=en (accessed Sep. 08, 2025).

[34] A. Speight, “Which Translation App Should You Use in 2021?,” *Wired*, May 14, 2021. https://www.wired.com/story/best-translation-apps/

[35] “What is the translation accuracy of DeepL? Comparison results with Google and Microsoft in business emails,” *Human Science Co., Ltd.* https://www.science.co.jp/en/nmt/blog/20529/

[36] “Language support | Cloud Translation,” *Google Cloud*. https://cloud.google.com/translate/docs/languages

[37] D. SE, “DeepL Translate,” *Google.com*, 2021. https://play.google.com/store/apps/details?id=com.deepl.mobiletranslator (accessed Sep. 08, 2025).

[38] “DeepL Translate App for Android and iOS,” *Deepl.com*, 2024. https://www.deepl.com/en/mobile-apps

[39] DeepL, “DeepL expands real-time voice translation capabilities with new features and upcoming Zoom integration,” *Prnewswire.com*, Jul. 23, 2025. https://www.prnewswire.com/news-releases/deepl-expands-real-time-voice-translation-capabilities-with-new-features-and-upcoming-zoom-integration-302511784.html.

[40] “DeepL Translator languages,” *DeepL Help Center | How Can We Help You?*, 2024. https://support.deepl.com/hc/en-us/articles/360019925219-DeepL-Translator-languages

[41] “Microsoft Translator,” *Microsoft Translator for Consumers*. https://www.microsoft.com/en-us/translator/

[42] "Microsoft Translator Languages - Microsoft Translator," *Microsoft Translator for Consumers*, May 31, 2024. https://www.microsoft.com/en-us/translator/languages/.

[43] "iTranslate Voice," *itranslate.com*. https://itranslate.com/voice

[44] "Language Feature Availability," *Itranslate.com*, 2025. https://itranslate.com/languages.

[45] N. Corp, "Naver Papago - AI Translator," *Google.com*, 2021. https://play.google.com/store/apps/details?id=com.naver.labs.translator (accessed Sep. 08, 2025).

[46] "Papago Translation APIs," *Ncloud-docs.com*, 2025. https://guide.ncloud-docs.com/docs/en/papagotranslation-api.

[47] "Naver's Papago Vs Google Translate, Which one better?," *Linguise.com*, Sep. 24, 2024. https://www.linguise.com/blog/guide/navers-papago-vs-google-translate-which-one-better/

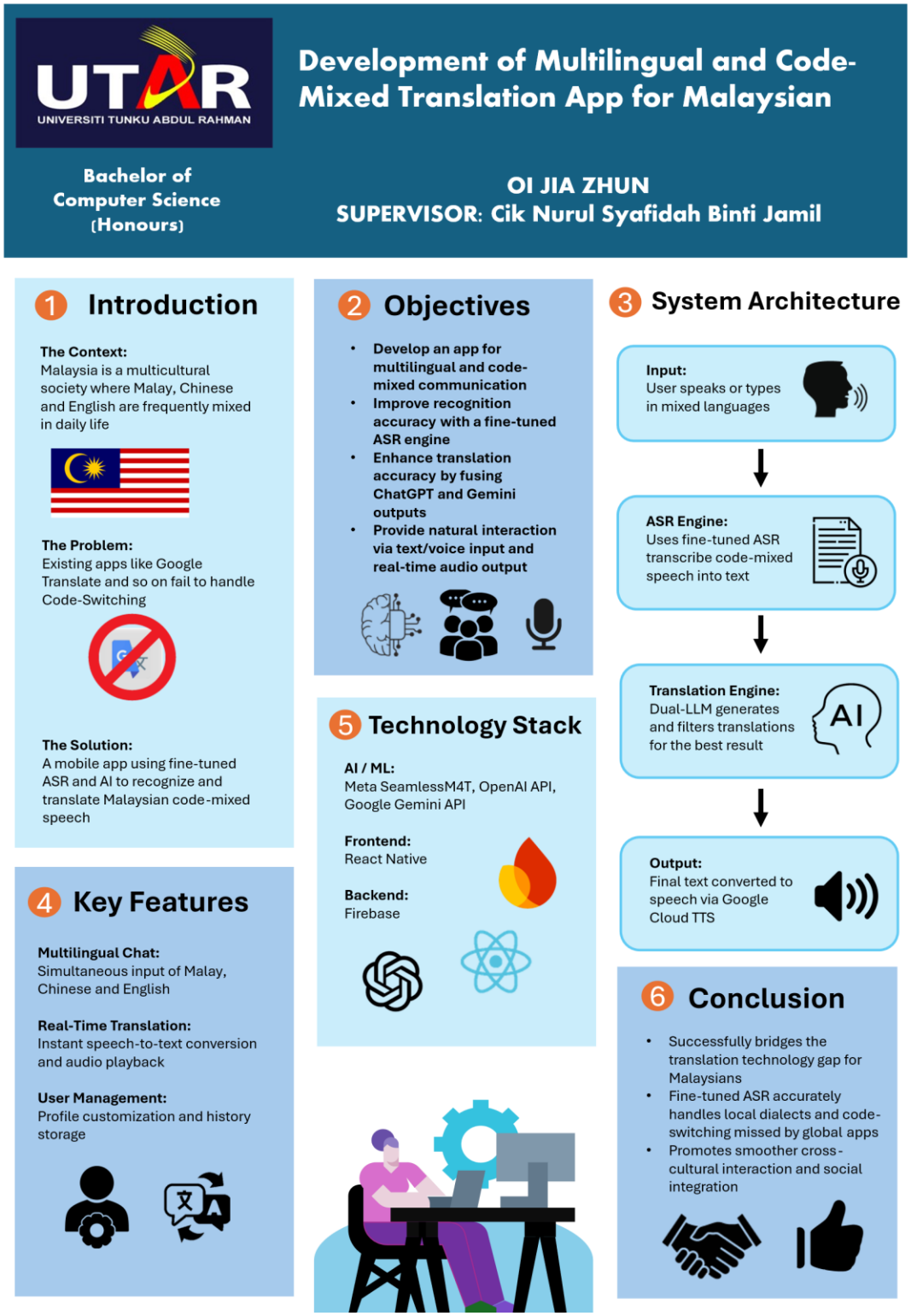
[48] "Naver's Papago Vs Google Translate Punch Digital Marketing -," *Punch Digital Marketing*, Sep. 29, 2024. https://www.punchkorea.com/naver-papago-vs-google-translate/.

[49] C. Drumond, "What is scrum and how to get started," *Atlassian*, 2024. https://www.atlassian.com/agile/scrum

[50] I. Belcic and C. Stryker, "Parameter-efficient fine-tuning," *Ibm.com*, Aug. 15, 2024. https://www.ibm.com/think/topics/parameter-efficient-fine-tuning

[51]J. Noble, "lora," *Ibm.com*, Jan. 27, 2025. https://www.ibm.com/think/topics/lora

Poster


UTAR
UNIVERSITI TUNKU ABDUL RAHMAN
Development of Multilingual and Code-Mixed Translation App for Malaysian
Bachelor of Computer Science (Honours)
OI JIA ZHUN
SUPERVISOR: Cik Nurul Syafidah Binti Jamil
1 Introduction
The Context:
Malaysia is a multicultural society where Malay, Chinese and English are frequently mixed in daily life
The Problem:
Existing apps like Google Translate and so on fail to handle Code-Switching
The Solution:
A mobile app using fine-tuned ASR and AI to recognize and translate Malaysian code-mixed speech
2 Objectives
Develop an app for multilingual and code-mixed communication
Improve recognition accuracy with a fine-tuned ASR engine
Enhance translation accuracy by fusing ChatGPT and Gemini outputs
Provide natural interaction via text/voice input and real-time audio output
3 System Architecture
Input:
User speaks or types in mixed languages
ASR Engine:
Uses fine-tuned ASR transcribe code-mixed speech into text
Translation Engine:
Dual-LLM generates and filters translations for the best result
AI
Output:
Final text converted to speech via Google Cloud TTS
4 Key Features
Multilingual Chat:
Simultaneous input of Malay, Chinese and English
Real-Time Translation:
Instant speech-to-text conversion and audio playback
User Management:
Profile customization and history storage
5 Technology Stack
AI / ML:
Meta SeamlessM4T, OpenAI API, Google Gemini API
Frontend:
React Native
Backend:
Firebase
6 Conclusion
Successfully bridges the translation technology gap for Malaysians
Fine-tuned ASR accurately handles local dialects and code-switching missed by global apps
Promotes smoother cross-cultural interaction and social integration