DEVELOPMENT OF SIBI AND BISINDO FINGERSPELLING LEARNING MEDIA BASED ON CLOUD VOICE RECOGNITION: AN ITERATIVE DEVELOPMENT APPROACH

Conduct by:

Name	: Muhammad Fazli Ramadhani Sukma
Student ID	: 22523070

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2026

SUPERVISOR ENDORSEMENT PAGE

DEVELOPMENT OF SIBI AND BISINDO FINGERSPELLING LEARNING MEDIA BASED ON CLOUD VOICE RECOGNITION: AN ITERATIVE DEVELOPMENT APPROACH

THESIS

Conduct By:

Name	: Muhammad Fazli Ramadhani Sukma
Student ID	: 22523070

Yogyakarta, August 3rd, 2026

Advisor,

(Ir. Chandra Kusuma Dewa, S.Kom., M.Kom., Ph.D.)

EXAMINER ENDORSEMENT PAGE

DEVELOPMENT OF SIBI AND BISINDO FINGERSPELLING LEARNING MEDIA BASED ON CLOUD VOICE RECOGNITION: AN ITERATIVE DEVELOPMENT APPROACH

THESIS

Has been defended in front of the examiners as one of the requirements to obtain a Bachelor of Informatics degree from the Undergraduate Program in Informatics at the Faculty of Industrial Technology, Universitas Islam Indonesia

Yogyakarta, August 3rd, 2026

Chair

Ir. Chandra Kusuma Dewa, S.Kom., M.Kom., Ph.D. _____

Examiner 1

Ahmad Fathan Hidayatullah, S.T., M.Cs., Ph.D. _____

Examiner 2

Ir. Kurniawan Dwi Irianto, S.T., M.Sc. _____

Acknowledged by,

Head of Undergraduate Program in Informatics

Faculty of Industrial Technology

Universitas Islam Indonesia

(Ir. Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.)

AUTHENTICITY STATEMENT

The undersigned:

Name	:	Muhammad Fazli Ramadhani Sukma
Student ID	:	22523070

Final project with title:

DEVELOPMENT OF SIBI AND BISINDO FINGERSPELLING LEARNING MEDIA BASED ON CLOUD VOICE RECOGNITION: AN ITERATIVE DEVELOPMENT APPROACH

Stating that all components and contents in this final project are my own work. If in the future it is proven that some parts of this work are not my own work, the final project submitted as my own work is ready to be withdrawn and ready to bear any risks and consequences.

Thus this statement letter is made, hopefully it can be used properly.

Yogyakarta, August 3rd, 2026



(Muhammad Fazli Ramadhani Sukma)

DEDICATION

This thesis is wholeheartedly dedicated to:

My Mom and Dad, whose endless prayers, patience, and sacrifice made every step of this journey possible. Nothing I achieve would exist without you.

Yuvita Indah Althaffy, for the constant support, encouragement, and belief in me on the days when I doubted myself.

My cat, Lemon, for the support, even though you did nothing and that bothers me, but thank you for being alive and being the cutest brother for me.

My thesis supervisor, Mr. Chandra Kusuma Dewa, for the invaluable guidance, patience, time, and knowledge generously shared throughout this research.

My classmates, Umar, Ikip, Jundi, Nalen, Vladi, Aura, Abdul, Arafi, Ferrel, and many more, who turned years of deadlines and late nights into something worth remembering.

My friends from the TEDx team, Abi, Aufa, Abrar, Byan, Reshita, Murni, and Caca, and all for the former team, Nida, Nita, and more, for showing me what it means to build something bigger than ourselves.

My friends at Marketive, Nabila, Syifa and more, for the lessons, the teamwork, and the growth we shared.

Last but not least, to myself, for staying strong when things were hard, for going right when you were left, for the patience to keep going, and for pouring everything into this. Thank you for refusing to give up, and for always trying to be the best version of yourself, for yourself.

And to everyone whose names cannot all be written in this report, but whose kindness, help, and presence carried me here. Thank you.

MOTTO

“Indeed, with every hardship comes ease.” (QS. Al-Insyirah:5-6)

FOREWORD

All praise and gratitude are due to Allah Subhanahu wa Ta'ala, for His mercy and guidance, which enabled the author to complete this thesis, titled "Development of SIBI and BISINDO Fingerspelling Learning Media Based on Cloud Voice Recognition." Salawat and salam are also conveyed to the Prophet Muhammad Sallallahu 'alaihi wasallam.

This thesis is submitted as one of the requirements for obtaining the Bachelor's degree (Sarjana Komputer) in the Informatics Study Program, Faculty of Industrial Technology, Universitas Islam Indonesia. The author is fully aware that its completion would not have been possible without the guidance, support, and assistance of many parties. Therefore, the author would like to express sincere gratitude to:

1. The Rector of Universitas Islam Indonesia, for the opportunity to study at this institution.
2. The Dean of the Faculty of Industrial Technology, Universitas Islam Indonesia.
3. The Head of the Informatics Study Program, Universitas Islam Indonesia.
4. Chandra Kusuma Dewa, as the thesis supervisor, for the invaluable guidance, patience, time, and knowledge generously shared throughout this research.
5. All lecturers and academic staff of the Informatics Study Program, for the knowledge and support given during the author's studies.
6. The author's beloved parents and family, for their endless prayers, love, and encouragement.
7. All friends and colleagues who cannot be mentioned individually, for their support and companionship.

The author realizes that this thesis is still far from perfect and welcomes any constructive criticism and suggestions for improvement. Finally, the author hopes that this work may bring benefit to readers and contribute to the development of knowledge.

Wassalamu'alaikum warahmatullahi wabarakatuh.

Yogyakarta, July 3rd, 2026

(Muhammad Fazli Ramadhani Sukma)

ABSTRACT

Communication barriers between the hearing public and the Deaf community remain a significant problem, partly because public knowledge of sign language is limited and conventional static media offer little interactivity for learning fingerspelling. This study develops an Android application as an interactive learning medium that translates spoken words into Indonesian sign language fingerspelling, covering both the SIBI (Sistem Isyarat Bahasa Indonesia) and BISINDO (Bahasa Isyarat Indonesia) standards, using cloud-based voice recognition. The application was developed using the Iterative Development Method comprising analysis, design, implementation, testing, and evaluation phases. To respect Deaf culture, the application is intentionally limited to translating fingerspelling for the letters A to Z. Development proceeded through two architectural scenarios: an initial on-device processing approach (Scenario 1) and a cloud-based approach using the AssemblyAI REST API (Scenario 2). The results show that migrating from on-device processing to the cloud-based architecture substantially reduced transcription latency, from approximately 83 seconds per word to about 3.35 seconds per word on the same device. Black Box Testing confirmed that all application components (state management, user interface, translation practice, dictionary, and evaluation quiz) functioned correctly with a 100% pass rate after iterative refinement, and an end-to-end integration test produced correct fingerspelling visualization for all 45 multi-character word inputs, while single-letter inputs (A–E) were not recognized. User Acceptance Testing with 19 respondents, conducted via Google Form using a five-point Likert scale across four aspects, yielded an overall acceptance score of 89.58%, categorized as Very Feasible (Sangat Layak).

Keywords: Learning Media, Sign Language, SIBI, BISINDO, Voice Recognition, Cloud ASR, AssemblyAI, Android.

GLOSSARY

Andragogy — The theory and practice of teaching adult learners, as distinct from pedagogy.

API (Application Programming Interface) — A set of rules and endpoints that lets one software application request services from another.

ASR (Automatic Speech Recognition) — Technology that automatically converts spoken language into written text.

AssemblyAI — A cloud-based speech-to-text service accessed through a REST API, used to transcribe Indonesian speech.

BISINDO (Bahasa Isyarat Indonesia) — The natural sign language that developed organically within the Indonesian Deaf community, with its own visual-spatial grammar.

Black Box Testing — A software testing method that verifies functionality from the user's perspective without examining the source code.

Cloud-based processing — Performing computation on remote internet-connected servers rather than on the local device.

ConstraintLayout — An Android layout manager that positions interface elements using constraints for responsive layouts.

Deaf community — The cultural and linguistic community of Deaf people who share a sign language and common identity.

Fingerspelling — Representing individual letters of the alphabet (A to Z) through distinct hand shapes to spell words letter by letter.

GERKATIN — Gerakan untuk Kesejahteraan Tunarungu Indonesia, a national organization of the Indonesian Deaf community.

JNI (Java Native Interface) — A framework that lets Kotlin or Java code call native C or C++ code; used to run the on-device Whisper model.

Kotlin — The programming language officially recommended by Google for Android development.

Latency — The time delay between an input (such as finishing speaking) and the corresponding output.

Likert scale — A rating scale (here 1 to 5) used to measure a respondent's level of agreement with a statement.

Logcat — A logging tool in Android Studio used to view application messages and

measure latency.

On-device inference — Running computation such as speech recognition locally on the device rather than in the cloud.

PCM 16-bit — Pulse-Code Modulation, an uncompressed digital audio format; audio here uses 16-bit PCM at 16 kHz, mono.

Polling — Repeatedly querying a server at intervals to check whether a requested task has completed.

Quantization — A model-optimization technique that lowers a neural network's numerical precision to reduce size and speed up inference.

RecyclerView — An Android component that efficiently displays large, scrollable lists or grids.

REST API — Representational State Transfer, a web API style using standard HTTP requests; used to access AssemblyAI.

Sample rate — The number of audio samples captured per second (here 16,000 Hz, or 16 kHz).

SDK (Software Development Kit) — A collection of tools for building applications; the minimum SDK here is Android 7.0 (API 24).

SDLC (Software Development Life Cycle) — The structured process of planning, designing, building, testing, and evaluating software.

SharedPreferences — An Android mechanism for storing small key-value data persistently on the device; used to save the quiz high score.

SIBI (Sistem Isyarat Bahasa Indonesia) — The government-standardized Indonesian sign system that follows the grammar of spoken Indonesian, used in formal special education.

SLB (Sekolah Luar Biasa) — Special-needs schools in Indonesia where SIBI is used for formal instruction.

Transformer — A neural network architecture underlying modern speech and language models.

UAT (User Acceptance Testing) — The final testing stage in which representative users evaluate whether the system is acceptable for use.

WAV — Waveform Audio File Format, a standard uncompressed audio format; the app records speech as a .wav file before uploading.

WER (Word Error Rate) — A metric for speech-recognition accuracy measuring the proportion of words transcribed incorrectly.

Whisper — An open speech-recognition model developed by OpenAI; its compiled C++ version (Whisper Tiny) was tested for on-device recognition in Scenario 1.

TABLE OF CONTENTS

TITLE PAGE i
SUPERVISOR ENDORSEMENT PAGE ii
EXAMINER ENDORSEMENT PAGE iii
AUTHENTICITY STATEMENT iv
DEDICATION v
MOTTO vi
FOREWORD vii
ABSTRACT viii
GLOSSARY ix
TABLE OF CONTENTS xii
LIST OF TABLES xiv
LIST OF FIGURES xv
CHAPTER I INTRODUCTION 1
1.1 Background 1
1.2 Problem Formulation 3
1.3 Scope and Limitations 3
1.4 Research Objectives 4
1.5 Research Benefits 4
1.6 Thesis Organization 4
CHAPTER II LITERATURE REVIEW 6
2.1 Sign Language in Indonesia 6
2.1.1 Sistem Isyarat Bahasa Indonesia (SIBI) 6
2.1.2 Bahasa Isyarat Indonesia (BISINDO) 6
2.2 Public Interest in Sign Language Learning 7
2.3 Voice Recognition Technology 7
2.3.1 AssemblyAI 8
2.4 Android Application Development 8
2.5 Iterative Development Method 9
2.6 Software Testing 9
2.6.1 Black Box Testing 9
2.6.2 User Acceptance Testing (UAT) 9
2.7 AI Technology for Deaf Communication 10
2.8 Related Research and Research Gap 10
CHAPTER III RESEARCH METHODOLOGY 13
3.1 Type of Research 13
3.2 Iterative Development Method 13
3.2.1 Analysis Phase 14
3.2.2 Design Phase 14
3.2.3 Implementation and Coding Phase 14
3.2.4 Testing Phase 15
3.2.5 Evaluation Phase 15
3.3 Scenario-Based Iterative Workflow 15
3.4 Scope and Limitations 17
3.5 Testing Parameters 18
3.5.1 Testing Parameters for Scenario 1 (Local Processing) 18
3.5.2 Testing Parameters for Scenario 2 (Cloud Processing) 18
3.5.3 User Acceptance Testing Design 18

3.6 Tools and Technologies 21
CHAPTER IV IMPLEMENTATION AND TESTING 22
4.1 Scenario 1 Implementation: Local/Offline AI 22
4.2 Scenario 1 Testing Results 22
4.3 Scenario 2 Implementation: Cloud API 23
4.4 Black Box Testing 25
4.4.1 Translation Module Testing (Core Feature) 25
4.4.2 Dictionary Module Testing (Education) 27
4.4.3 Gamification Module Testing (Quiz) 27
4.4.4 Error Handling Testing 29
4.4.5 Refinements and Retesting (Iteration 2) 29
4.5 Performance Testing (Processing Speed and Latency) 30
4.6 End-to-end Integration Testing (Accuracy) 31
4.7 User Acceptance Testing 36
4.7.1 Aspect A: Interface and Design (UI/UX) 36
4.7.2 Aspect B: Performance 37
4.7.3 Aspect C: Usability 37
4.7.4 Aspect D: Educational Value 37
4.7.5 Overall Recapitulation 38
4.7.6 Open-Ended Feedback and Suggestions 38
CHAPTER V RESULTS AND DISCUSSION 40
5.1 Comparison of the Two Scenarios 40
5.2 Discussion of Performance Results 41
5.3 Discussion of Integration and Accuracy Results 42
5.4 Discussion of UAT Results 42
5.5 Discussion of Ethical Consideration 43
CHAPTER VI CONCLUSION 46
6.1 Conclusions 46
6.2 Research Limitations 47
6.3 Recommendations for Future Development 48
REFERENCES 50
APPENDIX 53

LIST OF TABLES

Table 2.1 Summary of Related Research and Research Gap....11
Table 3.1 Likert Score Interpretation Ranges....20
Table 4.1 Scenario 1 Testing Results....22
Table 4.2 Translation Module Testing....26
Table 4.3 Dictionary Module Testing....27
Table 4.4 Gamification Module Testing....27
Table 4.5 Error Handling Testing....29
Table 4.6 Iteration 2....30
Table 4.7 Performance Testing....31
Table 4.8 Accuracy Testing....32
Table 4.9 Accuracy Testing....35
Table 4.10 UAT Respondent Demographics....36
Table 4.11 Aspect A: Interface and Design (UI/UX)....36
Table 4.12 Aspect B: Performance....37
Table 4.13 Aspect C: Usability....37
Table 4.14 Aspect D: Educational Value....37
Table 4.15 UAT Overall Recapitulation....38
Table 4.16 Summary of Open-Ended Feedback from Respondents....38
Table 5.1 Comparative of Two Scenarios....40

LIST OF FIGURES

Figure 3.1 The Iterative Process of Software Development 13
Figure 3.2 The Workflow Diagram 16
Figure 4.1 System Architecture of the Cloud-based Application 24
Figure 4.2 Example Output for the Spoken Word “Buku” 25
Figure 0.1 On-Device Whisper Integration via JNI (Scenario 1). 53
Figure 0.2 Audio Capture Configuration (16 kHz, Mono, PCM 16-bit) 54
Figure 0.3 Cloud Transcription Pipeline via AssemblyAI REST API (Scenario 2) 55
1Figure 0.4 Voice-to-Fingerspelling Mapping 57
Figure 0.5 SIBI/BISINDO Toggle Logic (Test Case TC-TR-05) 58
Figure 0.6 Quiz Generation and Score Persistence (Test Case TC-GM-05) 58
Figure 0.7 Sign Image Item Layout (85dp, ConstraintLayout) 59

CHAPTER I
INTRODUCTION

1.1 Background

Communication is a fundamental human right that enables every individual to participate fully in social life. Nevertheless, the lack of understanding of sign language among the general hearing public remains a major contributing factor to communication barriers with the Deaf community, a condition that frequently results in social exclusion (Wahyudin, 2020). According to a report published by the World Health Organization, more than 5% of the world's population experiences disabling hearing loss and requires rehabilitation (WHO, 2026). In Indonesia, people with hearing impairments face significant barriers to communication and social participation, which limits their access to information and their integration within the wider community (Wulandari et al., 2026). These figures underscore that bridging the communication gap is not merely a technological challenge but a matter of social inclusion that demands active participation from the hearing majority.

In the practice of everyday communication, two distinct sign language standards coexist in Indonesia. The first is Sistem Isyarat Bahasa Indonesia (SIBI), a system developed and standardized by the government for use in formal education within Sekolah Luar Biasa (SLB). SIBI follows the grammatical structure of spoken Indonesian, including its system of affixation. The second is Bahasa Isyarat Indonesia (BISINDO), a natural language that is constantly evolving and is the form most widely used in everyday interaction within the Deaf community itself (Gumelar et al., 2018). The coexistence of these two standards adds a further layer of complexity to the communication gap: for hearing individuals who wish to begin learning sign language, familiarity with a single standard is insufficient to represent how sign language is actually used across both formal and everyday contexts. Most existing digital tools, however, tend to focus on only one of these standards, which limits their usefulness.

Public interest in learning sign language exists and is growing. Recent research employing focus group discussions found that the primary motivations of hearing individuals to learn sign language include curiosity, social media exposure, social empathy, and emotional connection; however, the principal challenges they face are gesture complexity and limited access to formal learning tools (Onggo, 2025). Traditionally, sign language learning for the general public has relied on conventional media such as printed books or two-dimensional

static images. This approach carries significant drawbacks: sign language is a visual-spatial language in which transitions and body movement dynamics play a vital role, and static media are fundamentally unable to capture these dynamic dimensions. The result is a high learning curve and a learning experience that is not interactive (Mursalat et al., 2023).

Various prior studies have attempted to create digital tools for learning sign language, but gaps remain. Fatmawati et al. (2022) designed a voice-based Android application using Google's Speech-to-Text API and 3D animation technology; although interactive, it remains limited to the SIBI standard only and incorporates 3D technology that can cause performance problems on low-end Android devices. 'Isyatin Rodhiyah & Wulandari (2024) created a mobile learning application covering both SIBI and BISINDO standards as an improvement over traditional media, but the study did not include voice recognition and relied solely on manual navigation. Likewise, Nurpiena et al. (2021) developed an Android-based learning application evaluated against the ISO 25010 quality standard, but it also excluded voice recognition and was limited to a single standard.

Given the limited interactivity of conventional static media and the need to accommodate both sign language standards, an interactive approach that converts spoken input into a visual representation of fingerspelling offers a more accessible alternative for the general public. By allowing a user to speak a word and immediately see its fingerspelling visualization, such an approach addresses the lack of interactivity that characterizes books and two-dimensional images, while remaining lightweight enough to run on the low-end mobile devices commonly used in Indonesia. This research therefore develops an Android application that integrates both SIBI and BISINDO standards, replaces heavy 3D rendering with optimized lightweight 2D sequential animation, and offloads voice processing to the cloud using the AssemblyAI REST API. Beyond the engineering contribution, this thesis positions the application explicitly as an interactive learning medium and evaluates its acceptability through User Acceptance Testing (UAT). In short, whereas the companion paper centres on the engineering of the application like its architecture, latency, and functional reliability, this thesis centres on the application's role and acceptability as a learning medium; the two are designed to be read as complementary outputs of the same project, sharing the same system and data but differing in analytical focus.

It should be stated clearly from the outset that this application supports fingerspelling familiarization only like an introductory entry point, and does not claim to teach or translate full sign language. It should also be stated transparently that no Deaf individuals, BISINDO users, or sign-language instructors were directly involved in designing the application, creating

its visual assets, or evaluating it; the decision to limit the scope to fingerspelling is a precautionary, culturally cautious design choice rather than the result of community participation, and direct involvement of the Deaf community is acknowledged as essential for future development and validation.

1.2 Problem Formulation

Based on the background described above, the research questions of this study are formulated as follows:

a. How can an Android mobile application be designed and developed to facilitate the learning of SIBI and BISINDO fingerspelling alphabets for the general hearing public?
b. Which processing architecture on-device or cloud-based is suitable for voice recognition on low-to-mid-end Android devices, and how can the selected architecture be integrated efficiently?
c. How reliably does the developed system translate spoken word input into correct SIBI and BISINDO fingerspelling visualizations?
d. How acceptable is the developed application as a sign language learning medium, as evaluated through User Acceptance Testing?

1.3 Scope and Limitations

This research is bounded by the following limitations:

a. The application is developed exclusively for the Android platform, targeting the compatibility with low-to-mid-end devices used in Indonesia.
a. The translation feature is intentionally limited to fingerspelling of the letters A to Z and does not attempt to cover the full vocabulary or grammatical structures of either SIBI or BISINDO.
b. The scope limitations is grounded in an academic and ethical justification related to Deaf Culture, elaborated in Chapter III.
c. Performance testing was conducted under standard 4G/LTE network connectivity on a single device: Xiaomi Redmi Pad (MediaTek Helio G99, Android 14).
d. The scenario 1 (on-device) latency figures were measured on a debug build with an unquantizied Whisper Tiny model and therefore represent the tested configuration rather than an inherent lower bound of on-device ASR performance.

e. End-to-end accuracy testing was conducted with a single speaker under controlled conditions.

1.4 Research Objectives

The objectives of this research are as follows:

a. To develop an Android mobile application as an interactive learning medium for the SIBI and BISINDO fingerspelling alphabets, driven by voice recognition.
b. To determine a suitable processing architecture by iteratively comparing an on-device approach using the AssemblyAI REST API.
c. To measure and evaluate the latency and end-to-end translation reliability of the developed system.
d. To evaluate the application's acceptability as a sign language learning medium through User Acceptance Testing.

1.5 Research Benefits

The anticipated benefits of this research are threefold. Practically, the research provides an interactive, accessible, and inclusive sign language learning medium for members of the general public who wish to begin communicating with the Deaf community, lowering the entry barrier identified in prior studies. Academically, it contributes to the field of AI-powered mobile application development in the context of accessibility, demonstrating a working integration of cloud-based ASR with dual-standard sign language visualization that has not previously been addressed in the Indonesian context. Socially, it supports the inclusion of the Deaf community by improving sign language literacy among the hearing public, in line with the principle that inclusion requires effort from the hearing majority.

1.6 Thesis Organization

This thesis is organized into six chapters. Chapter I presents the research background, problem formulation, scope and limitations, objectives, benefits, and thesis organization. Chapter II presents the literature review, covering the theoretical foundations of SIBI and BISINDO, voice recognition technology, Android development, the Iterative Development Method, software testing, and an analysis of related research. Chapter III presents the research methodology, including the scenario-based iterative workflow, the ethical justification grounded in Deaf Culture, the testing parameters, and the tools used. Chapter IV presents the

implementation of both architectural scenarios and the complete testing results, including Black Box Testing, performance testing, end-to-end integration testing, and User Acceptance Testing. Chapter V presents the comparative analysis and discussion of all results. Chapter VI presents the conclusions, research limitations, and recommendations for future development.

CHAPTER II
LITERATURE REVIEW

2.1 Sign Language in Indonesia

2.1.1 Sistem Isyarat Bahasa Indonesia (SIBI)

Sistem Isyarat Bahasa Indonesia (SIBI), is a formalized and standardized system of signing developed through the initiative of the Indonesian government's Ministry of Education and Culture. The SIBI system is designed to be used in formal education settings in the Sekolah Luar Biasa (SLB) and follows the grammar structure of the spoken language of Indonesia. As a standardized system of signing, SIBI has a standardized sign vocabulary that can serve as a guide for developing digital learning materials (Wahyudin, 2020).

In the SIBI system of fingerspelling, there is use of stationary hand signs that correspond to all letters of the Latin alphabet from A to Z. This characteristic makes it easier for people to learn SIBI because it will be easy for them to correlated hand shapes to particular letters of the Latin alphabet.

2.1.2 Bahasa Isyarat Indonesia (BISINDO)

Bahasa Isyarat Indonesia (BISINDO) is the natural sign language that has evolved organically within the Indonesian Deaf community. Unlike SIBI, which was constructed top-down, BISINDO emerged from the daily social interactions of Deaf individuals and constitutes a living linguistic and cultural heritage. Research on the construction of meaning within the Deaf community organization GERKATIN demonstrates that BISINDO functions not merely as a communication tool but as a core marker of Deaf cultural identity (Gumelar et al., 2018). BISINDO possesses its own grammar, which is highly complex in the visual-spatial modality, encompassing facial expression, intensity of movement, the use of three-dimensional signing space, and other non-manual features.

Wahyudin, (2020) reflecting on the experiences of sign language speakers in Yogyakarta, documents how the marginalization of Deaf culture is linked to power asymmetries in inter-language relations dominated by hearing perspectives. Because BISINDO is an integral part of Deaf culture, its development and teaching should ideally involve or be led by the Deaf community itself, in order to preserve the meaning of the language and to avoid misuse of the language and culture. This principle, also emphasized by sign language institutions such as the

Language Institute of the Faculty of Humanities at Universitas Indonesia (LBI FIB UI, 2021), directly informs the ethical scope of the present research.

2.2 Public Interest in Sign Language Learning

Onggo, (2025), using a descriptive qualitative approach through focus group discussions with informants from student, private-employee, and creative-industry backgrounds, found that primary motivations for learning sign language include curiosity, social media exposure, social empathy, and emotional connection. Learning methods vary from digital platforms to structured community-led classes, but the main challenges remain gesture complexity and limited access to formal learning tools. The study concludes that cross-sector collaboration is essential to expand access and support for sign language learning as part of inclusive communication efforts in Indonesia. This social imperative is reinforced by Murni et al. (2024), who document the vital role of Indonesian sign language in building communication and social integration for Deaf children, underscoring that improved sign language literacy among the hearing public carries direct social value. Similarly, research on the development of learning media for Deaf students confirms that static, non-interactive materials create barriers, and that visual, interactive media significantly improve engagement (Mursalat et al., 2023). These findings position accessible, interactive digital learning media as a concrete response to a documented social need.

2.3 Voice Recognition Technology

Voice recognition, or automatic speech recognition (ASR), is a technology that enables computer systems to identify and interpret human spoken language into a machine-processable format, typically text. The field has advanced dramatically with the development of deep learning; a systematic review by Nassif et al. (2019) documents the transition from traditional Hidden Markov Model approaches to deep neural network architectures that handle variations in accent, speaking rate, and environmental conditions with substantially higher accuracy. A landmark contribution is the Whisper model of Radford et al. (2023), which demonstrates that large-scale weakly supervised training on 680,000 hours of multilingual audio produces speech recognition systems that generalize well to standard benchmarks and approach human-level accuracy and robustness without dataset-specific fine-tuning. Whisper is also available as an open model that can be compiled for on-device inference (Whisper C++), which is the configuration evaluated in Scenario 1 of this research.

For deployment, a practical question is whether to run ASR on-device or in the cloud. A comparative analysis of speech-to-text APIs in the context of supporting the Indonesian Deaf community covering Deepgram, Google, and Whisper AI, found that Whisper achieved the lowest word error rate but at lower transcription speed, while Deepgram offered the best balance of accuracy and speed (Handayani et al., 2025). These trade-offs in accuracy, speed, and accessibility frame the architectural comparison conducted in this research.

2.3.1 AssemblyAI

AssemblyAI is a cloud-based AI platform providing audio transcription services through a REST API, employing deep learning models trained on large-scale datasets. The platform supports multiple languages, including Bahasa Indonesia, which is a prerequisite for this research. Compared to alternatives such as Google Cloud Speech-to-Text and Microsoft Azure Speech, AssemblyAI offers a simple REST workflow (audio upload, polling, and transcript retrieval) with minimal integration overhead, making it well suited to native mobile integration (AssemblyAI, 2026). It should be noted that the selection of AssemblyAI in this research was made on these practical grounds rather than on a claim of superior accuracy; a cross-provider accuracy benchmark was not conducted.

2.4 Android Application Development

Android is a Linux-based mobile operating system developed by Google that dominates the global smartphone market with a share exceeding 68% (Stat Counter, 2026), making it the natural target platform for an application intended to reach the broadest possible Indonesian audience. Native Android development in this research uses Kotlin, the language officially recommended by Google, which offers concise syntax, null safety, and first-class coroutine support for asynchronous operations such as network calls to a cloud API ((Android Developers, 2023). The user interface is defined in XML using ConstraintLayout, a layout manager that renders complex, responsive interfaces without deeply nested view hierarchies. In this research, ConstraintLayout with fixed 85dp image dimensions is used to optimize the sequential rendering of fingerspelling images, achieving an average visual render time of 0.01 seconds.

2.5 Iterative Development Method

The Iterative Development Method is a Software Development Life Cycle (SDLC) paradigm in which the system is created step by step through repeated cycles of planning, design, implementation, and testing (Bolung & Tampangela, 2017). The historical roots of iterative and incremental development trace back decades, long predating the modern Agile movement (Larman & Basili, 2003). In contrast to the linear Waterfall approach, the iterative technique can incorporate changes in requirements and architectural design that arise during development, making it suitable for exploratory work, particularly work involving new technologies such as cloud-based ASR APIs whose real-world behavior on target devices can only be validated empirically. A comparative study of development models for final-year projects similarly concludes that iterative and agile models offer superior flexibility for handling periodic redevelopment (Paksi et al., 2023). This flexibility proved decisive in the present research, where the initial architectural assumption (on-device inference) failed empirically and had to be replaced mid-project.

2.6 Software Testing

2.6.1 Black Box Testing

Black Box Testing is a software testing method that verifies system functionality from the end-user perspective without examining internal code logic, evaluating functional requirements such as interface behavior, navigation, connections, and performance (Nidhra & Jagruthi, 2012) For Android applications specifically, Black Box approaches have been demonstrated as effective in validating learning applications (Saman & Indah Ratnasari, 2022) and, in decision-table form, in systematically covering test scenarios and edge cases in mobile applications (Handayanto & Nuryasin, 2024a). In this research, Black Box Testing is applied to all three application modules and to error handling, across two iterations.

2.6.2 User Acceptance Testing (UAT)

User Acceptance Testing is the final stage of testing in which actual or representative users evaluate the system to verify that it meets requirements and is acceptable for real-world use, focusing on user experience and fitness for purpose rather than technical correctness alone (Black 1964-, 2007) UAT is particularly relevant for learning media, where acceptance by the target learner population is itself a research outcome. Precedents in the sign language learning domain include Nurpiena et al. (2021), who reported a usability score of 83.2% under the ISO

25010 framework, and Molina et al. (2024), whose sign language learning application achieved an acceptance rate of over 80% in user evaluation. In this research, UAT is administered through Google Form to respondents who directly tested the application, using a five-point Likert scale (1 = Strongly Disagree to 5 = Strongly Agree) across four evaluation aspects.

2.7 AI Technology for Deaf Communication

Broader literature confirms the accelerating role of AI in facilitating communication for the Deaf and hard-of-hearing. ZainEldin et al. (2024) provide a comprehensive review of artificial intelligence, deep learning, and machine learning applied to sign language interpretation, speech recognition, and text-to-speech synthesis. Alam et al. (2024) in a systematic review covering 2012–2023, document that smartphones are now computationally capable of running sophisticated sign language detection and interpretation models. Concrete deployments such as Efhamni, a deep learning-based Saudi Sign Language recognition application (Al Khuzayem et al., 2024) and continuous sign language recognition systems with intonation-colored speech synthesis (Amangeldy et al. 2023) demonstrate the feasibility of consumer-device deployment. Notably, these works concentrate on sign-to-text recognition (the inverse direction of the present research) and none address the dual SIBI/BISINDO standard in the Indonesian context.

2.8 Related Research and Research Gap

Table 2.1 summarizes the prior studies most directly relevant to this research and positions the present study against them.

Table 2.1 Summary of Related Research and Research Gap

Author (Year)	Platform	Standard	Voice Input	Output	Limitations Relative to This Study	Deaf/Expert Involvement
Nugraha (2016)	Desktop (Flash)	BISINDO	None	2D interactive	Desktop only; no voice input	Not reported
Fatmawati et al. (2022)	Android	SIBI	Google STT	3D animation	Single standard; 3D performance risk	Not reported
Rodhiyah & Wulandari (2024)	Android	SIBI + BISINDO	None	Static media	No voice recognition; manual navigation	Not reported
Nurpiena et al. (2021)	Android	BISINDO	None	Learning media	No voice input; single standard	Not reported
Ardiani & Fachrie (2023)	Android	SIBI + BISINDO	None	AR video	No voice input; AR-dependent	Not reported
Permana & Sutopo (2023)	Android	SIBI	None (camera)	Detection	Sign-to-text direction; 77% accuracy	Not reported
Hikmatia & Zul (2021)	Android	BISINDO	None (camera)	Voice output	Inverse direction; 54.8% model accuracy	Not reported
Kamil Siregar et al. (2021)	Android	BISINDO	None	Animation	No voice input; single standard	Not reported
Molina et al. (2024)	Mobile	Ecuadorian SL	None	3D animation	Not Indonesian context	Not reported
David et al. (2024)	Review	Multiple	—	—	Identifies need; no implementation	Not reported
This study (2026)	Android	SIBI + BISINDO	AssemblyAI cloud	2D sequential	—	Not involved (stated)

Table 2.1 warrants a closer, critical reading of each prior study, since each contributes a useful idea yet leaves a specific weakness that the present work addresses. Nugraha (2016) was among the earlier efforts to present BISINDO through interactive two-dimensional media; its strength lies in this interactivity, but it runs only on a desktop (Flash) platform, accepts no voice input, and covers a single standard. Nurpiena et al. (2021) is methodologically strong, as it evaluates a BISINDO learning application against the ISO 25010 quality standard and reports a usability score of 83.2%; however, it is limited to one standard and offers no voice-based input. Kamil Siregar et al. (2021) improves on static media by using animation for BISINDO, but it likewise lacks voice input and dual-standard coverage.

Among the voice-enabled and both-standard works, the limitations become more specific. Fatmawati et al. (2022) is the closest in spirit to this research, since it is voice-driven

through the Google Speech-to-Text service; its weakness, however, is that it supports only SIBI and relies on 3D animation, which introduces a performance risk on the low-to-mid-end devices that this study targets. Rodhiyah & Wulandari (2024) and Ardiani & Fachrie (2023) both cover the two standards, which is their key strength, yet neither accepts voice input: the former depends on manual navigation of static media, and the latter depends on augmented-reality video, which adds hardware and complexity requirements. Two further works operate in the opposite direction to this study, converting signs into text or voice rather than speech into fingerspelling; Permana & Sutopo (2023) achieves camera-based SIBI detection but at only 77% accuracy and for a single standard, while Hikmatia & Zul (2021) proposes a gesture-to-voice system whose reported model accuracy is low at 54.8%. Finally, Molina et al. (2024) reports strong user acceptance, above 80%, but targets Ecuadorian Sign Language and therefore does not apply to the Indonesian context, and David et al. (2024) is a review that clearly identifies the need for such tools but provides no implementation.

Taken together, this critical review shows that the strengths of prior work are scattered across separate systems, while their weaknesses recur along the same axes: single-standard coverage, the absence of voice input, reliance on heavy 3D or AR rendering, operation in the inverse direction, or evaluation outside the Indonesian context. No single prior study combines the four characteristics that define this research: first, coverage of both Indonesian sign language standards, SIBI and BISINDO; second, cloud-based voice recognition as the input modality; third, lightweight two-dimensional sequential visualization suitable for low-to-mid-end devices; and fourth, explicit evaluation as a learning medium through User Acceptance Testing.

The present study addresses this combined gap by developing SiBisindo, which integrates the AssemblyAI cloud speech recognition service with sequential 2D fingerspelling visualization for both standards on the Android platform, thereby consolidating the individual strengths of prior work while avoiding their recurring weaknesses. It is also worth noting that none of the reviewed studies report the involvement of Deaf users or experts; the present study likewise did not involve the Deaf community, and it states this explicitly as a limitation to be addressed in future validation rather than leaving it unmentioned.

CHAPTER III
RESEARCH METHODOLOGY

3.1 Type of Research

This research is applied research in the field of software engineering, conducted with an experimental, scenario-based approach. It develops a technological artifact (an Android application functioning as interactive sign language learning media) and evaluates that artifact empirically through performance, functionality, integration, and user acceptance testing.

3.2 Iterative Development Method

This research employs the Iterative Development Method (Bolung & Tampangela, 2017; Larman & Basili, 2003), consisting of five phases executed across two architectural scenarios. The iterative process is illustrated in Figure 3.1.

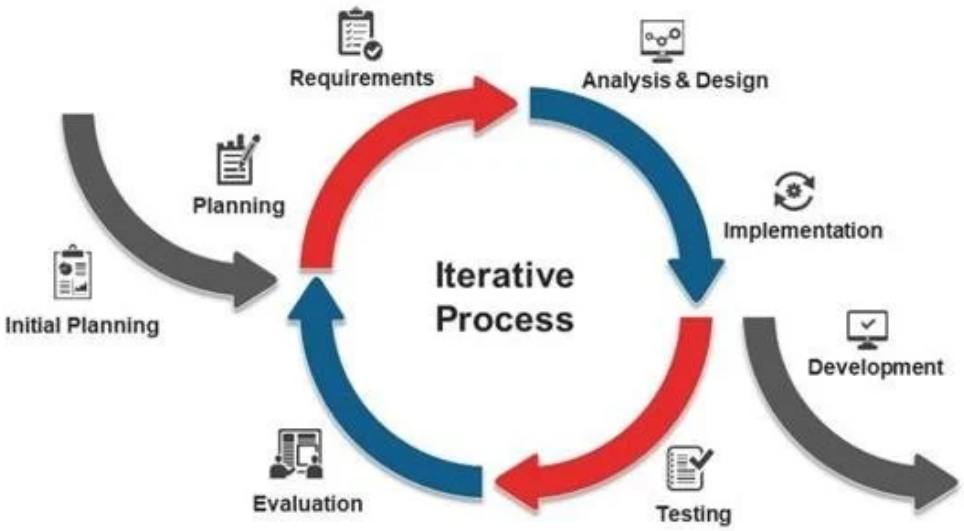


Figure 3.1 The Iterative Process of Software Development

The Iterative Development Method was selected over alternative software process models because it best matched the characteristics of this project. The central technical question, whether efficient voice recognition was feasible on low-to-mid-end devices, could not be answered from specification alone and had to be validated empirically; an iterative model, in which each cycle produces a working build that is tested and evaluated before the next cycle, directly supports this kind of experimental, evidence-driven development. A linear Waterfall model was unsuitable because it assumes that requirements and architecture are fixed

at the outset and does not accommodate the mid-project architectural change from on-device (Scenario 1) to cloud (Scenario 2) that the empirical results made necessary. Agile Scrum, with its emphasis on team roles, sprint ceremonies, and continuous client collaboration, is oriented toward multi-person teams and evolving stakeholder requirements, neither of which applied to this single-developer academic project with fixed research objectives. The Prototype model, although iterative in spirit, centres on progressively refining the user interface against user feedback, whereas the core uncertainty here was architectural and performance-related rather than interface-related. The chosen approach is consistent with prior work that adopts iterative and incremental development for comparable exploratory software projects (Bolung & Tampangela, 2017; Larman & Basili, 2003; Paksi et al., 2023).

3.2.1 Analysis Phase

The analysis phase specifies the requirements of the interactive translation software and designs a system architecture capable of processing voice input while maintaining low latency on low-to-mid-end devices. This phase includes a literature study of the SIBI and BISINDO standards, available ASR technologies, and prior related applications.

3.2.2 Design Phase

The design phase develops the responsive user interface and the system architecture. The application is divided into three educational modules: Translation Practice (the core feature, converting spoken words into sequential fingerspelling visualization), Dictionary Reference (a browsable A–Z reference for both standards), and Evaluation Quiz (a gamified module reinforcing recognition of fingerspelling images). Interface design prioritizes accessibility and ease of use for first-time learners.

3.2.3 Implementation and Coding Phase

Code is written in the native Android environment using Kotlin and XML. Critically, this phase underwent two iterations across different computational architectures, offline (on-device) and cloud (online), to determine empirically the most responsive solution. The two iterations are formalized as Scenario 1 and Scenario 2 below.

3.2.4 Testing Phase

The testing phase evaluates UI/UX functionality and translation reliability through Black Box Testing, which assesses functional requirements from the end-user perspective without examining internal code logic (Nidhra & Jagruthi, 2012); Handayanto & Nuryasin, (2024)), together with performance testing, end-to-end integration testing, and User Acceptance Testing.

3.2.5 Evaluation Phase

The evaluation phase assesses the effectiveness of the developed architecture and the functionality of the user experience based on all testing results. Evaluation outcomes from each iteration feed directly into the next, and ultimately into the selection of the final architecture.

3.3 Scenario-Based Iterative Workflow

The central engineering problem of this research is efficient voice recognition processing on low-end mobile devices without crashing the application. The implementation was therefore split into two scenarios, as illustrated in the research workflow in Figure 3.2.

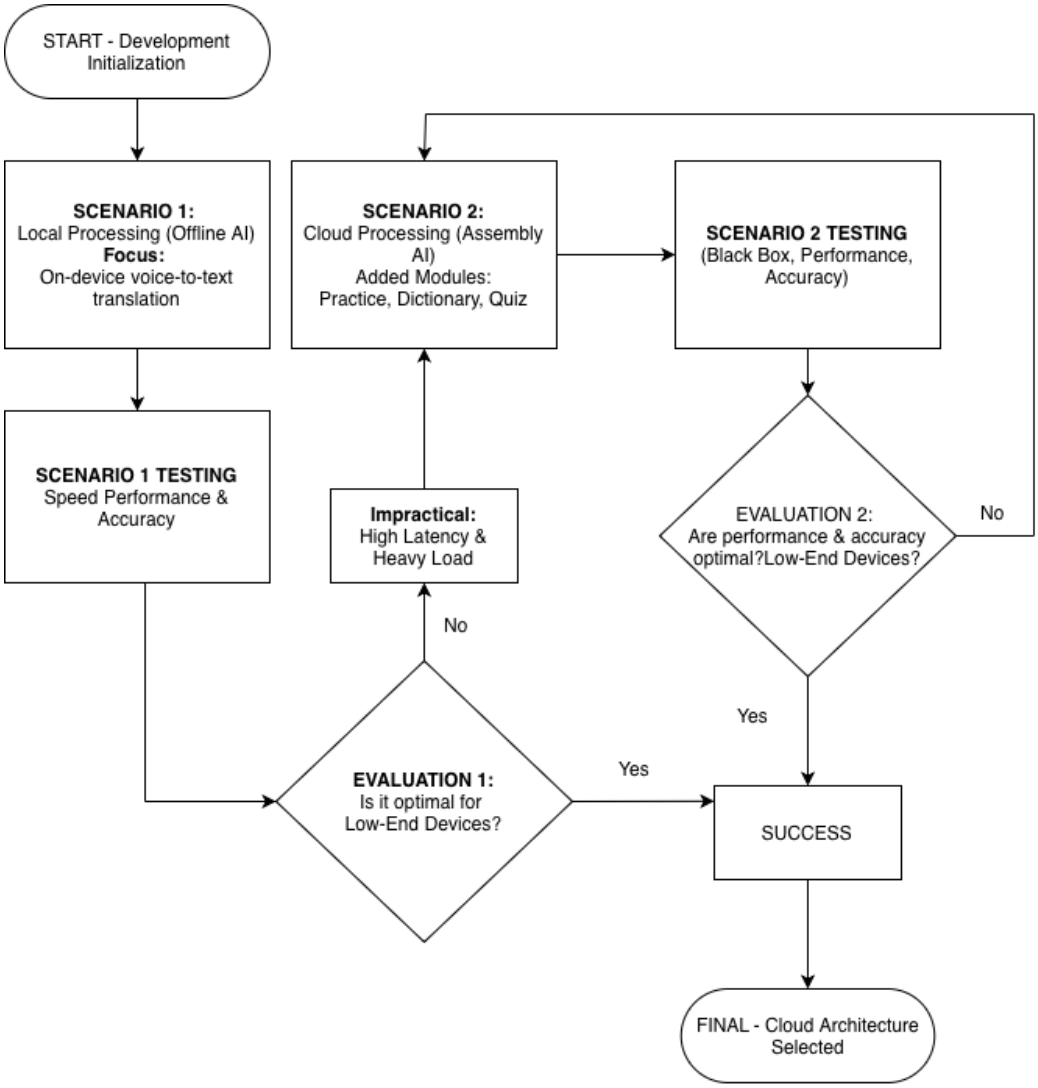


Figure 3.2 The Workflow Diagram

Scenario 1: Local/Offline AI Processing. The application is first implemented as a single-feature prototype using a local AI model (Whisper C++, based on Radford et al. (2023) for speech-to-text translation. All computation, from audio extraction to text conversion, is completed on-device through 4 CPU cores. The test device is a Xiaomi Redmi Pad (MediaTek Helio G99, Android 14), with measurements taken on a debug build using the unquantized Whisper Tiny model.

Scenario 2: Cloud Processing. Following the empirical failure of Scenario 1 under the tested configuration, the architecture is entirely revamped: voice processing is delegated to the cloud-based AssemblyAI REST API, and the application is expanded into its full three-module form. It must be emphasized from the outset that this architectural shift is an engineering

compromise made under project time and configuration constraints; it does not establish that optimized on-device inference is impossible, a point revisited in the limitations.

3.4 Scope and Limitations

In creating technology that directly interacts with a linguistic minority culture, ethical justification is as important as the technology created. The scope and limitations of this research are defined as follows.

a. Device Accessibility Focus: the application is intended to be inclusively accessible to the general public, including users of low-end smartphones. The application therefore performs no 3D rendering and utilizes cloud-based technology to keep the on-device footprint light.
b. Translation Feature Limitation: the application's visual output is limited to fingerspelling from A to Z.
c. Academic and Ethical Justification: SIBI follows the grammar of spoken Indonesian and is standardized, making its fingerspelling safe to represent with static images. BISINDO, by contrast, is the natural language of the Deaf community with a highly complex grammar in the visual-spatial modality: facial expression, intensity of movement, and non-manual features (Wahyudin, 2020). For hearing developers, the forceful translation of BISINDO vocabulary from spoken words into simple static 2D images is highly risky in the context of overstepping ethical boundaries with the language and culture. Limiting functionality to fingerspelling is the safest and most academically defensible way of bridging the language gap without diluting the complexity of the natural sign language.
d. Pedagogical scope of the application: To avoid overstating the system's capability, three related but distinct tasks are distinguished. Fingerspelling familiarization refers to recognizing and reproducing the manual alphabet (A–Z) used to spell words letter by letter; this is the introductory scope addressed by the present work. Sign language learning is a broader process that involves acquiring vocabulary, grammar, and the visual-spatial and non-manual features of a natural sign language such as BISINDO. Sign language translation, in turn, refers to converting complete utterances between a spoken language and a sign language while preserving meaning and grammatical structure. This research is deliberately positioned at the first level only: it supports

fingerspelling familiarization as an entry point, and does not claim to teach or translate full sign language

e. Statement on Deaf community involvement: It must be stated transparently that no Deaf individuals, BISINDO users, or sign-language instructors were directly involved in the design of this application, in the selection or creation of its visual assets, or in its evaluation. The cultural caution adopted in this study, limiting the scope to fingerspelling rather than full sign language, is therefore a precautionary design decision rather than the result of community participation. Direct involvement of the Deaf community and sign-language experts is acknowledged as an essential requirement for future development and validation.

3.5 Testing Parameters

3.5.1 Testing Parameters for Scenario 1 (Local Processing)

Because Scenario 1 had only a single core feature (voice-to-text translation) and was at a nascent stage, testing was restricted to two parameters: Performance Testing (Speed), evaluating the time required by the device to translate voice into text; and Accuracy Testing, evaluating the correctness of the transcribed text relative to the voice input.

3.5.2 Testing Parameters for Scenario 2 (Cloud Processing)

With the application fully developed in its three-module form, four sets of tests were conducted. (1) Black Box Testing checks the functionality of the UI/UX, the buttons, and the application state. (2) Performance Testing measures the speed of the API round-trip and of the visualization rendering. (3) Accuracy/Integration Testing verifies the end-to-end integration between the application and the AssemblyAI service using 50 standard words; it is explicitly framed as an integration test rather than a measurement of the ASR engine's intrinsic recognition accuracy, since transcription is handled entirely by the third-party service. (4) User Acceptance Testing evaluates the application's acceptability as a learning medium.

3.5.3 User Acceptance Testing Design

UAT is administered via Google Form to 19 respondents drawn from the general hearing public, each of whom directly operates all three modules of the application before responding. The instrument uses a five-point Likert scale, where 1 = Strongly Disagree (Sangat Tidak Setuju), 2 = Disagree (Tidak Setuju), 3 = Neutral (Netral), 4 = Agree (Setuju), and 5 = Strongly

Agree (Sangat Setuju). The statements are organized into four evaluation aspects comprising ten statements in total: Interface and Design (UI/UX) with two statements, Performance with two statements, Usability with three statements, and Educational Value with three statements. A final open-ended item collects critique and suggestions for development. The complete list of statements is presented together with the results in Section 4.7. The questionnaire was administered in Indonesian to suit the respondent population; English translations are used in this document for readability.

The 19 respondents were obtained through non-probability sampling, specifically a combination of purposive and convenience sampling: participation was invited from members of the general hearing public, who are the intended users of the application and who were accessible to the researcher. The sample size was not derived from a population formula such as Slovin, because the objective of the UAT is to assess the perceived acceptability and feasibility of the application among its target users rather than to produce statistically generalizable inferences about a defined population. For acceptance and usability evaluation of this kind, a relatively small sample is a common and accepted practice, since the aim is formative feedback on the product rather than population-level estimation. The participants are considered appropriate representatives because the application is explicitly designed for the general hearing public who wish to begin learning fingerspelling, and the respondents were drawn from exactly this group; moreover, each respondent operated all three modules before responding, so their ratings reflect direct, hands-on experience with the application. It should nonetheless be acknowledged that the number and demographic diversity of the respondents remain a limitation, and that the sample does not include members of the Deaf community; a larger and more diverse sample is therefore recommended for future validation.

For each statement, the acceptance index is computed as the ratio of the obtained score to the maximum possible score, following the Likert percentage-index method:

$$Score\ (\%) = \frac{\sum(\mathrm{f_k} \times \mathrm{k})}{(N \times 5)} \times 100\% \qquad (3.1)$$

Where k is the scale weight (1 to 5), f_k is the number of respondents selecting weight k for that statement, and N is the total number of respondents. Equation (3.1) yields the acceptance index of a single statement. The score of an evaluation aspect is computed as the mean of the indices of the statements that belong to that aspect, as shown in Equation (3.2):

$$Aspect\ Score\ (\%) = \frac{(I_1 + I_2 + \ldots + I_m)}{m} \quad (3.2)$$

Where I_1 to I_m are the statement indices obtained from Equation 3.1 and m is the number of statements in the aspect (m = 2 for Aspects A and B, and m = 3 for Aspects C and D). The overall acceptance score is computed as the mean of the ind ices of all ten statements, as shown in Equation 3.3:

$$Aspect\ Score\ (\%) = \frac{(I_1 + I_2 + \ldots + I_{10})}{10} \quad (3.3)$$

As a worked example, statement A1 was rated by the 19 respondents as follows: none selected weights 1, 2, or 3; 11 selected weight 4 (Agree); and 8 selected weight 5 (Strongly Agree). Applying Equation 3.1, the index is [(0×1) + (0×2) + (0×3) + (11×4) + (8×5)] / (19×5) × 100% = (44 + 40) / 95 × 100% = 84 / 95 × 100% = 88.42%. The same procedure is applied to every statement. For Aspect A, averaging the indices of A1 (88.42%) and A2 (90.53%) with Equation 3.2 gives 89.47%, and averaging the indices of all ten statements with Equation 3.3 gives the overall acceptance score of 89.58%.

The resulting percentage, whether for an individual statement, an aspect, or the overall instrument, is interpreted using the conventional Likert interpretation ranges shown in Table 3.1.

Table 3.1 Likert Score Interpretation Ranges

Score Range (%)	**Interpretation (Category)**
0 – 20	Sangat Tidak Layak (Very Unfeasible)
21 – 40	Tidak Layak (Unfeasible)
41 – 60	Cukup Layak (Moderately Feasible)
61 – 80	Layak (Feasible)
81 – 100	Sangat Layak (Very Feasible)

This scoring approach follows established practice in usability evaluation of mobile learning applications (Hikmatia A. E. & Ihsan Zul, 2021; Nurpiena et al., 2021).

3.6 Tools and Technologies

a. Programming language: Kotlin (application logic), XML (layout definitions).
b. IDE: Android Studio, including Logcat for millisecond-precision latency measurement.
c. Voice recognition: Whisper C++ with Whisper Tiny model (Scenario 1); AssemblyAI REST API (Scenario 2).
d. Layout engine: ConstraintLayout with fixed 85dp image dimensions for optimized sequential rendering.
e. Local storage: SharedPreferences for quiz high-score persistence.
f. Build configuration: minimum SDK 24 (Android 7.0) for low-to-mid-end device coverage, with the HTTP client OkHttp 4.12.0 for REST communication, Glide 4.16.0 for image rendering, and Lottie 6.0.0 for UI animation.
g. UAT instrument: Google Form with a five-point Likert scale (10 statements across four aspects).
h. Development machine: MacBook Air (Apple M2); test device: Xiaomi Redmi Pad (MediaTek Helio G99, Android 14), low-to-mid-end class, tested over standard 4G/LTE connectivity.
i. Visual assets: 2D SIBI and BISINDO fingerspelling images created by the author based on references from publicly available websites; assets were not validated by sign-language experts or the Deaf community.

CHAPTER IV
IMPLEMENTATION AND TESTING

4.1 Scenario 1 Implementation: Local/Offline AI

In Scenario 1, the software was created as a single-feature application using a local AI model (Whisper C++) for speech-to-text translation (Radford et al., 2023). All computation involved in the entire process, from extraction of audio to conversion into text, was completed by the device itself (on-device) through 4 CPU cores, using the Whisper Tiny model with four threads. All measurements were taken on a debug build that did not apply compile-time or run-time optimizations, and the model was not quantized; this configuration caveat is essential for interpreting the results below.

4.2 Scenario 1 Testing Results

Inference times were measured using Android Studio's Logcat. Sample logs from the performance test involving basic Indonesian words are shown in Table 4.1.

Table 4.1 Scenario 1 Testing Results

No.	Target Word	Inference Start	Transcription End	Inference Duration	Result	Status
1.	Ibu (mother)	11:29:05.660	11:30:14.719	69.05 s	"Ibu"	Valid
2.	Buku (book)	11:31:47.557	11:33:20.768	93.21 s	"Buku."	Valid
3.	Makan (to eat)	11:33:30.631	11:35:18.188	107.55 s	"[MENIKA]"	Invalid
	Average (real device)			~83 s/word		
	Average (emulator)			~124 s/word		

Empirical testing of Scenario 1 revealed three critical issues. First, extreme latency: transcription of a single three-to-five-letter word required approximately 83 seconds on the real device (Xiaomi Redmi Pad) and approximately 124 seconds inside the Android Studio emulator. Such latency is fundamentally incompatible with an interactive learning medium in which a user speaks a word and waits for visual feedback. Second, memory and processor overload: Logcat analysis showed repeated Garbage Collection events (e.g., NativeAlloc

concurrent mark compact GC freed), indicating that local inference imposed unsustainable resource loads on the low-powered target device. Third, accuracy degradation: during extended testing, the model began producing transcription hallucinations, as exemplified by the "Makan" → "[MENIKA]" failure in Table 4.1.

As a result, the on-device inference process described in Scenario 1 was deemed impractical under the tested configuration, prompting the shift to the cloud processing approach of Scenario 2. This decision should not be interpreted as establishing the intrinsic impossibility of on-device inference; rather, it is an engineering compromise under the project's development constraints. An optimized implementation (a release build with a quantized model) was not analyzed and remains future work.

4.3 Scenario 2 Implementation: Cloud API

In Scenario 2, the architecture was revamped to delegate voice processing to the cloud-based AssemblyAI REST API; the resulting application is named SiBisindo. The overall system architecture, including its input (the user's spoken word) and output (the sequential fingerspelling visualization), is illustrated in Figure 4.1. AssemblyAI was chosen over alternatives such as Google Cloud Speech-to-Text and Microsoft Azure Speech on practical grounds rather than a claim of superior accuracy: it offers a simple REST workflow (audio upload, polling, transcript retrieval) with minimal integration overhead, and it supports Bahasa Indonesia, a prerequisite for this study. Trade-offs between cloud ASR services in accuracy, speed, and accessibility have been examined in prior comparative work for the Deaf community context (Handayani et al., 2025); a comparative benchmark across providers was not conducted here and is left as future work.

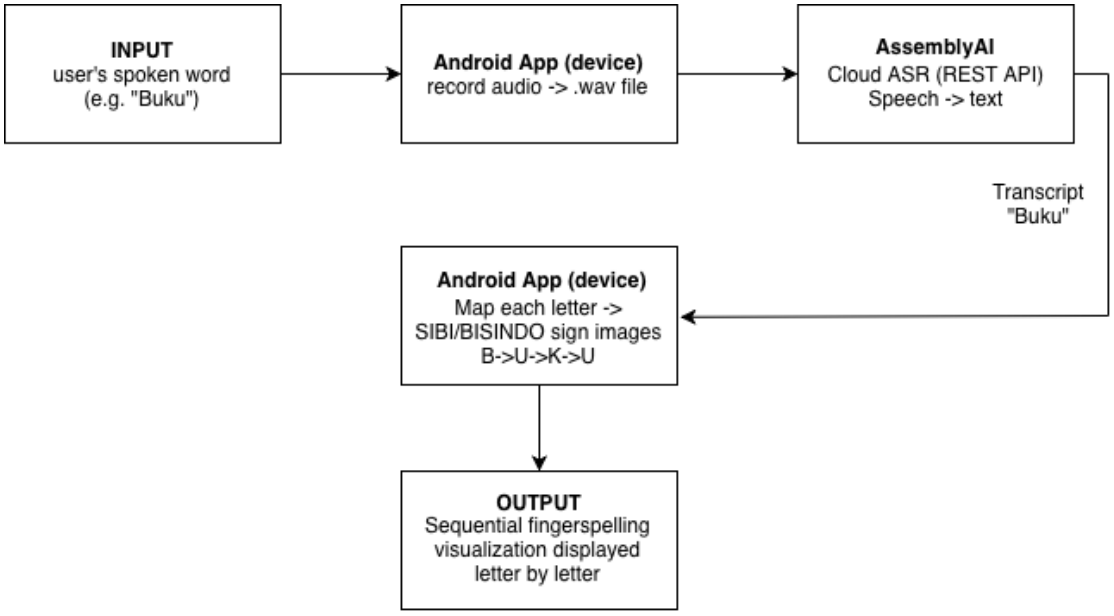


Figure 4.1 System Architecture of the Cloud-based Application

The system was extended with three fully functional modules: Translation Practice (core), Sign Dictionary Reference, and Gamified Evaluation Quiz. All 2D sign images are dimensioned at 85dp with ConstraintLayout ratios to optimize rendering. As an illustration of the workflow, when a user speaks "*Buku*", the system transcribes it and displays the fingerspelling images sequentially in the order B–U–K–U, as shown in Figure 4.2. During development, the Android Studio emulator was used for initial functional debugging of the interface and the cloud request flow; however, all latency measurements reported for Scenario 2 in Section 4.2 were taken on the physical device (Xiaomi Redmi Pad), so that the comparison against Scenario 1 in Section 5.1 uses device-based figures for both scenarios. Because the heavy processing in Scenario 2 occurs on the AssemblyAI server and over the network rather than on the local CPU, the choice of execution environment has only a minor effect on its latency, unlike the on-device Scenario 1 where emulator and device timings differ substantially.

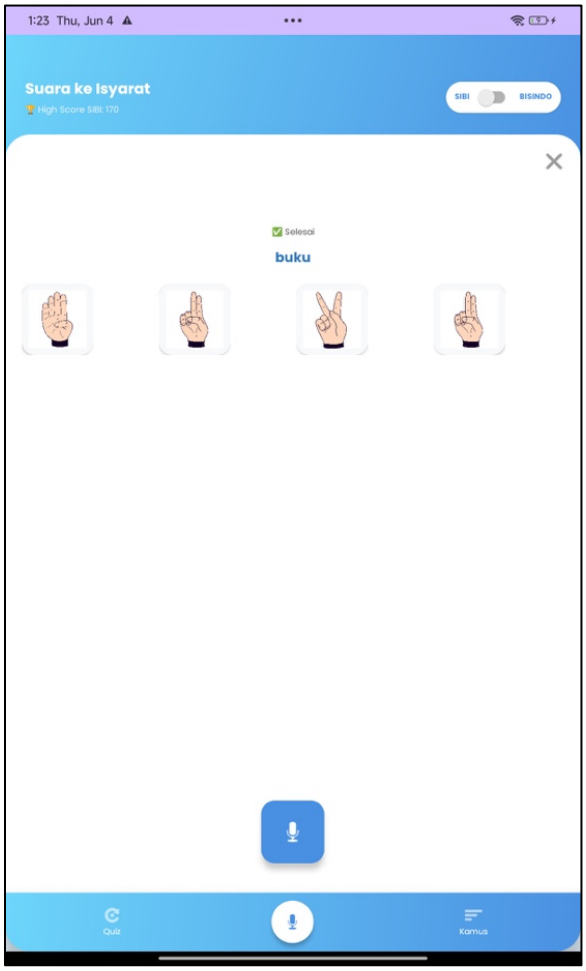


Figure 4.2 Example Output for the Spoken Word "*Buku*"

4.4 Black Box Testing

Black Box Testing was conducted on each application module independently (Translation, Dictionary, and Gamification) and on the application's error handling, evaluating functional behavior from the end-user perspective without examining internal code logic. Each module is reported in its own table together with the test steps performed, so that the coverage of each functional area is fully traceable.

4.4.1 Translation Module Testing (Core Feature)

The Translation module is the core feature of the application, responsible for capturing the user's voice, transmitting it to AssemblyAI, and rendering the resulting fingerspelling sequence. Its test cases and results are presented in Table 4.2

Table 4.2 Translation Module Testing

Test ID	Test Scenario	Test Steps	Expected Result	Status
TC-TR-01	Recording activation	Press the microphone (floating button) on the main page.	The microphone icon changes status (visual feedback); the application starts capturing audio.	Valid
TC-TR-02	Stop recording & request API	Release/re-press the microphone button after speaking.	A loading indicator appears, indicating the upload process to AssemblyAI is running.	Valid
TC-TR-03	Text output visualization	Speak a clear word (e.g., "Buku").	The transcribed text ("Buku") appears on the screen before the images appear.	Valid
TC-TR-04	Image output visualization (sequential)	Observe the appearance of the sign images after the text appears.	The sign images B-U-K-U appear one by one sequentially with a time delay, not simultaneously.	Valid
TC-TR-05	SIBI/BISINDO toggle feature	(1) Translate the word in SIBI mode. (2) Change the toggle to BISINDO. (3) Re-translate.	The visualization images change according to the selected standard (SIBI signs differ from BISINDO signs).	Invalid
TC-TR-06	Visualization reset	Press the "Delete"/"Reset" button after the translation is complete.	The text and visualization images disappear; the system is ready for new input.	Invalid

As shown in Table 4.2, four of the six test cases for the Translation module passed in the first iteration, confirming that recording activation, cloud transcription, sequential visualization, and text output behaved as expected. Two cases failed: TC-TR-05, in which switching the SIBI/BISINDO toggle did not re-render the already-displayed result, and TC-TR-06, in which the reset button did not fully clear the display. At this stage, the functional requirements for dynamic re-rendering and state reset were therefore not yet satisfied; both were resolved in Iteration 2 (Table 4.6).

4.4.2 Dictionary Module Testing (Education)

The Dictionary module provides a browsable A–Z reference of fingerspelling signs for both standards, supporting self-directed study. Its test cases and results are presented in Table 4.3.

Table 4.3 Dictionary Module Testing

Test ID	Test Scenario	Test Steps	Expected Result	Status
TC-DICT-01	Letter list display	Open the Dictionary menu from the Dashboard.	The A–Z letter list appears in a grid/list format; thumbnail images load smoothly.	Valid
TC-DICT-02	Category filter (SIBI/BISINDO)	Select the SIBI tab/option, then switch to BISINDO on the dictionary page.	The image list is updated according to the selected sign standard category.	Valid
TC-DICT-03	Scroll stability	Scroll the screen from letter A to Z quickly.	The application remains responsive, does not stutter (lag), and images are not misplaced.	Valid

Table 4.3 shows that all three test cases for the Dictionary module passed. The A to Z letter list displayed correctly for both standards, the SIBI and BISINDO category filter updated the images as expected, and scrolling remained stable. The functional requirements of the Dictionary module, namely reference browsing and standard filtering, were therefore fully satisfied in the first iteration, with no refinement required.

4.4.3 Gamification Module Testing (Quiz)

The Gamification module presents a guess-the-letter quiz that reinforces recognition of fingerspelling images, with scoring to encourage active practice. Its test cases and results are presented in Table 4.4.

Table 4.4 Gamification Module Testing

Test ID	Test Scenario	Test Steps	Expected Result	Status
TC-GM-01	Start game	Press the "Quiz" button.	The quiz page opens, the picture	Valid

			question appears, and the timer starts counting down.	
TC-GM-02	Correct answer logic	Select the answer button that matches the picture question.	The score increases (+10), a "Correct" notification appears, and the next question loads automatically.	Valid
TC-GM-03	Incorrect answer logic	Select the wrong answer button (distractor).	The score does not increase (remains the same), and an "Incorrect" notification appears.	Valid
TC-GM-04	Timer expiration function	Let the quiz run without answering until the time reaches 0 seconds.	The quiz automatically stops; a "Game Over" pop-up/page appears displaying the final score.	Valid
TC-GM-05	Score storage (persistence)	(1) Achieve a high score (e.g., 100). (2) Close the application. (3) Reopen the Quiz menu.	The highest score (High Score: 100) remains displayed on the quiz menu screen (saved in SharedPreferences).	Invalid

As presented in Table 4.4, four of the five test cases for the Gamification module passed, confirming that question generation, correct and incorrect answer scoring, and the timer functioned as intended. The single failure, TC-GM-05, occurred because the quiz high score was not retained after the application was closed and reopened, indicating that the score-persistence requirement was not yet satisfied. It was resolved in Iteration 2 by storing the score in SharedPreferences (Table 4.6).

4.4.4 Error Handling Testing

Beyond the three functional modules, the application's resilience to error conditions was tested to ensure it fails gracefully rather than crashing. The error handling test cases and results are presented in Table 4.5.

Table 4.5 Error Handling Testing

Test ID	Test Scenario	Test Steps	Expected Result	Status
TC-ERR-01	Connection lost	Turn off Data/WiFi, then try the Voice Translator feature.	The application does not crash; a Toast/dialog appears: "Internet connection is unavailable".	Valid
TC-ERR-02	Microphone permission denied	On first install, select "Deny"/"Reject" on the voice recording permission.	The recording feature does not work; the application informs the user that permission is required.	Valid
TC-ERR-03	Silent voice input	Record but produce no sound (silent), then stop.	AssemblyAI returns no text; the application displays: "Suara tidak jelas" (the voice is unclear).	Valid

Table 4.5 shows that all three error-handling test cases passed. When microphone permission was denied, when the network connection was lost, and when silent input was provided, the application failed gracefully with an informative message rather than crashing. The requirement for robust, non-crashing error handling was therefore satisfied, which underpins the stability that distinguishes the cloud-based Scenario 2 from the on-device Scenario 1.

4.4.5 Refinements and Retesting (Iteration 2)

A debugging and source-code refinement process was conducted based on the three failures identified in the first iteration (TC-TR-05, TC-TR-06, and TC-GM-05). For TC-TR-05, image asset update logic was added using notifyDataSetChanged when the toggle button is

pressed, ensuring the visualization refreshes immediately when the standard is switched. For TC-TR-06, the clear function in the visualization ArrayList was refined to ensure the display is cleared completely before new input. For TC-GM-05, score storage was corrected using SharedPreferences to guarantee persistence across application sessions. The retesting results are shown in Table 4.6

Table 4.6 Iteration 2

Test ID	**Test Scenario**	**Test Steps**	**Expected Result**	**Status**
TC-TR-05	SIBI/BISINDO toggle feature	(1) Translate in SIBI mode. (2) Change the toggle to BISINDO. (3) Re-translate.	The visualization images change according to the selected standard.	Valid
TC-TR-06	Visualization reset	Press the "Delete"/"Reset" button after the translation is complete.	The text and visualization images disappear; the system is ready for new input.	Valid
TC-GM-05	Score storage (persistence)	Achieve a high score, close the application, and reopen the Quiz menu.	The high score remains displayed after closing and reopening.	Valid

Following Iteration 2, all 17 test cases across the three modules and error handling passed, resulting in a final Black Box Testing pass rate of 100%. This confirms that, after iterative refinement, all functional requirements of the application, covering the user interface, module navigation, state management, and error handling, were successfully satisfied.

4.5 Performance Testing (Processing Speed and Latency)

Performance evaluation in Scenario 2 measures the time efficiency of the cloud processing pipeline. Because this is an interactive application whose usability depends on promptness of feedback, system response time is a crucial quality factor. Latency was measured at millisecond precision using the Android timekeeping system monitored through Logcat. Three time parameters are evaluated: Audio Duration (t1), the time taken by the user to utter the word; API Response Time (t2), the time taken to upload the .wav file to the server, poll,

and obtain the transcribed text; and Visual Render Time (t3), the time taken by the device to match the transcribed word against the image dataset array and display the first frame of the sequential animation. Total System Latency is t2 + t3. Testing used standard 4G/LTE connectivity on the same low-to-mid-end device as Scenario 1, ensuring a fair architectural comparison. The testing results are shown in Table 4.7.

Table 4.7 Performance Testing

No.	Test Word	Characters	Audio (t1)	API Response (t2)	Render (t3)	Total Latency
1.	*Ibu*	3	1.80 s	5.16 s	0.01 s	5.17 s
2.	*Buku*	4	1.79 s	4.78 s	0.01 s	4.79 s
3.	*Makan*	5	1.61 s	2.90 s	0.01 s	2.91 s
4.	*Belajar*	7	1.73 s	1.85 s	0.01 s	1.86 s
5.	*Keluarga*	8	3.21 s	2.60 s	0.01 s	2.61 s
6.	*Semangat*	8	1.78 s	2.11 s	0.01 s	2.12 s
7.	*Terima Kasih*	11	2.24 s	2.27 s	0.01 s	2.28 s
8.	*Sama-sama*	8	2.16 s	5.04 s	0.01 s	5.05 s
	Average		2.04 s	3.33 s	0.01 s	3.35 s

Offloading processing from the device to the AssemblyAI server effectively resolved the stability issues observed in Scenario 1. The average API transcription time was 3.33 seconds, while ConstraintLayout-based visual rendering was negligible at 0.01 seconds, giving a total system latency of 3.35 seconds. Although this includes network overhead, it remains acceptable for the application's word-by-word interaction model, in which the user speaks a word and waits for the visual response. More importantly, the cloud approach eliminated the frame loss and memory crashes observed in Scenario 1.

4.6 End-to-end Integration Testing (Accuracy)

This test evaluates the application’s success in converting voice input into the correct fingerspelling visualization, using a sample of 50 common Indonesian inputs. Because transcription is performed entirely by the third-party AssemblyAI service, the test is framed as an end-to-end integration test of the complete pipeline (capture, upload, transcription, dataset matching, and sequential display) rather than as a measurement of the intrinsic recognition accuracy of the ASR engine. The failure condition is defined as the application incorrectly detecting the word (for example, recognizing "batu" as "satu") or the visualization failing to appear. The detailed results for all 50 sample inputs are presented in Table 4.8.

Table 4.8 Accuracy Testing

No.	Voice Input	Visualization Output (Sequential)	Note	Result
1.	A	Sequence of sign images "A"	Not recognized	Invalid
2.	B	Sequence of sign images "B"	Not recognized	Invalid
3.	C	Sequence of sign images "C"	Not recognized	Invalid
4.	D	Sequence of sign images "D"	Not recognized	Invalid
5.	E	Sequence of sign images "E"	Not recognized	Invalid
6.	*Halo*	Sequence of sign images "H-A-L-O"	Appropriate	Valid
7.	*Saya*	Sequence of sign images "S-A-Y-A"	Appropriate	Valid
8.	*Batu*	Sequence of sign images "B-A-T-U"	Appropriate	Valid
9.	*Satu*	Sequence of sign images "S-A-T-U"	Appropriate	Valid
10.	*Makan*	Sequence of sign images "M-A-K-A-N"	Appropriate	Valid
11.	*Minum*	Sequence of sign images "M-I-N-U-M"	Appropriate	Valid
12.	*Lapar*	Sequence of sign images "L-A-P-A-R"	Appropriate	Valid
13.	*Haus*	Sequence of sign images "H-A-U-S"	Appropriate	Valid
14.	*Pagi*	Sequence of sign images "P-A-G-I"	Appropriate	Valid
15.	*Malam*	Sequence of sign images "M-A-L-A-M"	Appropriate	Valid
16.	*Tangan*	Sequence of sign images "T-A-N-G-A-N"	Appropriate	Valid
17.	*Kaki*	Sequence of sign images	Appropriate	Valid

		"K-A-K-I"		
18.	*Mata*	Sequence of sign images "M-A-T-A"	Appropriate	Valid
19.	*Mulut*	Sequence of sign images "M-U-L-U-T"	Appropriate	Valid
20.	*Pipi*	Sequence of sign images "P-I-P-I"	Appropriate	Valid
21.	*Selamat*	Sequence of sign images "S-E-L-A-M-A-T"	Appropriate	Valid
22.	*Lakukan*	Sequence of sign images "L-A-K-U-K-A-N"	Appropriate	Valid
23.	*Semangat*	Sequence of sign images "S-E-M-A-N-G-A-T"	Appropriate	Valid
24.	*Berjuang*	Sequence of sign images "B-E-R-J-U-A-N-G"	Appropriate	Valid
25.	*Hebat*	Sequence of sign images "H-E-B-A-T"	Appropriate	Valid
26.	*Sayang*	Sequence of sign images "S-A-Y-A-N-G"	Appropriate	Valid
27.	*Keluarga*	Sequence of sign images "K-E-L-U-A-R-G-A"	Appropriate	Valid
28.	*Ayah*	Sequence of sign images "A-Y-A-H"	Appropriate	Valid
29.	*Ibu*	Sequence of sign images "I-B-U"	Appropriate	Valid
30.	*Kakak*	Sequence of sign images "K-A-K-A-K"	Appropriate	Valid
31.	*Adik*	Sequence of sign images "A-D-I-K"	Appropriate	Valid
32.	*Kakek*	Sequence of sign images "K-A-K-E-K"	Appropriate	Valid

33.	*Nenek*	Sequence of sign images "N-E-N-E-K"	Appropriate	Valid
34.	*Kepala*	Sequence of sign images "K-E-P-A-L-A"	Appropriate	Valid
35.	*Rambut*	Sequence of sign images "R-A-M-B-U-T"	Appropriate	Valid
36.	*Mata*	Sequence of sign images "M-A-T-A"	Appropriate	Valid
37.	*Hidung*	Sequence of sign images "H-I-D-U-N-G"	Appropriate	Valid
38.	*Telinga*	Sequence of sign images "T-E-L-I-N-G-A"	Appropriate	Valid
39.	*Tangan*	Sequence of sign images "T-A-N-G-A-N"	Appropriate	Valid
40.	*Kaki*	Sequence of sign images "K-A-K-I"	Appropriate	Valid
41.	*Senang*	Sequence of sign images "S-E-N-A-N-G"	Appropriate	Valid
42.	*Sedih*	Sequence of sign images "S-E-D-I-H"	Appropriate	Valid
43.	*Takut*	Sequence of sign images "T-A-K-U-T"	Appropriate	Valid
44.	*Sakit*	Sequence of sign images "S-A-K-I-T"	Appropriate	Valid
45.	*Sehat*	Sequence of sign images "S-E-H-A-T"	Appropriate	Valid
46.	*Baik*	Sequence of sign images "B-A-I-K"	Appropriate	Valid
47.	*Buruk*	Sequence of sign images "B-U-R-U-K"	Appropriate	Valid
48.	*Besar*	Sequence of sign images	Appropriate	Valid

		"B-E-S-A-R"		
49.	*Kecil*	Sequence of sign images "K-E-C-I-L"	Appropriate	Valid
50.	*Marah*	Sequence of sign images "M-A-R-A-H"	Appropriate	Valid

*Note: items 1–5 are isolated letters (A–E), which were not recognized; items 6–50 are multi-character daily words, all of which were transcribed and visualized correctly. The minimal pair "Batu" (item 8) and "Satu" (item 9) were both recognized correctly, confirming that the pipeline distinguishes acoustically similar word*s.

As the detailed results show, the 50 inputs fall into two clearly distinct categories whose behavior differs sharply. Table 4.9 summarizes the outcome by category.

Table 4.9 Accuracy Testing

Input Category	**Number of Inputs**	**Correct Visualization**	**Success Rate**
Multi-character words (items 6–50)	45	45	100%
Single letters A–E (items 1–5)	5	0	0%

All 45 multi-character words produced correct visualization (100%), while none of the 5 single-letter inputs were recognized (0%). The two categories are reported separately because combining them into a single figure would obscure their distinct behavior. The single-letter failure is attributed to the short acoustic duration of isolated letters, a known difficulty for cloud ASR with short utterances, and not specific to AssemblyAI. It must be emphasized that this test was conducted with a single speaker under controlled conditions and therefore has limited external validity; performance under varied speakers, accents, and background noise was not evaluated. For the purposes of this thesis, the practical implication is that the application reliably supports word-level fingerspelling practice (the core learning activity) across the tested daily vocabulary.

4.7 User Acceptance Testing

UAT was administered via Google Form following the design described in Section 3.5.3. A total of 19 respondents from the general hearing public operated all three modules of the application before completing the questionnaire, which used a five-point Likert scale (1 = Strongly Disagree to 5 = Strongly Agree) across four aspects and ten statements. Respondent demographics are summarized in Table 4.10; the rating distribution for each evaluation aspect is presented in Table 4.11 through Table 4.14; and the overall recapitulation is presented in Table 4.15. Of the 19 respondents, 10 (66.67%) were in the 18–24 age group and 5 (33.33%) were older than 34, indicating that the sample combined younger users with a smaller group of older adults. Gender and prior sign-language exposure were not collected in the questionnaire and are noted as such.

Table 4.10 UAT Respondent Demographics

Characteristic	Category	Count	Percentage
Age	18–24	10	52.63%
	25–34	0	0.00%
	> 34	5	47.37%
Prior sign language exposure	Yes	—	—
	No	—	—
Total Respondents	Yes	19	100%

4.7.1 Aspect A: Interface and Design (UI/UX)

Table 4.11 Aspect A: Interface and Design (UI/UX)

Code	Statement	STS (1)	TS (2)	N (3)	S (4)	SS (5)	Score (%)
A1	The application interface is attractive, neat, and easy to understand for new users.	0	0	0	11	8	88.42
A2	The transition animation between fingerspelling images runs smoothly and proportionally.	0	0	0	9	10	90.53
	Aspect A average score						89.47

4.7.2 Aspect B: Performance

Table 4.12 Aspect B: Performance

Code	Statement	STS (1)	TS (2)	N (3)	S (4)	SS (5)	Score (%)
B1	The voice-to-text translation feature responds and displays the images quickly.	0	0	1	8	10	89.47
B2	The application runs stably and lightly, without crashing during use.	0	0	0	7	12	92.63
	Aspect B average score						91.05

4.7.3 Aspect C: Usability

Table 4.13 Aspect C: Usability

Code	Statement	STS (1)	TS (2)	N (3)	S (4)	SS (5)	Score (%)
C1	The SIBI/BISINDO toggle button works well and greatly facilitates comparing the signs.	0	0	1	9	9	88.42
C2	The Sign Dictionary module is neatly organized, making it easy to find letter references from A to Z.	0	0	1	8	10	89.47
C3	The Evaluation Quiz module is easy to play and makes learning more enjoyable (interactive).	0	0	1	9	9	88.42
	Aspect C average score						88.77

4.7.4 Aspect D: Educational Value

Table 4.14 Aspect D: Educational Value

Code	Statement	STS (1)	TS (2)	N (3)	S (4)	SS (5)	Score (%)
D1	This application helps me understand the fundamental difference between SIBI and BISINDO fingerspelling.	0	0	1	7	11	90.53
D2	The voice recognition feature helps me practice spelling words into sign language more independently.	0	0	1	8	10	89.47

D3	Overall, this application is highly suitable and beneficial for use as a learning medium for the general public.	0	0	1	9	9	88.42
	Aspect D average score						89.47

4.7.5 Overall Recapitulation

Table 4.15 recapitulates the average score of each aspect and the overall acceptance score, with the corresponding interpretation category drawn from Table 3.1

Table 4.15 UAT Overall Recapitulation

No.	Evaluation Aspect	Number of Statements	Score (%)	Category
1	Interface and Design (UI/UX)	2	89.47	Sangat Layak
2	Performance	2	91.05	Sangat Layak
3	Usability	3	88.77	Sangat Layak
4	Educational Value	3	89.47	Sangat Layak
	Overall Acceptance Score	10	89.58	Sangat Layak

4.7.6 Open-Ended Feedback and Suggestions

In addition to the Likert items, 14 of the 19 respondents provided written feedback in the final open-ended question; the complete responses are listed verbatim in Table 4.16 Summary of Open-Ended Feedback from Respondents. The majority were general expressions of approval rather than specific critique (for example, "Goodjob", "*Sudah bagus*", and "*Sudah cukup*"), which is consistent with the high Likert scores. Four responses contained concrete, actionable suggestions, which cluster into three themes: refinement of the user interface and colour scheme (adding a slider and more varied colours to avoid a monotone two-colour appearance), improvement of processing speed, and the incorporation of additional AI-driven capabilities. These actionable themes are consistent with the quantitative results, in which the Usability aspect, though still rated highly, received the lowest score among the four aspects, and they directly inform the recommendations for future development in Section 6.3.

Table 4.16 Summary of Open-Ended Feedback from Respondents

No.	Respondent	Feedback (translated from the original Indonesian)	Category

1	N***** I********	*Perhaps the UI could add a slider to make it more attractive, and add more colors so it is not monotonous in just one or two colors; but overall the application is very interesting.*	UI / visual refinement
2	M******* F******* H****	*Already good, AWESOME, COOL*	General praise
3	F***** G********	*add AI into it :)*	Feature: add AI
4	M******* R****	The application UI could be developed further.	UI / visual refinement
5	A***	*the application is very good and the display is also easy to understand*	General praise
6	M******* F*****	*the UI appearance could be made a bit nicer*	UI / visual refinement
7	S*******	Good job	General praise
8	P*** P******** S****	*Already good. Hopefully it is useful.*	General praise
9	V****	*Please improve the speed further.*	Performance: speed
10	R**** H**********	*Already sufficient.*	General praise
11	F********** H******	*The application is already very interesting and very helpful, especially for beginners. The slide transition timing could be increased a little, especially for elderly users who fall under andragogy (education for the elderly) so it can be followed more optimally. Thank you, wishing continued success.*	UX: slide pacing
12	E**** P********	*I hope this application can be used and bring convenience to everyone, including the general public.*	General praise
13	R***	*Wishing continued success and ever more benefit for everyone.*	General praise
14	N**********	*Very useful for the future.*	General praise

CHAPTER V
RESULTS AND DISCUSSION

5.1 Comparison of the Two Scenarios

Table 5.1 presents a head-to-head comparison of Scenario 1 (Local/Offline AI) and Scenario 2 (Cloud API) across the testing parameters adopted in this research. To facilitate fair comparison, both scenarios were analyzed on the same device (Xiaomi Redmi Pad with MediaTek Helio G99 running Android 14) using the same debug build, so that any performance difference results from the computing model rather than from variations in hardware or software. For Scenario 2, a substantial amount of computation occurs on the AssemblyAI server side, and the recorded latency is therefore subject to network conditions at the time of testing.

Table 5.1 Comparative of Two Scenarios

No.	Testing Parameter	Scenario 1 (Local/Offline AI)	Scenario 2 (Cloud API)
1.	Architecture	On-device inference (Whisper C++, 4 CPU cores)	Cloud REST API (AssemblyAI), offloaded processing
2.	Avg. processing latency	~83 s/word (device); ~124 s/word (emulator); debug, unoptimized	~3.35 s/word (acceptable for interactive use)
3.	Accuracy (voice-to-text)	Degradation under sustained load; hallucinations detected	Words 45/45 (100%); single letters 0/5 (0%); single-speaker integration test
4.	Device stability	High resource load; frequent GC events; memory overload	Stable; no frame drops or crashes; light footprint
5.	Application features	Single feature (voice-to-text translation)	Three full modules: Translation, Dictionary, Quiz
6.	Black Box Testing	Not evaluated (abandoned at performance stage)	All test cases passed after iterative bug-fixing
7.	Internet dependency	None (fully offline)	Required for translation (handled via TC-ERR-01)
8.	Overall verdict	Not selected — impractical under tested configuration	Selected — meets functional, performance,

			and accessibility requirements

As Table 5.1 shows, the two scenarios differ markedly. Scenario 1, although functional, was impractical on the low-end target device in the tested configuration due to extreme latency and resource usage, though this reflects the debug configuration rather than an inherent limit of on-device ASR. Scenario 2 addresses these issues by offloading speech processing to the cloud, yielding a more responsive and lighter solution. Given the project's time and configuration constraints, Scenario 2 is the justified choice for the final application.

5.2 Discussion of Performance Results

The migration from on-device processing to cloud processing reduced average per-word latency from approximately 83 seconds to 3.35 seconds, a reduction of roughly 96%. The near-zero visual render time (0.01 seconds on average) confirms the effectiveness of the ConstraintLayout-based sequential rendering design and demonstrates that the entire performance bottleneck resides in the network round-trip (t2) rather than local rendering (t3). Consequently, future optimization effort should target t2, most plausibly through audio compression before upload.

A noteworthy pattern in Table 4.7 is that API response time does not correlate monotonically with word length: "Ibu" (3 characters) exhibits a higher API latency (5.16 s) than the considerably longer "Belajar" (7 characters, 1.85 s). This variability is most plausibly attributable to server-side load fluctuations at AssemblyAI during testing rather than to any property of the words themselves. It illustrates a structural characteristic of cloud-dependent architectures: a portion of system latency lies outside the developer's control. For a learning application with a word-by-word interaction model this variability is tolerable, but it would become a more serious concern for continuous, real-time translation use cases.

The application's dependence on an active internet connection for the Translation module is a direct consequence of the deliberate decision to delegate speech processing to the cloud, which is precisely what keeps the application lightweight and stable on low-end devices. Rather than being treated as a shortcoming of the application, this dependence is more accurately understood as an inherent and accepted trade-off of the cloud-based architecture: offloading computation to a remote server necessarily requires connectivity at the moment of translation. The trade-off is also bounded, since only the Translation module calls the cloud, whereas the

Dictionary and Quiz modules read local assets and therefore remain fully functional offline. For this reason, internet dependence is discussed here as a consequence of the architectural choice rather than listed among the study's limitations.

5.3 Discussion of Integration and Accuracy Results

The 100% correct visualization rate for multi-character words (45/45) validates the end-to-end pipeline (capture, upload, transcription, dataset matching, and sequential display) under the tested conditions. Because the test used a single speaker articulating clearly in a controlled environment, and because cloud ASR accuracy for Indonesian is generally lower than for high-resource languages such as English, this result should be read as a near-best-case scenario rather than a guarantee of field performance.

The complete failure on single-letter inputs (0/5) reflects a known limitation of ASR models optimized for word- and phrase-level recognition: an isolated letter provides too little acoustic information for reliable detection. This finding is consistent with Permana & Sutopo (2023), who likewise reported low recognition accuracy for individual letters (notably A, E, D, and J) in their vision-based SIBI recognition system, suggesting that single-letter recognition is challenging across modalities, not merely for audio. Future work may address this through a dedicated alphabet mode using phonetic letter names or standardized code words.

5.4 Discussion of UAT Results

The UAT evaluated the application across four aspects, Interface and Design (UI/UX), Performance, Usability, and Educational Value, using a five-point Likert scale, and the per-aspect and overall scores are recapitulated in Table 4.15. The overall acceptance score was 89.58%, which falls in the 81 to 100% band of the interpretation scale in Table 3.1 and is therefore categorized as Very Feasible (Sangat Layak). All four aspects also fell within this top band. The aspect that contributed most to the overall result was Performance, at 91.05%, followed by UI/UX and Educational Value at 89.47% each, while the aspect that still requires the most improvement was Usability, which scored the lowest at 88.77%. The strong Performance score is consistent with the stable, crash-free operation and acceptable word-by-word latency reported in Section 5.2, while the comparatively lower, though still high, Usability score aligns with the open-ended feedback, in which several respondents requested interface and colour-scheme refinements; the toggle and dictionary features were nonetheless well received, with no rating below 3 on any item.

Of particular relevance to the research objective is statement D3, "Overall, this application is highly suitable and beneficial for use as a learning medium for the general public", which obtained a score of 88.42% (Sangat Layak), with 9 of 19 respondents selecting Strongly Agree and the remaining 9 selecting Agree, and only 1 neutral, with no negative responses. Because this thesis frames the application explicitly as a learning medium, in contrast to the companion paper's engineering focus, this single item provides the most direct evidence addressing the fourth research question. The Educational Value aspect as a whole (statements D1–D3, averaging 89.47%) speaks to whether respondents perceived the application as helping them understand the SIBI/BISINDO distinction and practice fingerspelling independently, which are the core learning outcomes the application is designed to support.

The inclusion of UAT distinguishes this thesis from the companion paper publication, which deliberately exclude learning-effectiveness evaluation and identified it as future work. By evaluating user acceptance directly, this thesis addresses the fourth research question: whether the developed application is acceptable as a sign language learning medium. Precedents in the literature provide useful benchmarks for interpretation: Nurpiena et al. (2021) reported a usability score of 83.2% for a comparable sign language learning application, and Molina et al. (2024) reported an acceptance rate above 80%. Scores in this range are conventionally interpreted as indicating strong user acceptance for learning media of this type. It should nonetheless be acknowledged that UAT measures perceived acceptability rather than measured learning gain; a controlled learning-outcome study with pre- and post-test assessment remains a distinct and necessary follow-up. Consistent with the position taken in the companion paper (which frames the work as a functional and architectural feasibility study and explicitly does not evaluate learning effectiveness) the UAT in this thesis should be read as evidence of user-perceived acceptability of the application as a learning-support medium, not as proof of measured learning gain. Establishing actual fingerspelling learning outcomes would require a separate pre-test/post-test study with learners, teachers, and Deaf community involvement.

5.5 Discussion of Ethical Consideration

The decision to restrict the translation feature to fingerspelling proved sound from both technical and ethical standpoints. Technically, each fingerspelling letter corresponds to a unique, static hand configuration that can be represented faithfully by a 2D image, eliminating ambiguity in visualization. Ethically, the restriction prevents the application from

misrepresenting BISINDO's complex visual-spatial grammar (encompassing facial expression, movement intensity, and non-manual features) which cannot be reduced to static images without distortion.

This position is grounded in the literature on Deaf linguistic justice. Wahyudin (2020) documents how the marginalization of Deaf culture is linked to power asymmetries in which hearing perspectives dominate the representation of sign language; an application built by hearing developers that purported to translate full BISINDO would risk reproducing exactly that asymmetry. By contrast, fingerspelling is standardized, culturally neutral in its letter-to-handshape mapping, and explicitly positioned in this application as an entry point, a bridge toward communication, not a substitute for authentic engagement with the Deaf community and its natural language. The application is thus deliberately designed to invite further learning rather than to claim completeness.

It is important to be transparent about how this deliberate scope affects the application's educational value for users whose ultimate goal is practical, conversational sign-language communication. Fingerspelling covers only the manual spelling of the A to Z alphabet; it is not, in itself, sufficient for fluent everyday communication, which in BISINDO relies on a full lexicon of word-level signs together with visual-spatial grammar, facial expression, and other non-manual features that fingerspelling does not convey. Consequently, a learner who masters only the content of this application will be able to spell names and words letter by letter, but will not yet be able to hold a natural signed conversation. The educational value of the application should therefore be understood as that of a foundational entry point or scaffold: it lowers the barrier to first engagement, builds familiarity with the letter-to-handshape mapping of both standards, and prepares and motivates learners to pursue fuller sign-language instruction, ideally with the Deaf community and qualified instructors. Framed in this way, the fingerspelling-only scope is a limitation on communicative completeness, but not on the application's intended role as an accessible starting point.

It is also worth explaining why the Deaf community was not involved in the development and evaluation of this particular application. The primary reason is that the application is targeted at users who can hear and speak, since its core interaction depends on clear spoken input to drive the translation; members of the Deaf community, who typically do not communicate through spoken voice, are therefore not the intended direct users of this specific tool, and the User Acceptance Testing was accordingly conducted with hearing members of the public who represent that target group. A secondary consideration is that the scope was

restricted to fingerspelling, which, unlike the full visual-spatial grammar of BISINDO, is standardized: each letter corresponds to a fixed and well-documented handshape that is consistent across the numerous publicly available references against which the visual assets were cross-checked. This standardization gives reasonable confidence in the correctness of the A to Z assets specifically and lowers the risk of representational error, although, as discussed below, it does not fully substitute for expert or community validation.

The absence of Deaf community involvement also has implications for the validity and relevance of the results that must be stated plainly. Because no Deaf individuals, BISINDO users, or sign-language instructors participated in the design, asset creation, or evaluation, the linguistic and cultural accuracy of the SIBI and BISINDO visual assets has not been externally validated; the correctness of the signs therefore rests on publicly available references rather than on expert or community confirmation, which limits the ecological validity of the application as a faithful representation of the two standards. Similarly, the UAT results reflect the perceived acceptability of the application among hearing users, who are its intended audience, but they do not indicate whether the Deaf community would regard the assets or the overall approach as accurate and respectful. The findings are thus valid and relevant for the specific claim made, namely that hearing users find the application an acceptable entry-point learning medium, but they should not be generalized into a claim of validated linguistic accuracy or of endorsement by the Deaf community. This is precisely why involving the Deaf community is identified as an essential step for future validation rather than an optional enhancement.

CHAPTER VI
CONCLUSION

6.1 Conclusions

Based on the research conducted, the following conclusions are drawn. Each conclusion corresponds directly to a research problem formulated in Chapter I and to its associated research objective, presented in the same order, so that every research problem and objective is explicitly addressed:

a. An Android mobile application, named SiBisindo, was successfully designed and developed to facilitate the learning of SIBI and BISINDO fingerspelling for the general hearing public. Learning is facilitated through three complementary modules: Translation Practice, which converts spoken words into sequential fingerspelling visualization; Dictionary Reference, which allows learners to browse the full A to Z chart for both standards independently; and a Gamified Evaluation Quiz, which reinforces letter recognition through active recall. The scope of the translation feature was deliberately limited to A to Z fingerspelling, which is the academically and ethically appropriate boundary, as it reduces the risk of cultural appropriation by hearing developers and respects the status of BISINDO as an integral element of Deaf culture. The novelty of this work lies in its applied engineering value, namely the combination of a cloud ASR service, native Android development, and sequential fingerspelling visualization within a single application serving a dual-standard use case not previously addressed in the Indonesian context.

b. Regarding the suitable processing architecture and its efficient integration, the iterative development process showed that the initial on-device approach (Scenario 1) was impractical under the tested configuration, requiring about 83 seconds per word on the real device with memory overload and accuracy degradation. This led to its replacement with a cloud-based RESTful architecture (Scenario 2), integrated through the AssemblyAI REST API using an upload, poll, and retrieve flow, which achieved

an average total latency of 3.35 seconds per word, a reduction of approximately 96%, with stable operation on the low-end target device.

c. Regarding the reliability of the translation, end-to-end integration testing confirmed that the pipeline reliably converts spoken word input into correct fingerspelling visualizations. All 45 multi-character word inputs produced correct visualization (100%), while single-letter inputs from A to E were not recognized (0 of 5), a limitation attributed to the short acoustic duration of isolated letters. This functional reliability was further validated by Black Box Testing, which achieved a 100% pass rate across all 17 test cases after iterative refinement, covering the user interface, module navigation, state management, and error handling.

d. Regarding its acceptability as a learning medium, User Acceptance Testing with 19 respondents yielded an overall acceptance score of 89.58%, categorized as Very Feasible (Sangat Layak). All four aspects, namely UI/UX (89.47%), Performance (91.05%), Usability (88.77%), and Educational Value (89.47%), fell within the highest interpretation band, and the summary suitability item (D3) reached 88.42%. This indicates strong user-perceived acceptability of the application as a sign-language learning-support medium, while noting that perceived acceptability is distinct from measured learning gain, which was not assessed.

6.2 Research Limitations

This research acknowledges the following limitations:

a. The comparison between architectures was not conducted in a controlled benchmark setting. The on-device latency figures (~83 s/word on the device, ~124 s/word on the emulator) were measured on a debug build with an unquantized Whisper Tiny model; optimized code, threading, and quantization could reduce latency substantially. The choice of a cloud architecture was a pragmatic decision under time constraints, not a conclusion that efficient on-device inference is impossible.

b. The accuracy analysis addressed end-to-end integration rather than the intrinsic recognition accuracy of the ASR engine, which is operated by a third party. Testing used a single speaker and a 50-word sample, so external validity is limited; speech rate, regional accents, multiple speakers, and background noise were not evaluated. Cloud ASR accuracy for Indonesian is generally lower than for high-resource

languages, so the controlled single-speaker condition likely represents a near-best-case scenario.

c. Single-letter inputs (A–E) could not be recognized reliably.

d. UAT respondents were drawn from the general hearing public and did not include members of the Deaf community.

e. UAT measures perceived acceptability; formal learning outcomes (e.g., pre-/post-test gains) were not measured.

f. The SIBI and BISINDO visual assets were created by the author from publicly available web references and were not validated by sign-language experts or the Deaf community, so their linguistic and cultural accuracy is not guaranteed.

g. No Deaf individuals, BISINDO users, or sign-language instructors were involved in the design, asset creation, or evaluation of the application; the limitation of scope to fingerspelling is a precautionary design decision rather than the outcome of community participation.

h. Black Box Testing confirmed functional correctness, but formal regression testing across all modules was not separately conducted.

i. The ASR engine was not independently benchmarked; the 100% integration result reflects correct mapping of recognized text to visualization for a single speaker under controlled conditions, not broad speech-recognition accuracy.

j. Because audio is uploaded to a third-party cloud service (AssemblyAI), the application raises data-privacy considerations and is subject to API usage cost and the provider's continued service availability.

6.3 Recommendations for Future Development

Each recommendation below is derived directly from a specific limitation or empirical finding identified in this study, so that the recommendations logically extend its results:

a. To address the limitation that the architecture comparison was not conducted in a controlled benchmark setting, a controlled head-to-head benchmark should be conducted, comparing an optimized on-device configuration (release build, quantized model) against the cloud architecture, in order to establish a fair performance comparison.

b. To address the limitation that single-letter inputs from A to E could not be recognized reliably, a dedicated alphabet mode should be developed, using phonetic letter names or standardized code words.
c. To address the network-side latency that follows from the cloud-dependent architecture, and in response to the several UAT respondents who requested faster processing, audio compression prior to upload should be implemented to reduce latency (t2), particularly for lower-bandwidth conditions.
d. To address the limitation reflected in the Usability aspect scoring lowest, the user interface and visual design should be refined in response to UAT feedback, in which several respondents suggested adding interface controls (for example, a slider) and a more varied colour scheme to avoid a monotone two-colour appearance.
e. To address the feedback from an older respondent, who wrote from an andragogy (adult-learning) perspective, the pacing of the sequential fingerspelling animation should be made adjustable; this is particularly relevant given that nearly half of the UAT respondents were older than 34.
f. To address the feature requests raised by respondents in the open-ended feedback, additional AI-driven features, such as on-device or smarter recognition aids, should be explored, while retaining the lightweight cloud architecture that keeps the application accessible on low-end devices.
g. To address the limitation that the ASR engine was not independently benchmarked and that cloud ASR accuracy for Indonesian is variable, additional cloud ASR providers for Indonesian should be evaluated to identify the optimal accuracy-latency trade-off.
h. To address the limitation that no Deaf individuals or experts were involved and that the visual assets were not externally validated, the Deaf community, BISINDO users, and sign-language instructors should be involved in future asset validation, design, and evaluation.
i. To address the limitation that only perceived acceptability was measured, a controlled learning-outcome study using pre-test and post-test assessment should be conducted to establish actual fingerspelling learning gains.

REFERENCES

‘Isyatin Rodhiyah, B., & Wulandari, S. (2024). Mobile-Based Applications As a Tool for Basic Learning of Sign Language. *Journal of Information Technology and Computer Science (INTECOMS)*, *7*(1), 1–6. https://doi.org/https://doi.org/10.31539/intecoms.v7i1.8326

Al Khuzayem, L., Shafi, S., Aljahdali, S., Alkhamesie, R., & Alzamzami, O. (2024). Efhamni: A Deep Learning-Based Saudi Sign Language Recognition Application. *Sensors*, *24*(10). https://doi.org/10.3390/s24103112

Alam, I., Hameed, A., & Ziar, R. A. (2024). Exploring Sign Language Detection on Smartphones: A Systematic Review of Machine and Deep Learning Approaches. *Advances in Human-Computer Interaction*, *2024*. https://doi.org/10.1155/2024/1487500

Amangeldy, N., Ukenova, A., Bekmanova, G., Razakhova, B., Milosz, M., & Kudubayeva, S. (2023). Continuous Sign Language Recognition and Its Translation into Intonation-Colored Speech. *Sensors*, *23*(14). https://doi.org/10.3390/s23146383

Android Developers. (2023). *Kotlin and Android*. Developers.Android.Com. https://developer.android.com/kotlin

AssemblyAI. (2026). *AssemblyAI Documentation*. Assemblyai.Com. https://www.assemblyai.com/docs

Black 1964-, R. (2007). *Pragmatic software testing : becoming an effective and efficient test professional*. Indianapolis, Ind. : Wiley. https://searchworks.stanford.edu/view/13186544

Bolung, M., & Tampangela, H. R. K. (2017). Analisa Penggunaan Metodologi Pengembangan Perangkat Lunak. *Jurnal ELTIKOM*, *1*(1), 1–10. https://doi.org/10.31961/eltikom.v1i1.1

Fatmawati, R., Asmara, R., Prayogi, Y. R., & Hakkun, R. Y. (2022). Aplikasi Pembelajaran Sistem Isyarat Bahasa Indonesia (SIBI) Berbasis Voice Menggunakan OpenSIBI. *Technomedia Journal*, *7*(1), 22–39. https://doi.org/10.33050/tmj.v7i1.1690

Gumelar, G., Hafiar, H., & Subekti, P. (2018). Konstruksi Makna BISINDO sebagai Budaya Tuli bagi Anggota GERKATIN. *INFORMASI Kajian Ilmu Komunikasi*, *48*(1), 65–78. https://doi.org/https://doi.org/10.21831/informasi.v48i1.17727

Handayani, A. N., Hariyono, Nasih, A. M., Rochmawati, Hitipeuw, I., Rosyid, H. A., Ardiansah, J. T., Praja, R. I., Nurdiansyah, A., & Azizah, D. F. (2025). Comparative Analysis of Speech-to-Text APIs for Supporting Communication of the Deaf Community. *Indonesian Journal of Data and Science*, *6*(3), 492–500.

https://doi.org/https://doi.org/10.56705/ijodas.v6i3.327

Handayanto, I. S., & Nuryasin, I. (2024a). Pengujian Blackbox Decision Table pada Sistem Aplikasi Mobile Sharing Story App. *Smart Comp: Jurnalnya Orang Pintar Komputer*, *13*(2). https://doi.org/10.30591/smartcomp.v13i2.6572

Handayanto, I. S., & Nuryasin, I. (2024b). Pengujian Blackbox Decision Table pada Sistem Aplikasi Mobile Sharing Story App. *Smart Comp: Jurnalnya Orang Pintar Komputer*, *13*(2). https://doi.org/10.30591/smartcomp.v13i2.6572

Hikmatia A. E., N., & Ihsan Zul, M. (2021). Aplikasi Penerjemah Bahasa Isyarat Indonesia menjadi Suara berbasis Android menggunakan Tensorflow. *Jurnal Komputer Terapan*, *7*(1), 74–83. https://jurnal.pcr.ac.id/index.php/jkt/

Larman, C., & Basili, V. R. (2003). Iterative and incremental developments. a brief history. *Computer*, *36*(6), 47–56. https://doi.org/10.1109/MC.2003.1204375

Molina, J., Castillo, E., Noguera, F., Guerrero, G., & Jerez-Villota, E. (2024). *Sign Language Learning App: Empowering Deaf Inclusion BT - Management, Tourism and Smart Technologies* (Á. Rocha, C. Montenegro, E. T. Pereira, J. A. M. Victor, & W. Ibarra, Eds.; pp. 362–374). Springer Nature Switzerland. https://doi.org/https://doi.org/10.1007/978-3-031-74828-8_32

Murni, R. K., Padlurrahman, & Murcahyanto, H. (2024). Peran Vital Bahasa Isyarat Indonesia dalam Membangun Komunikasi dan Integrasi Sosial Anak Tuli. *KIBASP*, *8*(1). https://doi.org/https://doi.org/10.31539/kibasp.v8i1.10103

Mursalat, Siregar, E., & Tarjiah, I. (2023). Pengembangan Video Pembelajaran Desain Grafis untuk Buku Digital Interaktif bagi Siswa Tunarungu. *Jurnal Paedagogy*, *10*(2), 589–597. https://doi.org/https://doi.org/10.33394/jp.v10i2.7073

Nassif, A. B., Shahin, I., Attili, I., Azzeh, M., & Shaalan, K. (2019). Speech Recognition Using Deep Neural Networks: A Systematic Review. *IEEE Access*, *7*, 19143–19165. https://doi.org/10.1109/ACCESS.2019.2896880

Nidhra, S., & Jagruthi, D. (2012). Black Box and White Box Testing Techniques - A Literature Review. *International Journal of Embedded Systems and Applications*, *2*(2), 29–50. https://doi.org/10.5121/ijesa.2012.2204

Nurpiena, S. A., Wihidayat, E. S., & Budianto, A. (2021). Developing Indonesia Sign Language (BISINDO) Application with Android Based for Learning Sign Language. *Journal of Informatics and Vocational Education*, *4*(1). https://doi.org/10.20961/joive.v4i1.48629

Onggo, L. T. (2025). Menakar Minat Masyarakat dalam Belajar Bahasa Isyarat: Antara Kesadaran Sosial dan Tren Komunikasi Inklusif. *Jurnal Ilmu Komunikasi Dan Media Sosial (JKOMDIS)*, *5*(3), 575–581. https://doi.org/10.47233/jkomdis.v5i3.3064

Paksi, A. B., Hafidhoh, N., & Bimonugroho, S. K. (2023). Perbandingan Model Pengembangan Perangkat Lunak Untuk Proyek Tugas Akhir Program Vokasi. *Jurnal Masyarakat Informatika*, *14*(1), 70–79. https://doi.org/10.14710/jmasif.14.1.52752

Permana, D., & Sutopo, J. (2023). APLIKASI PENGENALAN ABJAD SISTEM ISYARAT BAHASA INDONESIA (SIBI) DENGAN ALGORITMA YOLOv5. *Jurnal Simantec*, *11*(2), 231–240. https://doi.org/10.21107/simantec.v11i2.19783

Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., & Sutskever, I. (2023). Robust Speech Recognition via Large-Scale Weak Supervision. *Proceedings of Machine Learning Research*, *202*, 28492–28518.

Saman, P., & Indah Ratnasari, C. (2022). Pengujian Black Box Pada Aplikasi Pembelajaran Bahasa Mandarin Berbasis Android. *Jurnal Ilmiah Intech : Information Technology Journal of UMUS*, *4*(1), 10–22. https://doi.org/https://doi.org/10.46772/intech.v4i01.637

Stat Counter. (2026). *Mobile Operating System Market Share Worldwide*. Stat Counter, Global Stats. https://gs.statcounter.com/os-market-share/mobile/worldwide

Wahyudin, Y. M. (2020). Kekuasaan dalam Relasi Bahasa: Refleksi Pengalaman Penutur Bahasa Isyarat di Yogyakarta. *Inklusi*, *7*(1), 47–70. https://doi.org/10.14421/ijds.070103

WHO. (2026). *Deafness and hearing loss*. Who.Int. https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss

Wulandari, I., Damastuti, F. A., Fananda, I. Y., Pramulen, A. S., & Hasim, N. J. A. (2026). Penerapan Teknologi Komunikasi Terintegrasi bagi Gangguan Pendengaran di Gerkatin Malang untuk Inklusi Disabilitas Indonesia. *Jurnal AbdiMas Nusa Mandiri*, *8*(1), 181–187. https://doi.org/10.33480/abdimas.v8i1.7429

ZainEldin, H., Gamel, S. A., Talaat, F. M., Aljohani, M., Baghdadi, N. A., Malki, A., Badawy, M., & Elhosseini, M. A. (2024). Silent no more: a comprehensive review of artificial intelligence, deep learning, and machine learning in facilitating deaf and mute communication. In *Artificial Intelligence Review* (Vol. 57, Number 7). Springer Netherlands. https://doi.org/10.1007/s10462-024-10816-0

APPENDIX

APPLICATION SOURCE CODE

This appendix presents selected excerpts of the application source code that directly support the methods and testing results discussed in Chapter III and Chapter IV. For brevity and readability, only the parts relevant to specific claims and test cases are included; user-interface layout files, list adapters, and standard Android boilerplate are omitted. The AssemblyAI API key has been replaced with the placeholder YOUR_API_KEY, and some long statements have been wrapped across several lines. The application declares two permissions, quoted below, which underlie the connectivity and microphone error-handling test cases (TC-ERR-01 and TC-ERR-02):

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />;
}
```

Figure 0.1 On-Device Whisper Integration via JNI (Scenario 1).

Figure 0.1 shows the on-device integration used in Scenario 1 (Section 4.1). The application loads a native Whisper C++ library through the Java Native Interface (JNI) and calls it to transcribe audio entirely on the device. The Whisper Tiny model file is copied from the application assets and loaded at startup, and the transcription time is measured with System.currentTimeMillis(), producing the latency figures reported in Table 4.1.

```
class MainActivity : AppCompatActivity() {


    // Native library (Whisper C++) loaded through JNI
    companion object {
        init { System.loadLibrary("sibiwhisper") }
    }


    private external fun loadModel(modelPath: String): Long
    private external fun transcribeAudio(audioPath: String): String
    private external fun freeModel()


    // Copy the Whisper Tiny model from assets, then load it on-device
    private fun initModel() {
        CoroutineScope(Dispatchers.IO).launch {
            val modelFile = File(filesDir, "ggml-tiny.bin")
            if (!modelFile.exists()) {
                assets.open("ggml-tiny.bin").use { input ->
```

```
                            FileOutputStream(modelFile).use { output ->
                                input.copyTo(output)
                            }
                        }
                    }
                    loadModel(modelFile.absolutePath)
                }
            }

            // On-device transcription with latency measurement
            private fun processTranscription() {
                CoroutineScope(Dispatchers.IO).launch {
                    val startTime = System.currentTimeMillis()
                    val resultText = transcribeAudio(recordedFile!!.absolutePath)
                    val duration = System.currentTimeMillis() - startTime
                    withContext(Dispatchers.Main) {
                        binding.txtResult.text = resultText
                        binding.txtStatus.text = "Selesai (${duration}ms)"
                    }
                }
            }

            override fun onDestroy() {
                super.onDestroy()
                freeModel()
            }
}
```

Figure 0.2 Audio Capture Configuration (16 kHz, Mono, PCM 16-bit)

Figure 0.2 shows the audio capture configuration shared by both scenarios. Audio is recorded as 16 kHz, mono, 16-bit PCM using the low-level AudioRecord API, which is the format required for upload to AssemblyAI in Scenario 2 (Section 4.3). The raw PCM stream is later wrapped in a standard 44-byte WAV header before being processed.

```
        class WavAudioRecorder {
            private val RECORDER_SAMPLE_RATE = 16000
            private val RECORDER_CHANNELS = AudioFormat.CHANNEL_IN_MONO
            private val RECORDER_AUDIO_ENCODING = AudioFormat.ENCODING_PCM_16BIT
            private var audioRecord: AudioRecord? = null
            private var isRecording = false
            private var bufferSize = 0
```

```
        @SuppressLint("MissingPermission")
        fun startRecording(outputFile: File): Boolean {
            bufferSize = AudioRecord.getMinBufferSize(
                RECORDER_SAMPLE_RATE,                    RECORDER_CHANNELS,
RECORDER_AUDIO_ENCODING)
            if (bufferSize == AudioRecord.ERROR ||
                bufferSize == AudioRecord.ERROR_BAD_VALUE) return false

            audioRecord = AudioRecord(
                MediaRecorder.AudioSource.MIC, RECORDER_SAMPLE_RATE,
                RECORDER_CHANNELS, RECORDER_AUDIO_ENCODING, bufferSize)
            if  (audioRecord?.state  !=  AudioRecord.STATE_INITIALIZED)  return
false

            audioRecord?.startRecording()
            isRecording = true
            Thread { writeAudioDataToFile() }.start()
            return true
        }

        private fun writeAudioDataToFile() {
            val data = ByteArray(bufferSize)
            val os = FileOutputStream(tempRawFile)
            while (isRecording) {
                val read = audioRecord?.read(data, 0, bufferSize) ?: 0
                if (read > 0) os.write(data, 0, read)
            }
            os.close()
        }
        // The PCM stream is wrapped with a standard 44-byte WAV header
        // (writeWavHeader) before upload; that routine is omitted here.
}
```

Figure 0.3 Cloud Transcription Pipeline via AssemblyAI REST API (Scenario 2)

Figure 0.3 shows the cloud transcription pipeline used in Scenario 2 (Section 4.3), implemented as a three-step AssemblyAI REST workflow: upload the audio file, request transcription with the Indonesian language code, and poll until the transcript is completed. The elapsed cloud time is logged as t2, the dominant component of the total latency reported in Table 4.7. The API key is censored.

```
    // API key is censored; supply your own AssemblyAI key here.
    private val ASSEMBLY_API_KEY = "YOUR_API_KEY"

    private val client = OkHttpClient.Builder()
        .connectTimeout(60, TimeUnit.SECONDS)
        .readTimeout(60, TimeUnit.SECONDS)
        .build()

    // Step 1: upload the .wav file, receive a temporary upload URL
    private fun uploadFileToAssembly(file: File): String {
        val body = file.asRequestBody(
            "application/octet-stream".toMediaTypeOrNull())
        val request = Request.Builder()
            .url("https://api.assemblyai.com/v2/upload")
            .addHeader("Authorization", ASSEMBLY_API_KEY)
            .post(body).build()
        val response = client.newCall(request).execute()
        return JSONObject(response.body?.string() ?: "")
            .getString("upload_url")
    }

    // Step 2: request transcription (Indonesian language code)
    private fun requestTranscription(audioUrl: String): String {
        val json = JSONObject().apply {
            put("audio_url", audioUrl)
            put("language_code", "id")
        }
        val body = RequestBody.create(
            "application/json".toMediaTypeOrNull(), json.toString())
        val request = Request.Builder()
            .url("https://api.assemblyai.com/v2/transcript")
            .addHeader("Authorization", ASSEMBLY_API_KEY)
            .post(body).build()
        val response = client.newCall(request).execute()
        return JSONObject(response.body?.string() ?: "").getString("id")
    }

    // Step 3: poll until the transcript status is completed
    private fun checkStatus(transcriptId: String): Pair<String, String> {
        val request = Request.Builder()
            .url("https://api.assemblyai.com/v2/transcript/$transcriptId")
            .addHeader("Authorization", ASSEMBLY_API_KEY)
```

```
            .get().build()
        val response = client.newCall(request).execute()
        val json = JSONObject(response.body?.string() ?: "")
        return Pair(json.getString("status"),
            json.optString("text", ""))
    }

    // Orchestration with cloud latency logging (t2)
    private fun processWithAssemblyAI(audioFile: File) {
        CoroutineScope(Dispatchers.IO).launch {
            val startCloud = System.currentTimeMillis()
            val audioUrl = uploadFileToAssembly(audioFile)
            val transcriptId = requestTranscription(audioUrl)
            var resultText = ""
            while (true) {
                val (status, text) = checkStatus(transcriptId)
                if (status == "completed") { resultText = text; break }
                else if (status == "error") { resultText = "Error API"; break }
                delay(1000)
            }
            val durationCloud = System.currentTimeMillis() - startCloud
            Log.d("PERFORMA_TEST", "Cloud t2: $durationCloud ms")
            // resultText is then mapped to fingerspelling images.
        }
    }
```

1Figure 0.4 Voice-to-Fingerspelling Mapping

1Figure 0.4 shows how recognized text is converted into a sequence of fingerspelling images. Each letter is mapped to a drawable resource using the naming convention mode_letter (for example, sibi_a) through resources.getIdentifier, and the matching time is logged as t3 (Section 4.6). Characters without a corresponding image are skipped.

```
    // Convert recognized text into a sequence of sign-image IDs
    private fun prosesTeksKeGambar(kalimat: String) {
        val startMapping = System.currentTimeMillis()

        // Keep letters only (A to Z); ignore other characters
        val teksBersih = kalimat.lowercase().filter { it.isLetter() }
        val daftarIdGambar = mutableListOf<Int>()

        for (huruf in teksBersih) {
            // File naming convention: mode_letter, e.g. sibi_a
```

```
            val namaFile = "${modeBahasa}_$huruf"
            val resourceId = resources.getIdentifier(
                namaFile, "drawable", packageName)
            if (resourceId != 0) daftarIdGambar.add(resourceId)
        }

        // t3: time to match text to the image dataset array
        val durationMapping = System.currentTimeMillis() - startMapping
        Log.d("PERFORMA_TEST", "Mapping t3: $durationMapping ms")

        // daftarIdGambar is then rendered sequentially (animation).
    }
```

Figure 0.5 SIBI/BISINDO Toggle Logic (Test Case TC-TR-05)

Figure 0.5 shows the SIBI/BISINDO toggle logic. When the standard is switched, the most recently recognized word is re-mapped and re-rendered immediately using the newly selected standard. This is the behavior corrected in the second iteration for test case TC-TR-05 (Section 4.4.5).

```
    // Switching the standard re-renders the current word immediately
    switchBahasa.setOnCheckedChangeListener { _, isChecked ->
        modeBahasa = if (isChecked) "bisindo" else "sibi"
        if (lastRecognizedText.isNotEmpty()) {
            // Re-map the last recognized text using the new standard
            prosesTeksKeGambar(lastRecognizedText)
        } else {
            signAdapter.updateData(emptyList())
        }
        updateHomeHighScore()
    }
```

Figure 0.6 Quiz Generation and Score Persistence (Test Case TC-GM-05)

Figure 0.6 shows the core of the evaluation quiz (Section 4.4.3). The function generateQuestion selects one correct answer and three random distractors and displays the corresponding sign image, while saveHighScore persists the best score across sessions using SharedPreferences, which is the fix applied for test case TC-GM-05.

```
    private val alphabet = ('a'..'z').toList()


    // Generate one question: one correct answer plus three distractors
    private fun generateQuestion() {
        correctAnswer = alphabet.random()
        val wrongAnswers = alphabet
            .filter { it != correctAnswer }.shuffled().take(3)
        val options = (wrongAnswers + correctAnswer).shuffled()


        val namaGambar = "${modeBahasa}_$correctAnswer"
        val resId = resources.getIdentifier(
            namaGambar, "drawable", packageName)
        if (resId != 0) Glide.with(this).load(resId).into(imgSoal)


        for (i in 0 until 4)
            buttons[i].text = options[i].toString().uppercase()
    }


    // Persist the high score across sessions (SharedPreferences)
    private fun saveHighScore() {
        val sharedPref = getSharedPreferences("QuizData", MODE_PRIVATE)
        val key = "HIGHSCORE_$modeBahasa"
        val oldHighScore = sharedPref.getInt(key, 0)
        if (currentScore > oldHighScore) {
            sharedPref.edit().putInt(key, currentScore).apply()
        }
    }
```

Figure 0.7 Sign Image Item Layout (85dp, ConstraintLayout)

Figure 0.7 shows the layout of a single sign image. Each image is rendered inside a fixed 85dp CardView using ConstraintLayout with fitCenter scaling, which supports the low visual render time (t3, on average 0.01 seconds) discussed in Section 2.3.1 and Section 4.5. Namespace declarations and layout constraints are omitted for brevity.

```
    <androidx.cardview.widget.CardView
        android:layout_width="85dp"
        android:layout_height="85dp"
        app:cardCornerRadius="16dp"
        app:cardElevation="3dp">
```

```
            <ImageView
                android:id="@+id/imgSign"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:scaleType="fitCenter"
                android:padding="8dp" />


</androidx.cardview.widget.CardView>
```