International Research Journal of Education and Technology
Peer Reviewed Journal
ISSN 2581-7795

OFFLINE AI TUTOR USING LLM

Nareshkumar M R and Lennox Cecil JM

II MSC Computer Science , Sri Krishna Arts and Science College, Coimbatore

ABSTRACT

The Offline AI Tutor with Local LLM is a next-generation language learning application designed to deliver an autonomous, immersive, and intelligent experience without requiring internet connectivity. Unlike cloud-based apps that depend on external servers and pose privacy risks, it leverages locally hosted Large Language Models (LLMs) such as GEMMA or DEEPSEEK. This architecture ensures complete data privacy, zero-latency responses, and uninterrupted learning, making it ideal for remote learners, frequent travelers, and privacy-conscious users.

The tutor enables natural, real-time conversations with a human-like AI while providing grammar correction, sentence restructuring, and vocabulary enhancement tailored to the learner's proficiency level. With Automatic Speech Recognition (ASR) for voice input and Text-to-Speech (TTS) for audio feedback, it supports hands-free interactions and pronunciation practice, fostering well-rounded development in speaking, listening, reading, and writing. Its multilingual design and adaptive learning path offer personalized challenges and feedback to improve outcomes effectively.

Optimized for consumer-grade hardware, the system runs efficiently using quantized or distilled models, balancing performance with minimal resource use. All interaction history remains securely stored on the device, ensuring full control over personal data. Whether for classrooms, travel, or independent study, the Offline AI Tutor provides a robust, private, and reliable language learning solution—delivering consistent access to high-quality education anytime, anywhere.

International Research Journal of Education and Technology
Peer Reviewed Journal
ISSN 2581-7795

CHAPTER-1

INTRODUCTION

In addressing one of the common challenges faced by students—reliable access to intelligent tutoring without dependence on the internet—we have conceptualized the idea of an Offline AI Tutor mobile application.This application aims to empower students with an AIpowered language learning assistant that operates entirely offline, thereby ensuring uninterrupted access, enhanced privacy, and zero latency in interaction. Unlike cloud-based tools that require constant connectivity and raise privacy concerns, our app integrates a locally hosted Large Language Model (LLM) such as GEMMA, DEEP SEEK, optimized to run efficiently on mobile devices. This allows users to engage in real-time AI conversations, receive grammar corrections, and build vocabulary without transmitting any data online. With features like speech recognition and text-to-speech capabilities, the Offline AI Tutor supports natural, voice-based interaction, enhancing the overall user experience. The goal of this project is to offer students, travelers, and language enthusiasts a secure, responsive, and intelligent platform for language learning— anytime, anywhere—without compromising data security or requiring an active internet connection .

1.1 PROJECT OVERVIEW

The Offline AI Tutor is a mobile application designed to provide students, language enthusiasts with an intelligent, interactive, and secure platform for language learning without relying on internet connectivity. Unlike conventional AI-powered learning tools that require cloud-based servers, this application integrates a locally hosted Large Language Model (LLM) such as GEMMA or DeepSeek, optimized for on-device processing. The project aims to address key challenges of existing systems, including privacy concerns, internet dependency, latency issues, and limited accessibility in offline environments. By leveraging speech recognition (ASR), text-to-speech (TTS), and natural language processing (NLP), the application enables real-time AI-driven conversations, grammar correction, and vocabulary assistance, all while maintaining full offline functionality

1.2 PROJECT OBJECTIVE

The objective of the Offline AI Tutor project is to develop a mobile application that enables intelligent, AI-powered language learning entirely offline, ensuring privacy, low latency, and accessibility without dependence on internet connectivity. The system aims to provide real- time grammar correction, vocabulary assistance, and interactive conversations through locally hosted large language models, while supporting both text and voice interaction using speech recognition and text-to-speech technologies. By optimizing performance for mobile devices, the application ensures smooth operation even on low-end hardware, offering a secure, responsive, and user-friendly platform for students, travelers, and language enthusiasts. Furthermore, the scalable design supports future enhancements such as multilingual capabilities and personalized learning paths, positioning the application as a comprehensive and reliable offline language learning solution. The purpose of the Offline AI Tutor project is to overcome the major limitations of existing cloud-based language learning applications, which require constant internet connectivity, raise privacy concerns, and suffer from latency issues. Current systems transmit user data to external servers, making them vulnerable to breaches while also excluding users in low-connectivity or offline environments. This project addresses those challenges by providing a fully offline, AI- powered language learning assistant that processes all interactions locally on the device using Large Language Models (LLMs) such as GEMMA and DeepSeek. By removing dependence on cloud infrastructure, the system ensures uninterrupted access, complete privacy, and faster response times, empowering students, travelers, and language enthusiasts to continue learning anytime, anywhere.

1.3 PROJECT OVERVIEW

The Offline AI Tutor application, developed using Jetpack Compose in the Android environment, is designed to offer a seamless and intelligent language learning experience without the need for an internet connection. Targeted at students, travelers, and language enthusiasts, this app empowers users to engage in real-time AI-driven conversations, receive grammar corrections, and improve their vocabulary—entirely offline.

Leveraging a locally hosted Large Language Model (LLM) such as GEMMA, DEEP SEEK, the app ensures fast, secure, and private interactions on mobile devices. It utilizes speech recognition (ASR) and text-to-speech (TTS) technologies to support both voice and text-based communication, delivering a natural and interactive user experience. The AI processing is optimized for mobile devices, making the app responsive even on hardware with limited resources.

With a focus on privacy, accessibility, and performance, the Offline AI Tutor removes the dependency on cloud-based infrastructure, allowing users to access its features anytime, anywhere. Its scalable architecture also supports future enhancements such as multilingual capabilities and further personalization, positioning it as a comprehensive offline language learning solution.

1.6 MODULE DESCRIPTION

The Offline AI Tutor application is structured into multiple functional modules, each responsible for handling specific tasks. These modules work together to ensure seamless, secure, and efficient offline language learning.

1. User Interface Module (UI/UX)

This module, built using Jetpack Compose, provides an intuitive and interactive interface for learners. It allows users to navigate easily between text and voice modes, view grammar suggestions, and access stored conversations. The interface is designed to be modern, responsive, and user-friendly, supporting features such as dark/light themes and customizable profiles.

2. AI Processing Module (LLM Integration)

This is the core of the system, integrating locally hosted Large Language Models (LLMs) such as GEMMA or DEEPSEEK. It processes user queries, generates responses, corrects grammar, and suggests vocabulary. The module operates entirely on-device, ensuring privacy and delivering real-time responses with minimal latency.

3. Speech Processing Module (ASR & TTS)

This module handles speech-based interaction by converting spoken input into text and generating natural-sounding speech for AI responses. It uses Automatic Speech Recognition (ASR) to capture user voice input and Text-to-Speech (TTS) tools like Piper or VITS to provide human-like spoken output. Multi-language and accent support make this module highly adaptable.

4. Security & Authentication Module

This module ensures user data is protected and interactions remain private. It uses encryption mechanisms such as Encrypted Shared Preferences for local data protection. For users who prefer synchronization across devices, optional authentication with Firebase Auth (Email/Password, OAuth 2.0) is also supported, along with secure session management.

5. Storage & Database Module

This module manages both offline and optional cloud storage of conversations, preferences, and user data. Locally, it uses Room Database or SQLite to ensure that essential information is always available without internet access. For users who want cross-device synchronization, Firebase Firestore provides cloud backup and syncing options, creating a hybrid storage approach.

6. **Performance Optimization & Compatibility Module**

This module ensures smooth operation of the application across a wide range of Android devices, including low-end hardware. It optimizes CPU and GPU usage for local LLM inference, reduces battery and memory consumption, and maintains responsiveness during prolonged use.

The module guarantees compatibility with Android 8.0 (Oreo) and above, adapting seamlessly to various screen sizes and device specifications.

International Research Journal of Education and Technology
Peer Reviewed Journal
ISSN 2581-7795

CHAPTER-2

SYSTEM SPECIFICATION

2.1 HARDWARE SPECIFICATION

PROCESSOR	: Quad Core (Intel i5 / ARM equivalent)
RAM	: 8 GB or higher
STORAGE	: 256 GB HDD/SSD (minimum)
GPU	: Integrated GPU / Mobile GPU support

2.2 SOFTWARE SPECIFICATION

OPERATING SYSTEM	: Android 8.0 (Oreo) or higher DEVELOPMENT
TOOL	: Android Studio
PROGRAMMING LANGUAGE	: Kotlin / Java
FRAMEWORK	: Jetpack Compose
DATABASE	: SQLite / Room Database
AI MODELS	: GEMMA, DeepSeek (locally hosted LLMs)

2.3 SOFTWARE DESCRIPTION

The Offline AI Tutor application leverages a powerful ecosystem of Android development tools, AI models, and speech processing frameworks to deliver an intelligent and interactive offline learning experience. Each software component plays a crucial role in ensuring privacy, low latency, and scalability, while supporting both text and speech-based interactions. Below is a description of the key software components used in the project:

Android Studio

Android Studio is the official integrated development environment (IDE) for Android application development. It provides a robust platform for coding, debugging, and designing the application. In this project, Android Studio is used to develop the Offline AI Tutor, offering tools for layout design, performance profiling, and seamless integration of backend logic.

Kotlin / Java

Kotlin is the primary programming language used for developing the application due to its concise syntax, enhanced safety features, and strong interoperability with Java. Java is optionally supported to maintain compatibility with legacy Android components. Together, these languages handle the app's logic, UI controls, and data management.

Jetpack Compose

Jetpack Compose is a modern declarative UI framework for Android. It simplifies building responsive, interactive, and adaptive user interfaces. In this project, Jetpack Compose is used to design the chat interface, role-play selection screens, and other user interactions, ensuring a smooth and intuitive learning experience.

Firebase

Firebase Authentication is integrated to provide secure login using email/password or OAuth 2.0, while Firebase Firestore is optionally used for cloud synchronization of user preferences and chat

history. In this project, Firebase enhances security and enables multi-device accessibility when internet access is available, without compromising offline functionality.

GEMMA / DeepSeek (Locally Hosted LLMs)

GEMMA and DeepSeek are large language models optimized for on-device inference. These models are integrated into the project using inference frameworks such as MediaPipe or Ollama, enabling real-time grammar correction, vocabulary assistance, and interactive conversations without requiring server-based processing**.**

Speech Recognition and Text-to-Speech (ASR/TTS)

The project uses the Google Speech Recognition API for converting voice input into text, and TTS tools such as Piper or VITS to generate natural human-like speech for AI responses. These components enable immersive voice-based interaction, enhancing accessibility and user engagement.

Together, these software tools form a robust and modular framework that enables end-to-end implementation of the Offline AI Tutor system. The combination of Android development tools, local databases, AI models, and speech technologies ensures that the application delivers secure, real-time, and offline-first language learning experiences for users

International Research Journal of Education and Technology
Peer Reviewed Journal
ISSN 2581-7795

CHAPTER-3

SYSTEM STUDY

3.1 EXISTING SYSTEM WITH LIMITATIONS

Currently, most AI language learning applications require a constant internet connection for processing user input. These systems rely on cloud-based servers, which raises privacy concerns, causes latency issues, and limits accessibility in offline or low-connectivity environments. Additionally, users have minimal control over customization and cannot access full AI functionality without being online.

3.2 LIMITATIONS

Current AI language-learning applications that require a constant internet connection suffer from several critical drawbacks: they pose significant privacy risks by transmitting sensitive user interactions and learning data to cloud servers, which may be vulnerable to breaches or misuse; they introduce latency and inconsistent real-time feedback, particularly on slower networks, degrading the learning experience; they exclude users in low-connectivity or offline environments—such as remote regions, airplane travel, or during service outages—and impose high data and battery consumption as devices continuously upload and download information; furthermore, they limit user autonomy and personalization by centralizing AI behavior and feature control on remote servers, lock advanced functionality behind online-only gateways and subscription models, and prevent edge-device scalability by offloading all processing to the cloud.

3.3 PROPOSED SYSTEM WITH ADVANTAGES

The proposed system enables fully offline AI interactions using a locally hosted language model. It allows users to practice language skills through text or speech without internet, ensuring privacy, low latency, and easy access anytime, anywhere.

ADVANTAGES

The proposed system offers several significant advantages by enabling fully offline AI interactions through a locally hosted language model. First and foremost, it ensures complete user privacy, as no data is transmitted to external servers, eliminating the risk of data breaches or unauthorized access. The low latency response time provides a smoother and more immediate user experience, ideal for real-time language practice. Additionally, the system offers uninterrupted accessibility, allowing users to practice anytime and anywhere, regardless of internet availability—perfect for remote areas, travel, or low-connectivity environments. The offline model also supports reduced data and battery usage, making it more efficient and costeffective. Furthermore, users gain greater control and customization, as the system can be tailored to individual learning preferences and local device capabilities without relying on centralized updates or subscriptions.

CHAPTER-4

SYSTEM DESIGN

This chapter discusses system design. System design is the process of defining elements of a system such as modules, architecture, components, their interfaces, and data, based on specified requirements. It is a critical phase in software development where key decisions are made that directly impact the success, reliability, and maintainability of the system. The design phase ensures that all requirements are accurately transformed into a complete and workable system.

4.1 UML DESIGN

The Unified Modeling Language (UML) is a standardized graphical language used to specify, visualize, construct, and document the artifacts of a software system. It helps in understanding and designing both the conceptual and physical aspects of a system. UML provides a set of semantics and rules to model the structure and behavior of systems effectively.

VISUALIZATION

UML diagrams help visualize the structure and interaction between components of the Offline AI Tutor application, making it easier to understand how the system will function once implemented.

SPECIFYING

Specifying means building models that are precise, unambiguous and complete UML addresses the specification of all the important analysis design, implementation decisions that must be made in developing and deploying a software system.

CONSTRUCTING

The UML models provide a foundation for developing the actual application using technologies such as Kotlin, Jetpack Compose, and LLM integration through Ollama. Forward and reverse engineering processes are also supported.

DOCUMENTING

In addition to source code, UML diagrams help document system requirements, design architecture, component interactions, and other key development artifacts for future reference and maintenance.

UML Diagram

UML (Unified Modeling Language) is essential for modeling the structure and behavior of the Offline AI Tutor system. It helps identify components, their roles, and interactions within the application.

Use Case Diagram :

A Use Case Diagram visually represents the interaction between users and the system, highlighting the functional requirements from the user's perspective. In the case of the Offline AI Tutor, actors include the User, and use cases involve actions such as initiating AI Conversation, using Speech Input, receiving Grammar Assistance, and selecting Voice Output. This diagram helps define the system boundary and clarifies expected user interactions, aiding in the design and implementation phases. Figure 4.1 shows the use case diagram of Offline AI Tutor.

4.1 SYSTEM FLOW DIAGRAM

A system flow diagram, also known as a flowchart, is a graphical representation of the logical sequence of operations or decisions within a software system. It visually outlines the flow of data and control from one part of the system to another.

International Research Journal of Education and Technology
Peer Reviewed Journal
ISSN 2581-7795

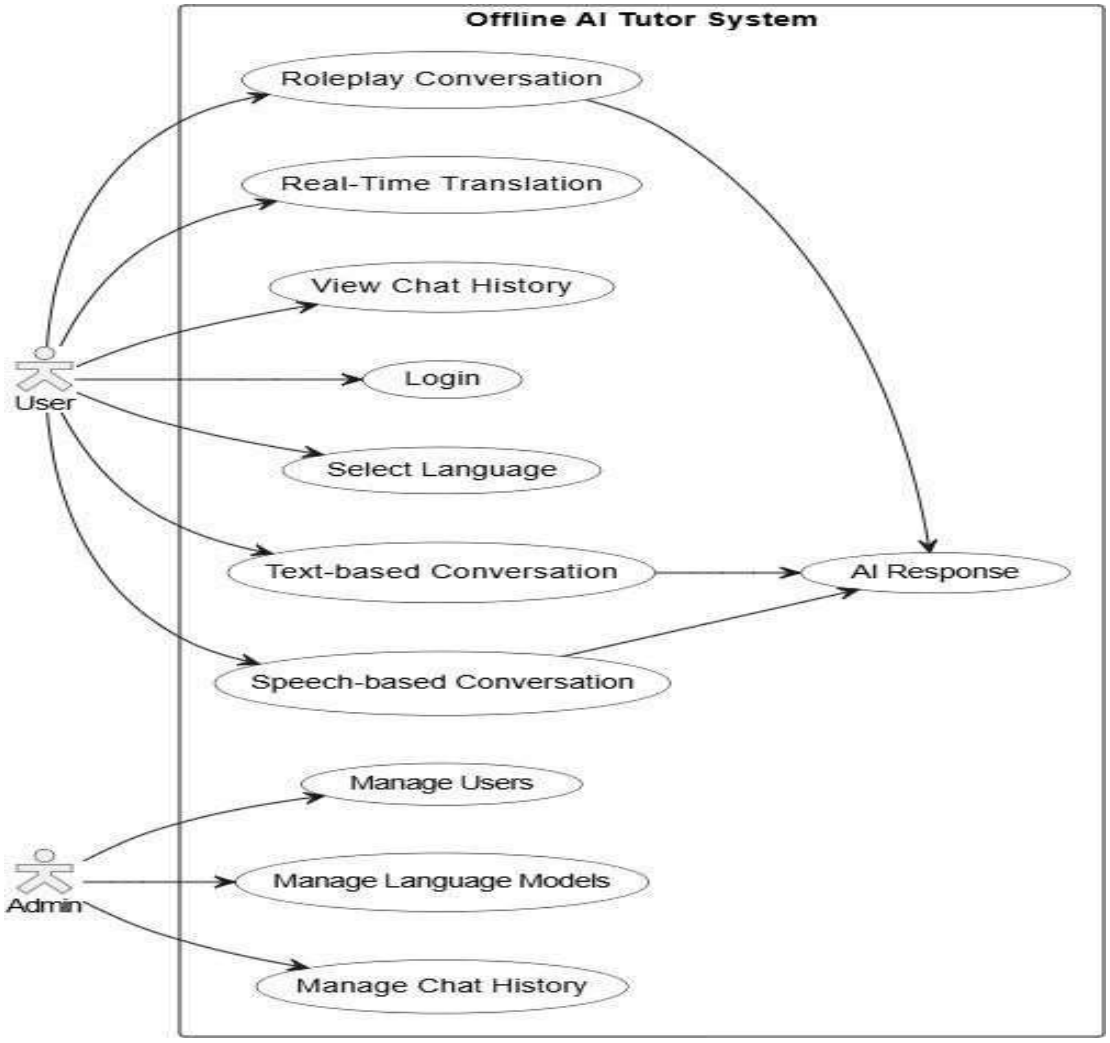
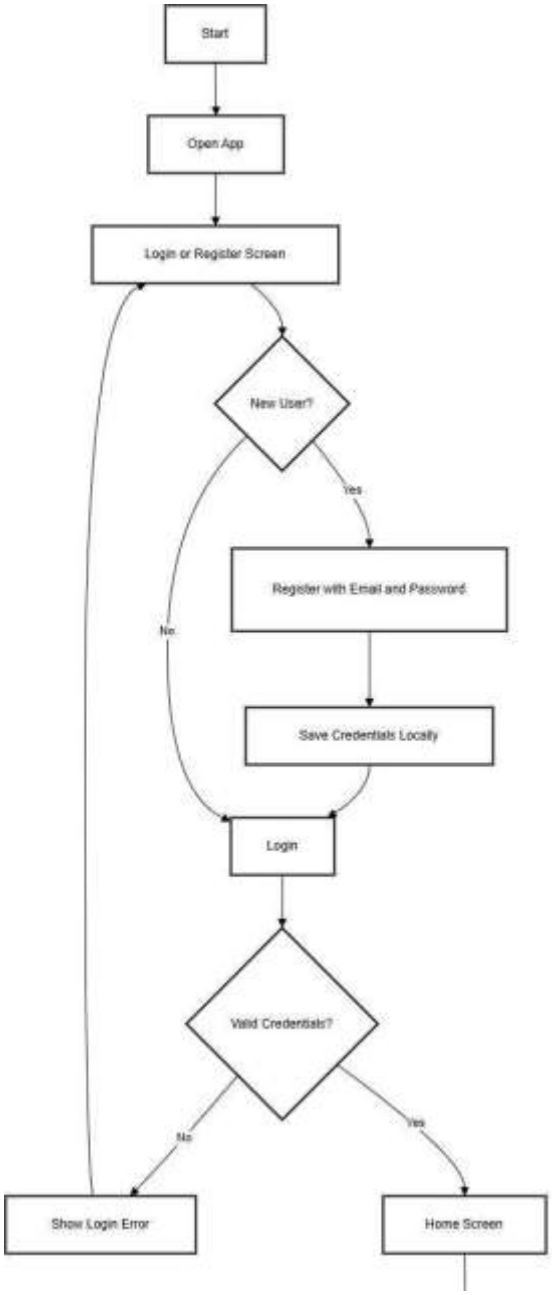


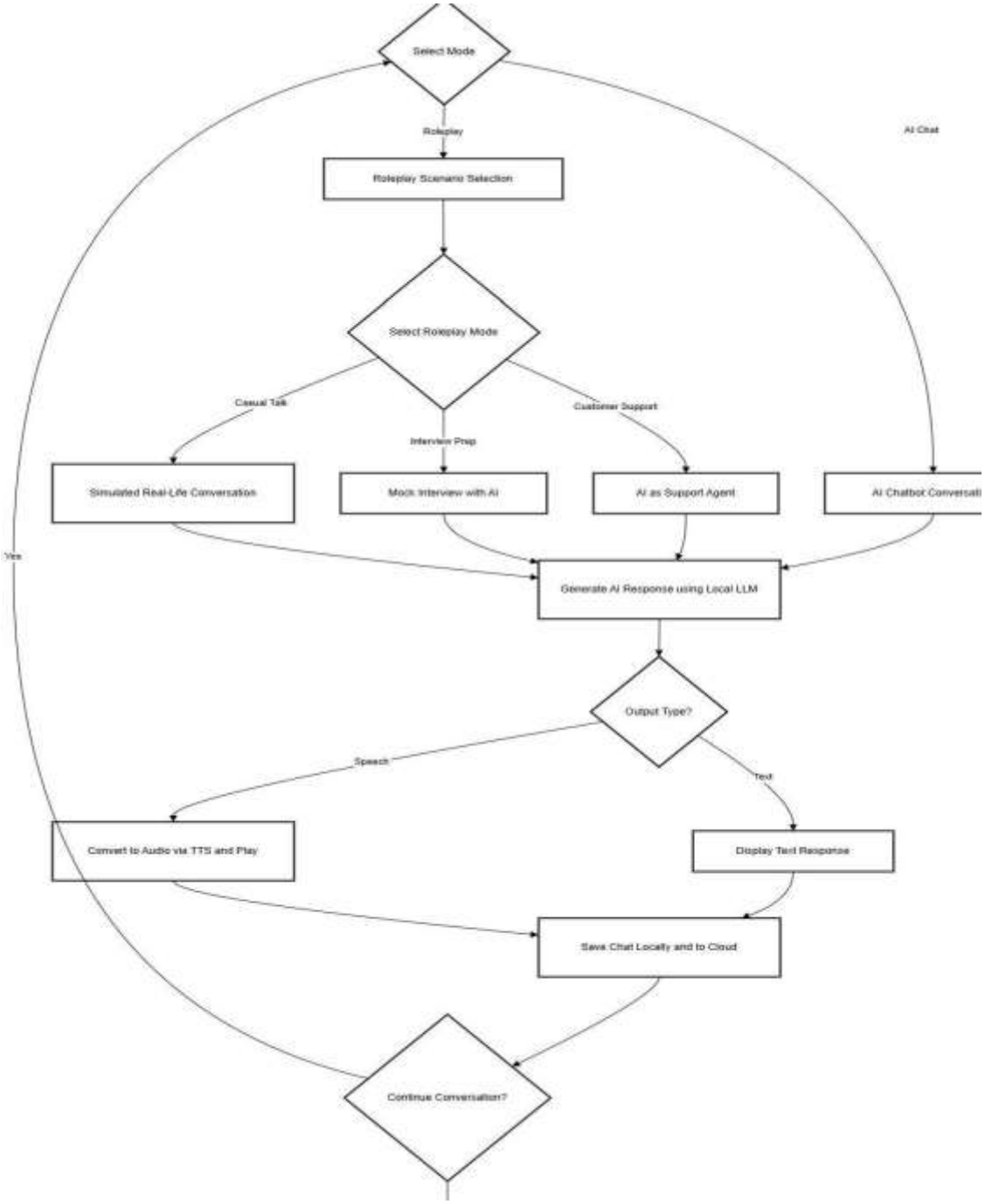
Figure 4.1 Use Case Diagram of Offline AI Tutor

International Research Journal of Education and Technology
Peer Reviewed Journal
ISSN 2581-7795



4.2 System Flow Diagram of Offline AI Tutor

International Research Journal of Education and Technology
Peer Reviewed Journal
ISSN 2581-7795



International Research Journal of Education and Technology
Peer Reviewed Journal
ISSN 2581-7795

4.2 DATABASE SCHEMA DIAGRAM

A database schema is the structural design that represents the logical view of the entire database, defining how data is organized and related. It outlines the entities, their attributes, relationships, and constraints, providing a clear blueprint for database design. Schema diagrams help developers understand and implement the database effectively.

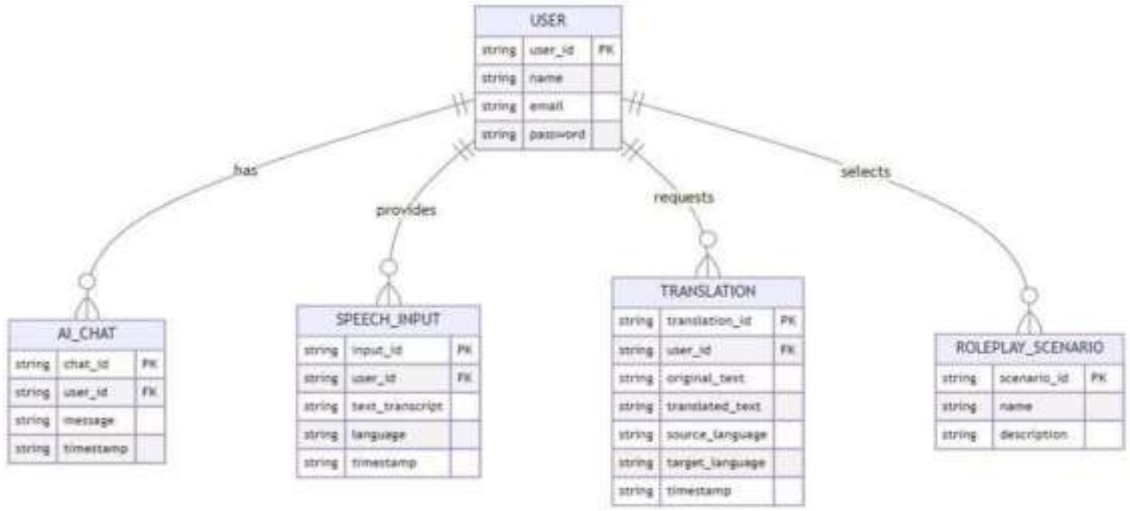


Figure 4.4 Database Schema Diagram of Offline AI Tutor

4.3 INPUT DESIGN

Input design is the process of transforming user data into a form that can be efficiently and accurately processed by the system. In the Offline AI Tutor application, the input design ensures that users can interact seamlessly with the system using both text and voice-based inputs, thereby supporting different learning preferences and accessibility needs.

International Research Journal of Education and Technology
Peer Reviewed Journal
ISSN 2581-7795

THE APPLICATION SUPPORTS TWO PRIMARY FORMS OF INPUT

TEXT INPUT

Users can type their queries, sentences, or words directly into the chat interface using the on-screen keyboard. Input validation mechanisms ensure that the entered text is properly formatted and free from invalid characters before being processed by the locally hosted Large Language Model (LLM). This design allows learners to practice grammar, spelling, and sentence construction in real time.

VOICE INPUT

For natural and hands-free interaction, the system integrates Automatic Speech Recognition (ASR) to capture spoken language and convert it into text. The Google Speech Recognition API (with offline support) is employed to handle multi-language voice input efficiently. To ensure usability, the design includes error handling features such as retry prompts and suggestions when speech recognition confidence is low.

The dual input design not only enhances flexibility and accessibility but also caters to both visual and auditory learners. It ensures a smooth user experience by providing instant feedback and allowing users to seamlessly switch between voice and text modes during conversations with the AI tutor.

4.4 OUTPUT DESIGN

Output design defines how information processed by the system is presented back to the user in a meaningful, accurate, and user-friendly manner. In the Offline AI Tutor application, the output is carefully structured to provide learners with real-time feedback, conversational responses, and personalized guidance, ensuring that interactions are both effective and engaging.

THE APPLICATION SUPPORTS TWO PRIMARY FORMS OF OUTPUT:

TEXT OUTPUT

The system displays AI-generated responses in a clean and responsive chat interface built using Jetpack Compose. These outputs include conversational replies, grammar corrections, sentence restructuring, and vocabulary suggestions tailored to the user's input. Text formatting ensures clarity, with highlighted corrections and suggestions to enhance the learning process.

VOICE OUTPUT

To support auditory learners and natural interaction, the application integrates Text-to-Speech (TTS) technology using engines such as Piper or VITS. The AI responses are converted into natural, human-like speech, allowing users to hear correct pronunciation and engage in hands- free learning. Adjustable voice speed, pitch, and language options ensure a personalized and realistic audio output experience.

The dual output design not only reinforces multimodal learning but also promotes accessibility, enabling learners to switch seamlessly between text-based and voice-based outputs. This ensures that the system caters to diverse learning preferences while delivering consistent, accurate, and context-aware feedback in real time.

CHAPTER-5

SYSTEM TESTING

This Shows the outlines the essential phase of system testing, highlighting various testing methodologies applied during the development of the Offline A.I. Tutor application. Testing was performed systematically to identify and resolve bugs, ensuring the app performs as expected in offline environments. The objective was to verify that all modules—AI interactions, chat history,

and authentication—function correctly with valid inputs and produce the expected outcomes, even without network connectivity.

TESTING TECHNIQUES

The testing process intricately examines the software's logical internals, affirming the completeness of code execution and scrutinizing functional aspects. It guarantees that specified inputs yield results consistent with the anticipated outcomes. Testing is integral to the development cycle, its extent contingent upon the application's size and complexity. This chapter elucidates the diverse testing strategies embraced in this project.

5.1.1 UNIT TESTING

Each core module—namely the Authentication Module, AI Interaction Module, and Chat History Module—was independently tested. For the authentication module, tests verified login and registration processes including error handling for invalid credentials. The AI module was tested for handling various input prompts (text/voice) and generating meaningful replies using the integrated LLM.

5.1.2 INTEGRATION TESTING

Integration testing ensured the seamless cooperation of all modules. The flow from user authentication to AI interaction, and subsequent saving of that conversation, was tested thoroughly. Special attention was given to the integration with the Ollama LLM interface, ensuring AI prompts were correctly processed and responses returned without errors. Offline functionality was tested to confirm independence from external services during interaction.

5.1.3 FUNCTIONAL TESTING

Functionality was validated against the expected use cases. For instance, when a user enters a prompt, the system must process it through the LLM and return an appropriate response. Chat history must reflect this interaction instantly. Authentication processes must allow or restrict access based on the correctness of credentials. Each feature was validated against its intended purpose.

5.1.4 USER INTERFACE TESTING

Graphical elements such as buttons, input fields, chat layout, and login screens were tested for usability and clarity. The UI was checked across different screen sizes to ensure proper alignment, readable fonts, and intuitive interaction. Error messages and notifications were verified to display correctly when inputs were invalid or operations failed.

5.1.5 TESTCASE REPORT

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and code generation. Once the source code has been generated,software must be tested to uncover as many errors as possible before delivery to the customer. In order to find the highest possible number of errors, tests must be conducted systematically and test cases must be designed using disciplined techniques.

TABLE 5.2 TEST CASE REPORT

S.No	Test ID	Test Case	Description	Expected Outcome

1	TC001	User Authentication	Verifies whether users can successfully log in with valid credentials.	Users are authenticated and navigated to the main chat interface.
2	TC002	AI Interaction (Text Input)	Tests text-based interaction with the offline LLM.	The LLM generates a relevant and context-aware response displayed in the chat window.
3	TC003	AI Interaction (Voice Input)	Verifies voice input functionality.	Voice is transcribed and the LLM provides an appropriate response.
4	TC004	Chat History Save	Checks if chat history is saved after each interaction.	Messages are stored and can be retrieved later.
5	TC005	Chat History Load	Tests chat history persistence after app restart.	Stored messages are correctly reloaded and displayed.
6	TC006	Flow Completion	Tests the complete flow from login to AI interaction to chat history viewing.	All modules work seamlessly, and user experience is uninterrupted.

CHAPTER-6

SYSTEM IMPLEMENTATION AND MAINTENANCE

6.1 SYSTEM IMPLEMENTATION

System implementation is the stage where the planned system design is transformed into a working application. The Offline AI Tutor has been implemented using Android Studio as the development environment, with Kotlin as the primary programming language and Jetpack

Compose for the user interface. The modular design ensures smooth integration of the frontend, backend, and database components.

- Frontend Implementation: The user interface was developed using Jetpack Compose, providing a modern, responsive, and user-friendly layout. Features such as text input fields, voice recording buttons, chat displays, and role-play options were designed to ensure ease of use and accessibility.

- Backend Implementation: The core AI processing was implemented by integrating locally hosted Large Language Models (LLMs) such as GEMMA and DeepSeek using frameworks like MediaPipe or Ollama. These models enable grammar correction, vocabulary suggestions, and conversational interactions in real time without internet dependency.

- Speech Module: Automatic Speech Recognition (ASR) was integrated using Android's Speech Recognition API, while Text-to-Speech (TTS) was implemented using tools such as Piper and VITS. This enables natural and interactive voice-based communication with the tutor.

- Database Implementation: Local data management was handled through Room Database / SQLite, which stores chat history, user preferences, and learning progress offline. An optional integration with Firebase Firestore allows cloud backup and multi-device synchronization when internet access is available.

- Security Implementation: Security was ensured by using Encrypted Shared Preferences for local data protection and Firebase Authentication with OAuth 2.0 for secure user login and identity management.This modular approach ensures that each component functions effectively on its own while integrating seamlessly with the overall system, resulting in a secure, offline-first language learning platform.

6.2 SYSTEM MAINTENANCE

System maintenance is essential to ensure that the Offline AI Tutor remains functional, efficient, and relevant after deployment. The application has been designed with a scalable and maintainable architecture to support future updates and enhancements.

- Corrective Maintenance: Any errors, bugs, or performance issues identified during usage will be addressed promptly. Regular debugging and code refinement will ensure system stability.

- Adaptive Maintenance: As new versions of Android and mobile hardware are released, the system will be updated to maintain compatibility. Integration with newer LLMs or speech engines can also be done without affecting the core architecture.

- Perfective Maintenance: User feedback will be incorporated to improve features such as personalization, role-play modes, and multilingual support. Updates may include enhanced grammar correction, better vocabulary suggestions, and advanced UI improvements.

- Preventive Maintenance: Routine optimization of storage, memory usage, and battery efficiency will be carried out to prevent performance degradation. Security patches and encryption upgrades will be applied to ensure continued data protection.

- Through effective implementation and ongoing maintenance, the Offline AI Tutor remains a reliable, secure, and scalable language learning solution that evolves alongside user needs and technological advancements.

CHAPTER-7

7. CONCLUSION

The Offline AI Tutor with Local LLM successfully demonstrates the potential of integrating advanced artificial intelligence into a mobile application that operates entirely without internet

connectivity. By leveraging locally hosted models such as GEMMA and DeepSeek, the system provides an intelligent, interactive, and private learning environment, overcoming the limitations of conventional cloud-based language learning tools.The project achieved its primary objectives of ensuring data privacy, zero-latency responses, and continuous accessibility, while supporting core language learning features such as grammar correction, vocabulary building, sentence restructuring, and real-time conversational practice. The integration of Automatic Speech Recognition (ASR) and Text-to-Speech (TTS) further enriched the learning experience by enabling natural, voice-based interactions, thereby improving users' listening and speaking skills alongside reading and writing.Testing and evaluation confirmed that the application is optimized for mobile devices, running smoothly on consumer-grade hardware while maintaining security and reliability. Its modular design, with offline-first architecture and optional cloud support, ensures scalability, adaptability, and long-term usability.In conclusion, the Offline AI Tutor stands as an innovative and reliable solution for students and language learners seeking a secure, intelligent, and always-available educational companion. By combining cutting-edge AI with an offline-first approach, the project redefines how learners engage with technology, bridging the gap between accessibility, privacy, and performance in modern education.

FUTURE ENHANCEMENT

The Offline AI Tutor has successfully achieved its core objectives, there are several opportunities for future enhancement to make the system more powerful, adaptive, and engaging for users. These enhancements would extend the application's capabilities and ensure it continues to meet evolving educational and technological needs.

- Expansion of Language Support – Add more languages, dialects, and accents to make the system accessible to a wider user base.

- Gamification Features – Introduce rewards, leaderboards, and streak tracking to boost user motivation and engagement.

- Learning Analytics Dashboard – Provide progress tracking, personalized feedback, and visual reports to guide learners effectively.

- Personalized Learning Paths – Implement adaptive AI that customizes lessons and recommendations based on user performance.

- Model Upgrades and Optimization – Continuously integrate newer, lightweight, and efficient AI models to improve speed and accuracy on mobile devices

8. **BIBLIOGRAPHY**

1. Google AI. (2024). MediaPipe LLM Inference: On-Device Large Language Models. Retrieved from https://ai.google.dev/edge/mediapipe/solutions/genai/llm_inference

2. Hugging Face. (2024). Gemma: Lightweight Language Models for On-Device Applications. Retrieved from https://huggingface.co/docs/transformers/en/model_doc/gemma

3. Google Developers. (2023). Kotlin for Android Development. Retrieved from https://developer.android.com/kotlin

4. Firebase. (2024). Firebase Authentication and Firestore Documentation. Retrieved from https://firebase.google.com/docs/

5. Android Developers. (2023). SpeechRecognizer API Documentation. Retrieved from https://developer.android.com/reference/android/speech/SpeechRecognizer

6. Android Developers. (2023). TextToSpeech API Documentation. Retrieved from https://developer.android.com/reference/android/speech/tts/TextToSpeech

7. TensorFlow. (2023). TensorFlow Lite: On-Device Machine Learning. Retrieved from https://www.tensorflow.org/lite/guide

8. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems 32.

9. Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media.

9. APPENDIX

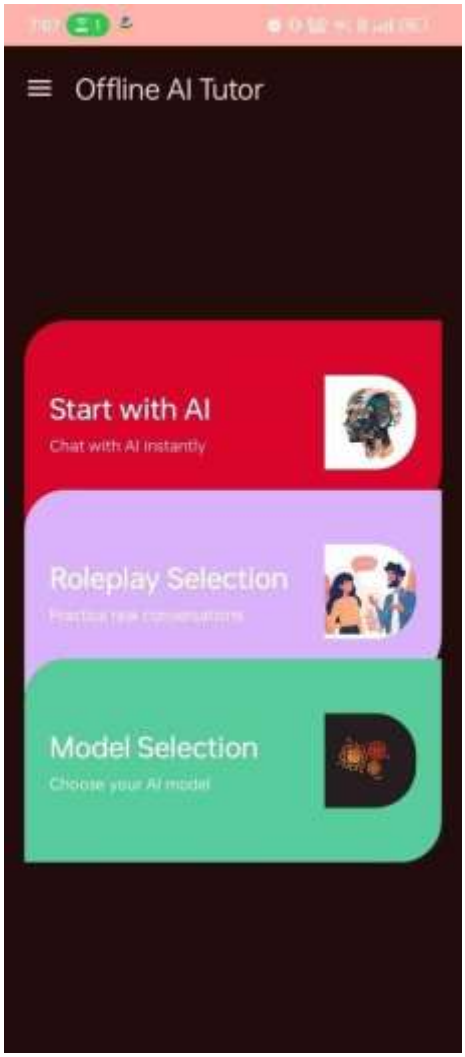
SAMPLE SCREENSHOTS

International Research Journal of Education and Technology
Peer Reviewed Journal
ISSN 2581-7795

Figure 8.1 Sign up Page

Figure 8.2 Login Page

International Research Journal of Education and Technology
Peer Reviewed Journal
ISSN 2581-7795



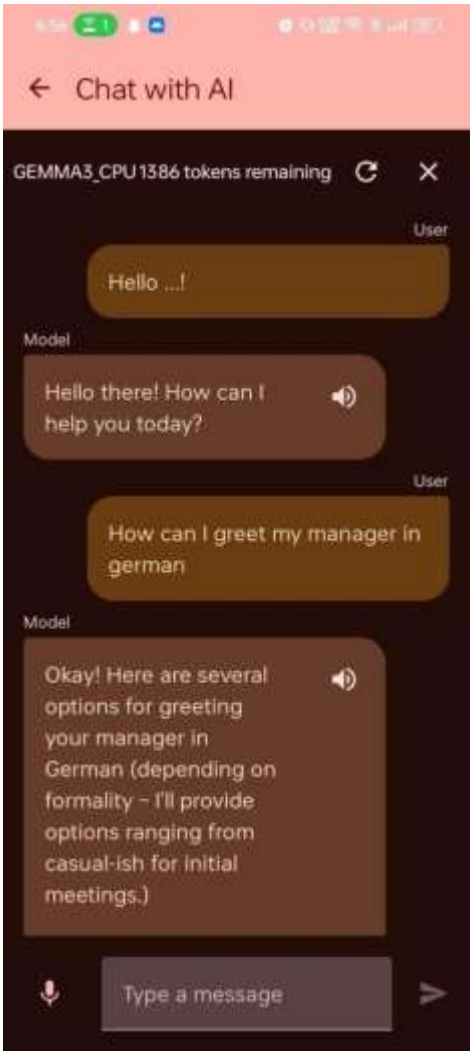


Figure 8.3 Home Page *Figure 8.4 Chat with AI*

International Research Journal of Education and Technology
Peer Reviewed Journal
ISSN 2581-7795

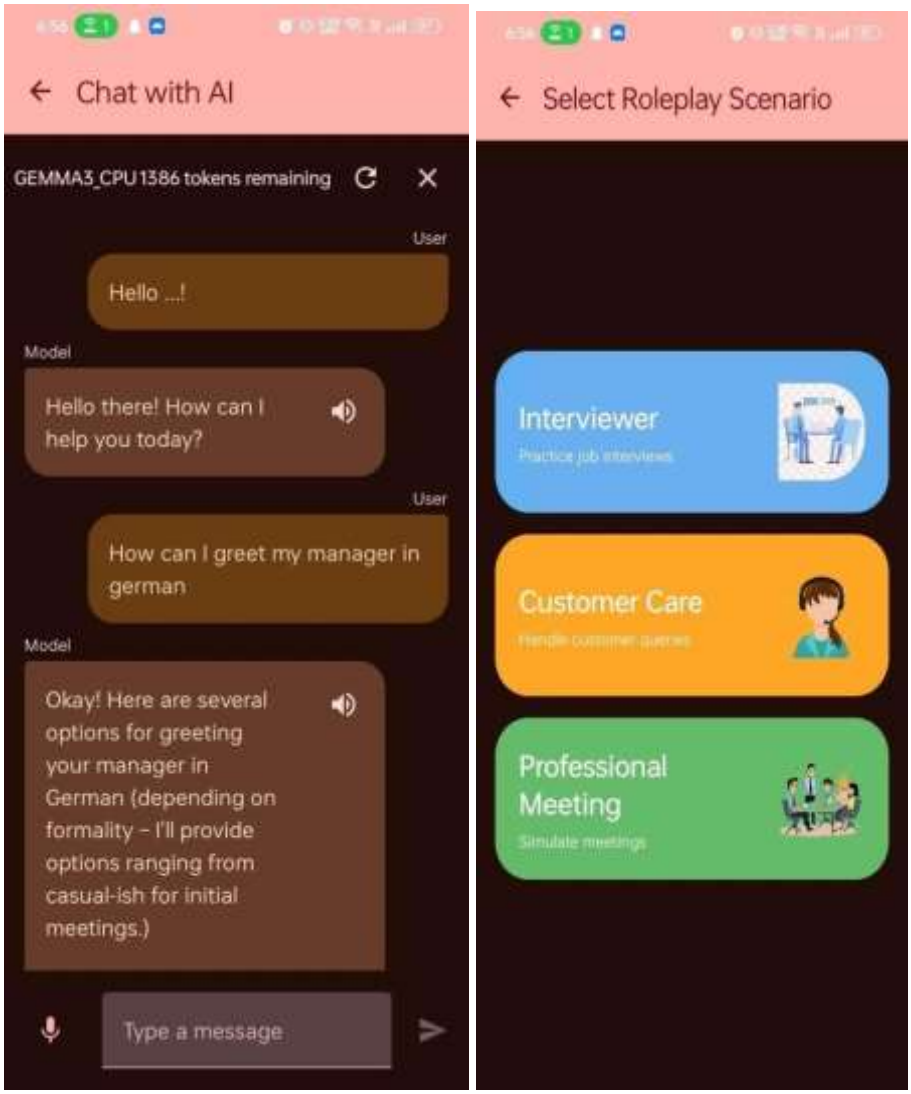
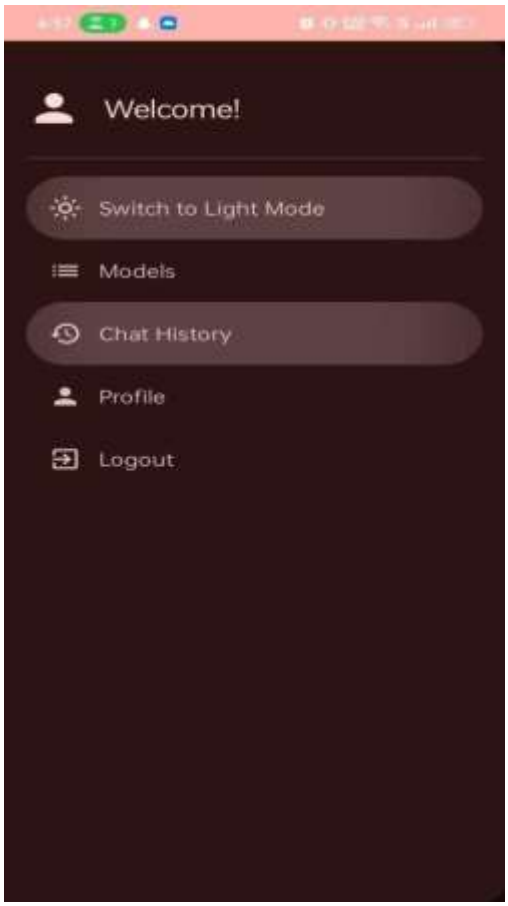


Figure 8.5 Roleplay Figure 8.6 Select Roleplay

International Research Journal of Education and Technology
Peer Reviewed Journal
ISSN 2581-7795



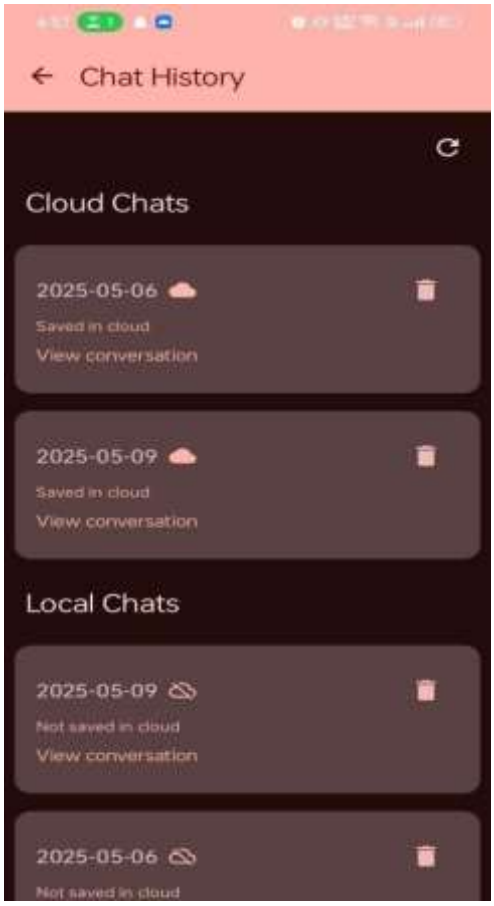


Figure 8.7 Menu

Figure 8.8 Chat history

SAMPLE CODING

```
package com.google.mediapipe.examples.llminference


import android.net.Uri import android.os.Bundle import android.util.Base64 import android.util.Log

import android.widget.ImageButton
```

```
import androidx.appcompat.app.AppCompatActivity import
net.openid.appauth.AuthorizationRequest import net.openid.appauth.AuthorizationService
import net.openid.appauth.ResponseTypeValues import java.security.MessageDigest
import java.security.SecureRandom

class LoginActivity : AppCompatActivity() {
private lateinit var authService: AuthorizationService private lateinit var codeVerifier: String
private lateinit var codeChallenge: String

override fun onCreate(savedInstanceState: Bundle?) { super.onCreate(savedInstanceState)
setContentView(R.layout.activity_login)

authService = AuthorizationService(this)

val loginButton: ImageButton = findViewById(R.id.btnLogin) loginButton.setOnClickListener
{
loginWithHuggingFace() // Start OAuth login when button is clicked
}
}
```

```
private fun loginWithHuggingFace() {

// Generate PKCE parameters codeVerifier = generateCodeVerifier()

codeChallenge = generateCodeChallenge(codeVerifier)


// Save the code verifier securely for later use in token exchange

SecureStorage.saveCodeVerifier(applicationContext, codeVerifier)


val authRequest = AuthorizationRequest.Builder( AuthConfig.authServiceConfig,

AuthConfig.clientId, ResponseTypeValues.CODE, Uri.parse(AuthConfig.redirectUri)

).setScope("read-repos") // Adjust scopes if needed

.setCodeVerifier(codeVerifier, codeChallenge, "S256") // Include PKCE

.build()


val authIntent = authService.getAuthorizationRequestIntent(authRequest)
startActivity(authIntent) // Launch OAuth login page

}


private fun generateCodeVerifier(): String { val random = ByteArray(32)

SecureRandom().nextBytes(random)

return Base64.encodeToString(random, Base64.URL_SAFE or Base64.NO_PADDING or
Base64.NO_WRAP)
```

```
}

private fun generateCodeChallenge(codeVerifier: String): String { val bytes =
codeVerifier.toByteArray()
val digest = MessageDigest.getInstance("SHA-256").digest(bytes)
return Base64.encodeToString(digest, Base64.URL_SAFE or Base64.NO_PADDING or
Base64.NO_WRAP)
}
}
```