Bubbles – Smarter AI Assistant

Final Year Project – Mid Report

Session 2022-2026

A project submitted in partial fulfillment of the
requirements for the Degree

of

BS in Computer Science

Department of Computer Science

COMSATS University Islamabad (CUI), Lahore Campus

Project Details

Project ID (for office use)	
Type of project	[✓] Traditional [ ] Industrial [ ] Continuing
Nature of project	[ ] Development [✓] Research & Development
Sustainable Development Goals(SDGs)	[ ] Good Health and Well-Being [✓] Quality Education [ ] Industry, Innovation, and Infrastructure [ ] Gender Equality [ ] Decent Work and Economic Growth [ ] Climate Action
Area of specialization	[✓] Artificial Intelligence (AI) [ ] Blockchain [ ] Cybersecurity [ ] Data Science and Analytics [ ] Game Development [ ] Internet of Things (IoT) [✓] Natural Language Processing (NLP) [✓] Mobile App Development [ ] Web Development

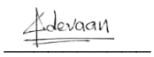
Project Group Members

Sr.#	Reg. #	Student Name	Email ID	Signature
(i)	Group Leader	FA22-BCS-025	qdevaan@gmail.com	
(ii)		FA22-BCS-164	sahitoattique@gmail.com	

Declaration: The candidates confirm that the work submitted is their own and appropriate credit has been given where reference has been made to the work of others.

Plagiarism Free Certificate

This is to certify that, I am Muhammad Ahmad S/D/o Umer Farooq, group leader of FYP under registration no CUI/FA22-BCS-025/LHR at the Computer Science Department, COMSATS University Islamabad, Lahore Campus. I declare that my FYP proposal is checked by my supervisor and the similarity index is _____% that is less than 20%, an acceptable limit by HEC. The report is attached herewith as Appendix A.

Date: _____ Name of Group Leader: _____ Signature: 

Name of Supervisor: _____ Co-Supervisor (if any):_____

Designation: _____ Designation: _____

Signature: _____ Signature: _____

HoD: _____

Signature: _____

ABSTRACT

To overcome the memory limitations of traditional AI, we developed this project Bubbles, a smart personal assistant. Typical voice assistants manage every interaction independently, whereas Bubbles utilizes a Hybrid Memory Architecture that combines a structured knowledge graph with long-term contextual storage. This allows the system to understand the intricate connections between the people, places, and occurrences in a user's life while also retaining information.

The system operates primarily in two modes: a real-time Wingman and an analytical Consultant. In the Wingman position, the assistant attentively listens to ongoing discussions while analyzing speech in real-time to provide the user with immediate, context-sensitive guidance and "whisper" pertinent information. To update its internal knowledge base, it concurrently gathers new data. In the Consultant mode, users can engage in detailed retrospective questioning sessions, where the AI compiles insights from its gathered history to deliver enhanced responses about the user's past activities and decisions.

This Project introduces a rather untraditional approach for tailored AI through the use of Graph-Retrieval Augmented Generation (GraphRAG). It extends simple command-response interactions to create a continuous, adaptive companion that evolves with the user, providing contextually appropriate and factually based assistance

Acknowledgement

The acknowledgement section is **OPTIONAL** and should comprise a short, formal note to express gratitude to the people, organizations, or institutions that supported you during the preparation of the report. It is not very long, typically one or two paragraphs.

Table of Contents

List of Tables

List of Figures

Chapter 1. Introduction

1.1 Introduction

Good communication serves as the cornerstone of professional success, academic achievement, and social interaction. The world is becoming more digital and fast-paced, requiring people to be able to articulate concepts clearly, recognize subtleties in tone, and retain important information from spoken conversations. Speaking is still difficult because it happens quickly, and we frequently don't realize our mistakes until the interaction is ended, but writing has greatly improved thanks to sophisticated checking tools. In recent years, artificial intelligence (AI) and natural language processing (NLP) have advanced quite quickly. The way computers comprehend us has evolved because of technologies like Large Language Models (LLM) and Automatic Speech Recognition (ASR). However, the majority of apps nowadays only perform one task at a time, such as transcribing, summarizing, or correcting grammar. They don't actually collaborate to provide real-time assistance. With the use of cutting-edge deep-learning models and contemporary tools like LangChain, this project bridges the gap between mobile app development, natural language processing, and human-computer interaction (HCI).

This project is motivated by a typical issue that many professionals and students encounter: we often fail to notice minor but crucial elements during talks, and we only become aware of our errors like using the inappropriate tone or stating something incorrectly—only after the interaction is over. In situations like business meetings, interviews, or lectures, missing an important point or sending the wrong message can seriously affect the outcome.

Currently, there are numerous note taking apps that uses AI like Otter.ai, Fathom or Notta. These applications work well for note taking during meetings and extracting key informations. But they do not provide an interface where the user can ask questions from the AI and get assistance to make the conversation more beneficial. Not only this but some apps have huge learning curves, missing key information and absurdly large prices. These apps are build with team-first-approach hence they do not provide feedback to improve on communication skills and their closed-source nature makes people hasitant to use.

BubblesAI tackle all these challenges with very simple implementation. BubblesAI is built for 1 user helping with just 1 skill, and that is communication improvement. The app would be opensource under Apache-V2 license so community can improve on the application and models used in it. All the models used will be open-weights, the fine-tunning techniques and methodologies used will be open to anyone who wants to contribute. The app would have only three modes live-wing, consultant and speak-improve to keep the app-adaptibilty as easy as possible. In this way we cut down the costs, reduce learning curve and increase the information extraction to maximum.

The main goal of this project is to build a smart mobile assistant that can capture, analyze, and improve conversations in real time. More specifically, the project aims to:

- Assist 1 user only, with their day to day meetings.
- Build a reliable ASR-LLM-KGraph system that uses modern models to answer accurately.
- Keep track of all the user interactions using a knowledge graph.
- Add smart analysis tools that can detect tone, suggest better wording, and explain difficult or technical terms as the conversation happens.
- Use speaker diarization so the system can tell who is speaking, even when there are multiple people in the conversation.
- Create a gamified dashboard that shows progress, points out repeated mistakes, and rewards improvement to keep users motivated and learning.

The scope of the project is building a mobile app in Flutter with a Python backend. The app will facilitate official, semi-formal, and informal conversations. It is currently limited to English and requires an internet connection and a microphone in order for the backend to conduct the intensive NLP processing in the cloud.

There are layers in the system. The primary layer manages user interaction; it displays keypoint summaries and leads to new sessions, either live or consultant. The next layer processes the data by converting speech to text and cleaning the transcript by removing noise and filler words. The main work happens in the AI layer. Here, LangChain manages how the data moves between different NLP models. We use speaker diarization to figure out who is talking and large language models (like Llama or Qwen) to summarize conversations and analyze tone and sentiment. The questions are then sent to the KGraph-RAG system, which retrieves any previous related information for the LLM to respond to the user query. After that, we use a feedback process, in which it identifies grammatical and tone errors, makes recommendations on how to be better and improve further, and also stores this data in a user profile for later review. The gamification elements, such as the "Replay and Review" option that allows users to review past interactions and see concise, helpful feedback and improve their skillsThis project is expected to produce a fully working mobile app that gives users a complete set of tools to improve how they communicate and an interface for assistance during meetings. The main results will include:

- A highly accurate transcription system that works even when multiple people are talking.
- A summarization feature that pulls out entities and details like names, dates, and action items from long conversations.
- Proper answers to user queries from past conversations.
- An interactive dashboard display trends in speech clarity, vocabulary, and tone.
- A clear reduction in communication mistakes over time, measured through the app’s progress tracking.
- Suggestions on how to improve past mistakes.

The goal of this project is to facilitate in education using NLP and power of LLMs such as:

- Combining multiple NLP tasks in one place: Instead of offering just summarization or just grammar correction, this system brings together transcription, speaker diarization, sentiment analysis, and helpful feedback in a single workflow.
- Real-time soft skills coaching: The app provides immediate suggestions during conversations, such as better responses or explanations of technical terms.
- Gamified communication improvement: It introduces a new way to track progress using points, levels, and error-frequency statistics—like a habit-building app but focused on communication.

1.2 Problem Statement

Often success depends on our effective communication, but it can be quite difficult to identify and correct errors while speaking. The majority of AI solutions that exist currently concentrate on transcription or post-meeting notes. And most of the time in conversation that is recorded, it isn't aware of the context or doesn't have enough details to provide good real-time feedback to the user. A tailored, privacy-conscious assistant that records conversations and offers real-time guidance on clarity, tone, and word choice is needed. Without the high cost or privacy issues, and without concerning yourself with technologies, this method can transform routine contacts into learning opportunities, assisting users in dynamically improving their communication abilities.

1.3 Proposed Solution

So, we propose the development of our app BubblesAI, a smart mobile app that acts as a personal assistant. It uses a client-server architecture with a Flutter mobile interface to capture live audio send it to Python backend that uses LangChain and Large Language Models (LLMs) like Llama or Qwen for natural language processing. Key components include an Automatic Speech Recognition (ASR) engine for accurate transcription, a Speaker Diarization module to tell speakers apart, and a Knowledge Graph enhanced with RAG to give context-aware answers from past conversations. To tackle the limitations of simple RAG, we use Knowledge Graph RAG in addition with vectors which has nodes and relations that mimic real-life interactions and can extract minute details on runtime just like a natural human conversation but smarter. By adding a gamified feedback loop, the system provides suggestions on tone and vocabulary like what could have been the better choice, giving users actionable insights to improve their soft skills.

1.4 Main Objectives

This app's primary objective is to develop an AI-powered assistant that facilitates improved real-time communication, conversation management, and future improvement. The particular goals we are attempting to accomplish are:

- To Develop a mobile app using Flutter for live audio capture and interactive frontend.
- Build a Python backend to manage ASR, LLM, and Speaker Diarization models via LangChain and communicate with the frontend flutter app.
- To Create a Neo4j powered, Knowledge Graph-based RAG system to use past conversations for accurate, context-aware responses.
- To Use open-weight LLMs for real-time tone analysis, summarization, and explaining technical terms.
- Gather datasets and utilize them to refine models for improved performance and accuracy.
- Create a gamified dashboard that uses metrics, points, and incentives to monitor progress.
- Assess performance using user engagement, response accuracy and speed, and transcription accuracy (WER).

1.5 Assumptions & Constraints

Assumptions:

- The user has a stable, fast internet connection to connect with the cloud backend.
- The app will be used in moderately quiet environments for accurate ASR performance.
- Conversations will mainly be in English.

Constraints:

- Processing Power: Response speed depends on the backend GPU resources running models.
- Latency: Network delays can affect real-time feedback.
- Data Privacy: User data must be protected and kept private.

1.6 Project scope

Bubbles Ai's scope includes creating an Android or iOS mobile application with a Python backend. Real-time speech-to-text transcription, speaker diarization for many speakers, and three modes: Live-wing (support in real time), Consultant (question and answer from previous discussions), and Speech-improve (coaching and feedback) are among its key features. User progress will be monitored by a gamification process. Video analysis and facial recognition are not included in the project, and all significant LLM processing takes place on the server (no offline processing on mobile devices). The software only supports English at this time.

1.7 Software Development Lifecycle Model

The project will use an iterative and incremental development model, which is ideal for AI and NLP R&D-based projects because it allows for continuous improvements of models and settings based on testing. As technology improves, we can introduce new technologies and algorithms to our projects.

Justification: This approach lets complex modules, such as speaker diarization, text transcription, entity extraction, or the RAG system, be developed and tested separately before being integrated into the main app.

Structure: The project will go through cycles of planning, requirement analysis, design, implementation, building modules, testing, validating models, and evaluation. This helps the team to catch errors early and allows the system to improve step by step towards the final solution with the latest tools and technologies, all equipped.

Chapter 2. Requirement Analysis

2.1 Literature Review

2.1.1 Introduction

Speaker diarization the concept to find who spoke and when and accurate transcription are foundational to many speech applications that claim to solve these problems. Traditional ASR systems used cascaded acoustic, pronunciation and language models, which are complex and inflexible. Recent industry analyses note that "most ASR systems rely on a combination of legacy systems that are slow, inaccurate, and inflexible," and advocate fully end-to-end deep learning approaches[12]. At the same time, commercial speech APIs raise concerns of cost, latency and data privacy. This motivates building custom on-premise solutions that leverage the latest research. In particular, by combining modern diarization with open-weight ASR (e.g. Whisper or Conformer-based models), a system can achieve state-of-art accuracy while retaining control over data and allowing domain-specific adaptation.

2.1.2 Evolution of Speaker Diarization

2.1.2.1 Foundational Approaches (Clustering & Embeddings)

Early diarization systems were modular: they detected speech segments (Voice Activity Detection), extracted speaker embeddings for each segment, and then clustered those embeddings. For example, x-vectors[1] – fixed-dimensional embeddings from a speaker-discriminative DNN – became a standard representation. Snyder *et al.* (2018) showed that

x-vectors, trained on large datasets, "achieve superior performance" compared to traditional i-vector baselines[2]. In practice, x-vectors from a DNN are extracted frame-wise and pooled per segment, then compared using a backend like PLDA (Probabilistic Linear Discriminant Analysis). Snyder *et al.* (2019) applied this to multi-speaker recordings: they extract x-vectors for short speech segments and compute pairwise PLDA scores, then apply agglomerative hierarchical clustering (AHC) to group segments by speaker[2]. This simple pipeline (x-vectors + PLDA + AHC) achieved state-of-the-art diarization accuracy on benchmarks, significantly reducing error rates in the presence of multiple speakers.

Subsequent work refined the clustering step[3]. Landini *et al.* (2022) proposed **VBx**, which replaces heuristic AHC with a Bayesian Hidden Markov Model over the x-vector sequence[3]. The VBx model clusters speakers in time under a Bayesian framework and has been shown to outperform earlier methods: on standard datasets (CALLHOME, AMI, DIHARD), VBx "achieves superior performance" over previous approaches[3]. In summary, the foundational diarization approach became: VAD → extract DNN x-vectors → model speaker similarity with PLDA → cluster (initially by AHC, later by VBx). These modular systems are robust and reproducible, but they assume one speaker per short segment, so they can fail under heavy overlap.

2.1.2.2 Shift to Deep Learning & End-to-End Models

A key limitation of the modular pipeline is **overlapping speech**. In conversational audio, multiple people often speak simultaneously, but traditional clustering assigns exactly one speaker per segment. To handle overlap more naturally, researchers developed **end-to-end neural diarization** models that directly output speaker activities over time.

One of the first was EEND (End-to-End Neural Diarization) by Fujita *et al.* (2019). Instead of separate modules, their single neural network takes an acoustic feature sequence and outputs a multi-label sequence indicating which of (a fixed number of) speakers are active at each frame[4]. Crucially, they use a permutation-free training loss that directly optimizes diarization error. A major advantage is that the model can be trained with overlapping speech, and indeed Fujita *et al.* report that their EEND model "explicitly handles overlapping speech" and yields much lower error: on simulated two-speaker mixtures, their EEND achieved a diarization error rate (DER) of 12.28%, compared to 28.77% for a conventional clustering system[4]. This demonstrates that end-to-end models can learn to untangle overlaps when given the right loss function.

Early EEND models assumed a fixed number of speakers. Horiguchi *et al.* (2020) addressed this by introducing **EEND-EDA** (Encoder-Decoder-based Attractors). Here, a self-attentive encoder produces frame-wise embeddings, and an LSTM encoder-decoder generates a set of "attractor" vectors that represent the different speakers[5]. The attractors automatically determine how many speakers to attend to. This makes the model flexible to varying speaker counts. In experiments, Horiguchi *et al.* showed EEND-EDA outperforming plain EEND: for example, in a two-speaker scenario their model reached 2.69% DER on simulated mixtures (vs. 4.56% for vanilla EEND), and in an open-set scenario it achieved 15.29% DER on the CALLHOME dataset versus 19.43% for an x-vector clustering baseline[5]. Thus EEND-EDA approaches the overlap-handling strengths of EEND while adding adaptability in speaker count.

Another hybrid approach is **TS-VAD** (Target-Speaker Voice Activity Detection) by Medennikov *et al.* (2020). TS-VAD combines clustering with neural refinement. First, an initial diarization (e.g. x-vector + AHC) is used to estimate an i-vector for each discovered speaker. Then a neural network takes the original audio features plus all speaker i-vectors as inputs, and outputs frame-wise activity probabilities for each target speaker (i.e. "does speaker 1 speak at time t?" and so on)[6]. This allows the model to revisit the audio and assign overlapping speech properly to multiple speakers. In the challenging CHiME-6 dataset, TS-VAD drastically improved performance: it achieved **over 30% absolute reduction in DER** compared to a strong x-vector baseline[6]. TS-VAD's strength comes from conditioning on known speakers; it effectively resolves heavy overlaps by checking for the presence of each speaker at every frame.

2.1.2.3 Open-Source Frameworks

The research above has been greatly facilitated by open-source toolkits. Notably, **pyannote.audio** (Bredin *et al.*, 2020) provides a modular, trainable pipeline for diarization[7]. Built on PyTorch, pyannote.audio offers ready-to-use neural components for voice activity detection, speaker embedding, speaker change detection, overlapping speech detection, etc., along with pretrained models. According to the authors, the toolkit "comes with pre-trained models covering a wide range of domains... reaching state-of-the-art performance for most of them"[7]. Researchers can combine and fine-tune these blocks to reproduce classic systems (e.g. x-vector clustering) or build new EEND models. In practice, pyannote.audio has become a popular foundation for diarization research and development, enabling rapid prototyping of custom pipelines.

2.1.3 Automatic Speech Recognition (Transcription)

For transcription, the trend mirrors that in diarization: large neural networks trained at scale can match or beat traditional methods. A breakthrough was **Whisper** (Radford *et al.*, 2023), a weakly-supervised model trained on **680,000 hours** of multilingual audio[8]. Whisper's massive scale yields remarkable robustness: Radford *et al.* report that their models "generalize well to standard benchmarks" and often rival the best supervised systems without any fine-tuning[8]. By "approaching human accuracy and robustness" on varied speech, Whisper suggests that collecting vast unlabeled data can substitute for carefully curated datasets. In practical terms, Whisper handles diverse accents, noise, and even translates between languages, making it competitive with commercial APIs out-of-the-box.

Alongside data scale, self-supervised learning has advanced ASR. For example, **wav2vec 2.0** (Baevski *et al.*, 2020) learns speech representations by masking segments of raw audio and predicting them via contrastive learning[9]. When fine-tuned on labeled data, wav2vec 2.0 achieves state-of-art results. Strikingly, it excels even with little labeled data: using all 960h of LibriSpeech, it gets ~1.8% WER on clean speech; with only 100h of labels it still outperforms prior methods; and even with just **10 minutes** of labels (plus pretraining on 53,000h unlabeled) it obtains about 4.8% WER on clean speech[9]. This shows that high-quality ASR is possible with limited transcription effort, provided pretraining is leveraged.

Architecturally, modern ASR models use Transformers augmented with convolution. **Conformer** (Gulati *et al.*, 2020) integrates convolutional layers for local feature extraction with Transformer layers for global context[10]. This combination captures both short-term acoustic patterns and long-range dependencies. In experiments on LibriSpeech, Conformer

"significantly outperforms" previous models, achieving ~2.1% WER on test-clean without an external language model[10] (and ~3.9% with one). Even a small Conformer variant (10M params) performs competitively[10]. Conformer has become a de facto backbone in many ASR systems (including some commercial products) due to this efficiency and accuracy.

Recent work has also fused ASR and diarization. **WhisperX** (Bain *et al.*, 2023) combines Whisper's transcription with pyannote's alignment to produce time-accurate transcripts[11]. In WhisperX, audio is first segmented (via a fast VAD+cut strategy) and fed to Whisper for content. Then forced phoneme alignment (guided by a neural alignment model) refines word-level timestamps. This yields transcripts with precise timing information. WhisperX achieves state-of-the-art results on benchmarks for long-form audio and word segmentation[11], and by batching segments it can run much faster. In effect, WhisperX provides an open-source blueprint: it shows how a robust transcription model and a diarization/aligner toolkit (like pyannote) can be combined for high-performance speech-to-text on long recordings.

2.1.4 Industry Standards vs. Custom Development

Industry leaders have quickly adopted these advances. For example, Deepgram's engineering blog emphasizes that older hybrid ASR (with separate acoustic/LFMs) is being replaced by fully end-to-end neural networks[12]. Deepgram notes that traditional pipelines make simplifying assumptions that limit performance, and argues that "deep learning is a better approach" for speech recognition[12]. Similarly, AssemblyAI (2022) has trained **Conformer-1** on **650,000 hours** of audio. Their results show this model is *far* more robust than off-the-shelf systems: on a noisy dataset spanning many domains, Conformer-1 makes 43% **fewer** errors on average than other leading models (including OpenAI's Whisper)[13]. This kind of scale (hundreds of thousands of hours) and architectural tuning is often only feasible for commercial teams with abundant resources.

Despite these successes, relying solely on closed APIs has its own drawbacks. Black-box ASR services may perform well generally, but they can struggle in niche conditions. In fact, open models allow domain-specific fine-tuning. For instance, when Whisper is combined with a specialized diarization/alignment pipeline, it can outperform some paid services on certain accents or long audios[13]. Additionally, researchers have shown that open ASR models can be customized to local conditions by using data augmentation techniques like noise injection or spectrum masking techniques like SpecAugment [14]. In actuality, a custom pipeline can be adjusted based on the speaker population, background noise profile, or target language/dialect, outperforming generic APIs in terms of performance and privacy.Moreover, on-premise solutions eliminate network latency and per-minute costs.

In summary, the literature indicates that modern diarization and ASR can approach or match commercial offerings. By using EEND-EDA and TS-VAD for flexible, overlap-aware diarization, and by employing large open ASR models like Whisper or Conformer, one can build a fully on-premises system with state-of-art accuracy. Frameworks like pyannote.audio and WhisperX[15]. provide practical implementations of these ideas. Thus, there is strong evidence that a carefully engineered custom pipeline can achieve "industry-grade" performance while ensuring data sovereignty and tunability.

2.2 Research Gap Identification

Based on the literature review, BubblesAI aims to address several important gaps:

2.2.1 No Real-Time Personal Coaching

Most existing tools, like Otter or Gong, only analyze a meeting after it has ended. There is no toolin the market that can work as a “Live Wingman,” while giving instant feedback on tone, word choices, or phrasing all while the conversation is still happening without interrupting the flow of the conversation.

2.2.2 Lack of Context Awareness in Personal Assistants:

Current assistants that are in the market such as Siri or Google Assistant or even chat GPT treat each interaction separately and forget previous conversations, as in, it would remember as long as you are in the session. They do not remember professional discussions or build long-term context and do not carry it to the other sessions. BubblesAI aims to use GraphRAG in addition with pgVectors to track relationships between people and topics across multiple conversations, providing more meaningful , and related information and help.

2.2.3 Privacy-Focused Personal Tool:

Most enterprise tools are built for companies,who leverages user data to make their products better, which can compromise individual privacy. There is a need for a lightweight, privacy-conscious app that focuses on personal growth, not organizational monitoring, and all the data control is with the user.

2.3 Requirements Elicitation

To find out how we can make a great assistant, we used three methods: looking at our own ideas, discussing it with friends as classmates (introspection), studying what’s already there in the market, and reviewing documents and publications. We checked top language learning apps like Duolingo and meeting assistants like Otter to figure out the important features and what can be improved upon. In the process the requirements that we gathered are listed below.

2.3.1 Functional Requirements

- FR01 Audio Recording: The app should be able to record audio and transmit it in real-time using the phone’s microphone.
- FR02 Real-Time Transcription: The app should be able to turn speech into text in less than a second.
- FR03 Speaker Diarization: The app should be able to tell the difference between the user and and other speakers(at least two distinct speakers should be identified).
- FR04 Tone Analysis: The app should check the user’s tone and wording to show if it is positive, neutral, or negative.
- FR05 Live Suggestions: In "Live-Wing" mode, the app should show keyword tips or better word suggestions while talking.
- FR06 Summarization: After a conversation, the app should make a short summary with the agenda, key decisions, and action items.
- FR07 Knowledge Retrieval: In "Consultant" mode, the user should be able to ask about past conversations (for example, “What did I tell Client X last week?”).
- FR08 Gamification: The app should give points for clear speech and using better vocabulary.

2.3.2 Non-functional Requirements

- NFR01-Latency: Suggestions should appear within 2 seconds of speaking.

- NFR01-Accuracy: The speech-to-text system should be correct at least 85% of the time, even in moderate noise.
- NFR01-Scalability: The system should handle many users at the same time without slowing down.
- NFR01-Privacy: User audio data should be safely encrypted both while sending and storing.
- NFR01-Usability: The app should be easy to use. Starting a recording should take no more than 3 taps.

2.3.3 Requirements Traceability Matrix

Req ID	Requirement Description	Module / Component	Priority	Test Case
FR-01	Real-time Audio Recording	Mobile Frontend (Flutter)	High	Audio Capture
FR-02	Real-time Transcription	Backend (Deepgram/ASR)	High	Live Text Output
FR-03	Speaker Diarization	AI Module (Deepgram)	Medium	Speaker split
FR-04	Tone/Sentiment Analysis	AI Module (LLM)	Medium	Sentiment detection
FR-07	Knowledge Retrieval (RAG)	Database (Neo4j)	High	RAG query test
NFR-01	Latency < 2 seconds	System Architecture	High	Latency check

Table 1: Requirements Traceability Matrix

2.4 Use Case Description

2.4.1 Use Case 1: Start Live-Wing Session

User Preconditions:

- The user is logged into the application.
- The device has an active internet connection.
- Microphone permissions are granted to the app.

Main Flow:

1. The User selects the "Live-Wing" mode from the dashboard.
2. The User chooses the conversation style (Formal, Semi-Formal, or Casual).
3. The System begins recording audio via the microphone.
4. The System sends audio chunks to the server for processing.
5. The System displays real-time transcription and suggestions on the screen.
6. The User reads the suggestions while talking.
7. The User ends the session when the conversation is over.

Postconditions:

- The audio is saved in the database.
- The transcript is generated and ready for review.

2.4.2 Use Case 2: Use Consultant Mode (Q&A)

User Preconditions:

- The user has at least one recorded conversation saved in history.
- The Knowledge Graph is updated with past context.

Main Flow:

1. The User selects "Consultant Mode" from the menu.

2. The User inputs a text or voice query (e.g., "What was the deadline for the project?").
3. The System uses the RAG module to search past conversations in the Knowledge Graph.
4. The System generates a context-aware answer using the LLM.
5. And then the System displays the answer to the User.

Postconditions:

- After all this, user receives appropriate response with all basic details.

2.4.3 Use Case 3: View Performance Dashboard

User Preconditions:

- The user has completed at least one conversation analysis.

Main Flow:

1. When the User navigates to the "Progress" tab.
2. And then the System retrieves user data, including frequent mistakes and vocabulary scores, tone scores, and what we have improved.
3. Then the System calculates the current rank and points based on recent improvements and performances.
4. Then the User can view graphs, points and improvements showing their communication growth over time.

Postconditions:

- The user is aware of their current skill level, frequent mistakes, inappropriate tones, and areas where he can improve improvement. He can also see his standings among other app user based on the points he has scored.

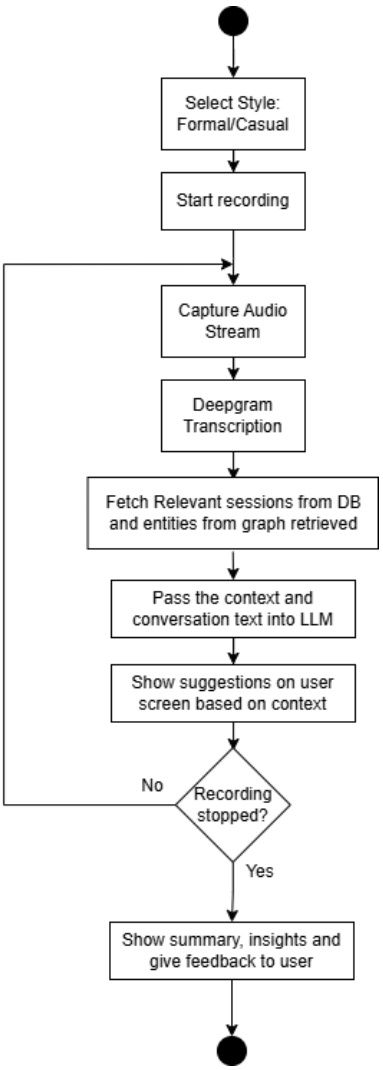
Chapter 3. System Design/Methodology

3.1 Activity Diagram

In the app we have three main Sections:

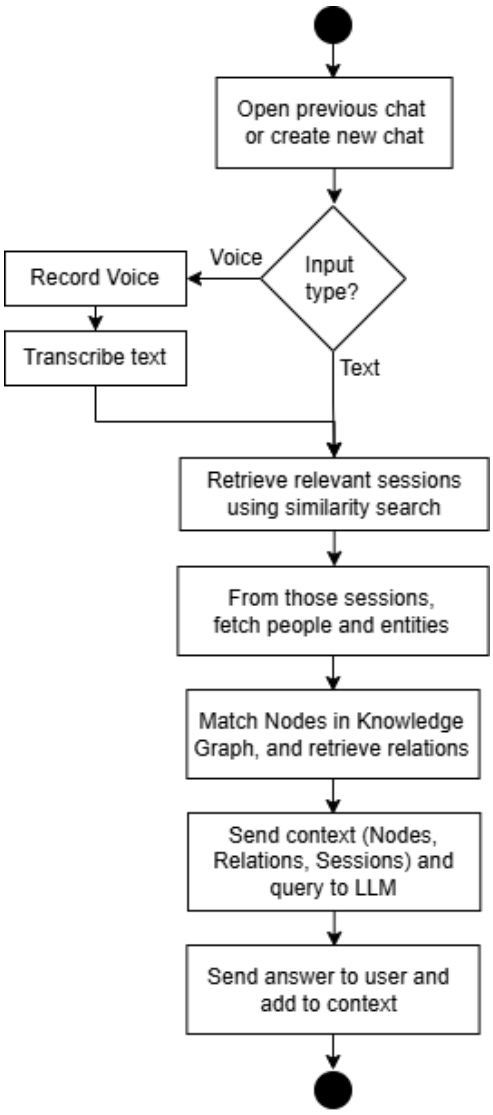
3.1.1 Live-Wing Mode

When the user chooses a conversation style (formal or casual) and begins recording, the process gets started. The system starts a continuous loop in which it records the audio stream and uses ASR to translate it into text. In order to create context, it simultaneously retrieves any relevant previous sessions from the database and associated entities from the Knowledge Graph. The Large Language Model (LLM) receives this contextual information and the current discussion in text. Throughout the meeting, the LLM continues to show important recommendations and comments on the user's screen. After ending the live session, the system creates a summary, offers insights, and sends opinions once the recording has stopped.



3.1.2 Consultant Mode

In this mode, the user opens a chat window and starts a new consultant session, then selects text or voice input. The device records the audio and converts it into text if voice is chosen. Once the text query has been submitted, the system uses a similarity search to find any relevant previous sessions for information. It extracts important people and entities from these sessions. To further add details and make meaningful connections, it then fetches the data from the Knowledge Graph. The user's question is sent to the LLM, along with all the collected information, including nodes, relations, and relevent previous session data. After that, the LLM creates a response, transmits it to the user, and records what happens in the context history.



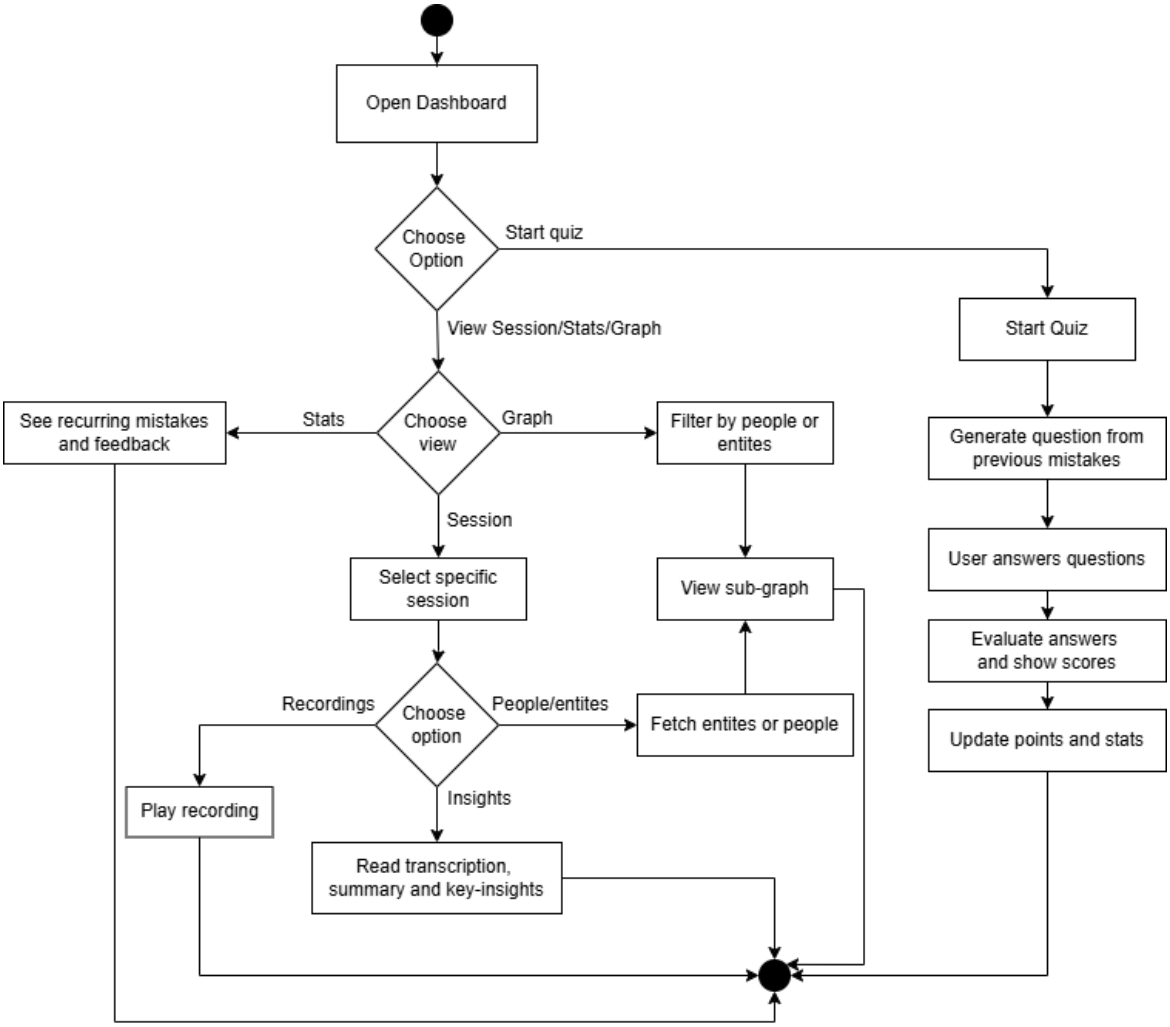
3.1.3 Progress & Gamification Mode

This section allows users to track their improvements, mistakes, and tone issues and practice their skills to overcome habitual mistakes. When a user opens the dashboard, they can choose between viewing their stats or identifying what the did wrong or what would have been the better choice.

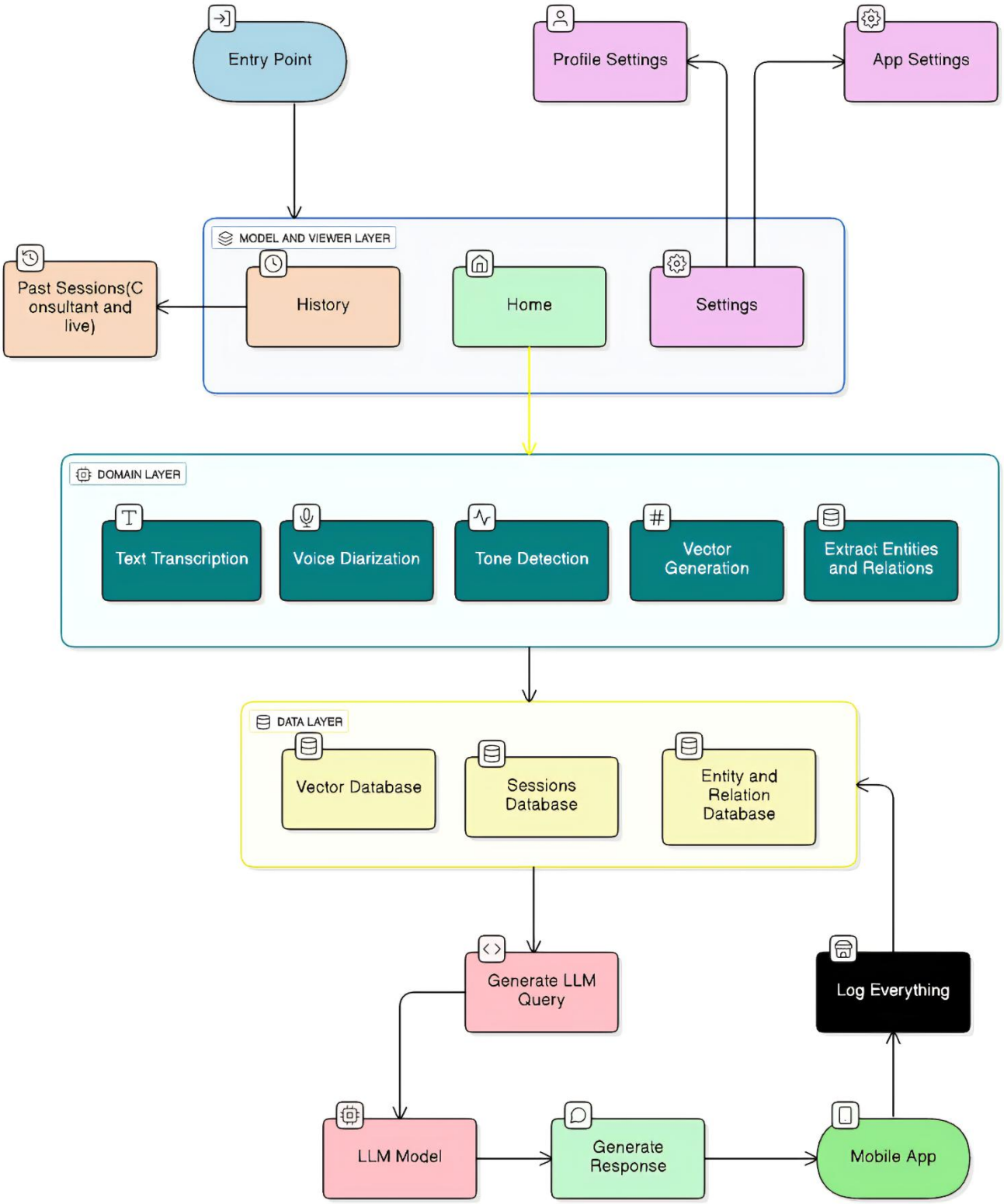
If they choose to view progress, they have three options:

- Check Level: The app displays their current rank and badge (like "Novice" or "Expert").
- Overall Stats: The system shows a list of mistakes they make most often across all their conversations.
- Session Stats: The user can select a specific past session history to see the feedback and suggestions for that particular session.

If the user chooses to opt for improvement, the system creates a personalized test based on their past errors and habitual mistakes. The user then answers the questions, and the system immediately grades them accordingly showing the score and the improvements they have made and updating their total points.



3.2 Software Architecture Diagram



The Bubbles App's layered architecture for processing user sessions (Live and Consultant) and producing insights utilizing an LLM is shown in this diagram.

The user interacts with the Model and Viewer Layer (e.g., Home, History) at the Entry Point, where the flow starts. The domain layer is where the majority of the processing takes place. It uses raw session data to carry out crucial activities, including text transcription, speaker identification using voice diarization, sentiment detection with tone detection, and the creation of semantically significant vector embeddings. Additionally, it generates structured data using relations and extracted entities.

The data layer, which includes an entity and relation database, a sessions database, and a vector database (for semantic search), stores all processed data. The LLM and output layer receive this data. Using the vector database for pertinent context, the system initially creates an efficient Generate LLM Query (Retrieval-Augmented Generation). This context is combined by the LLM model to provide a generated response. Ultimately, Log Everything records every step, and the user receives the result through the mobile app. This tiered strategy guarantees accurate, context-aware AI output, scalable data processing, and modularity.

3.3 Database Diagram (Optional)

3.4 Dataset Details

Unlike traditional machine learning projects that rely on static training datasets, this system utilizes a Dynamic Personal Knowledge Base that is generated in real-time from user interactions and pgVectors generated from past session transcripts The system relies on pre-trained state-of-the-art models such as Llama 3 for reasoning, Deepgram Nova-2 for transcription and constructs a dataset of user context on the fly.

3.4.1 *Data Source:*

The primary source of data is Real-time User Audio, captured via the LiveKit streaming interface and send to the backend. This unstructured audio takes two paths:

3.4.1.1 *Direct Interaction:* Audio input during "Consultant" or "Wingman" sessions, then converted into text to extract entities and relationships.

3.4.1.2 *Environmental Context:* Background conversations captured to build the user's "memory." To help user remember important details and conversation keypoints.

3.4.2 Data Creation & Processing Pipeline:

The data undergoes a multi-stage transformation process to become usable for the application, it goes through following processes:

Step 1: Transcription process: Raw audio is processed by Deepgram Nova-2. This step includes "Diarization" in which we identify who the main user is and who are the others and filtering out silence or non-speech noise.

Step 2: Vectorization: The clean text transcripts are passed through a Sentence Transformer model (all-MiniLM-L6-v2) to create 384-dimensional vector embeddings. These embeddings represent the semantic meaning of the conversation.

Step 3: Knowledge Extraction: The text is parallelly processed by Llama 3 (8B model) to extract structured knowledge (Entities and Relationships) into a JSON format (e.g., {source: "User", relation: "likes", target: "Coding"}).

3.4.3 *Data Structure & Fields:*

The system creates and maintains three distinct data structures in the Supabase database:

Vector Memory (memory table):

- content: The raw text chunk (e.g., "I have a meeting at 5 PM").
- embedding: A 384-float array used for semantic similarity search (RAG).
- created_at: Timestamps for temporal context.

Knowledge Graph (knowledge_graphs table):

- graph_data: A JSON structure representing a NetworkX graph.
- nodes: Unique entities (People, Places, Concepts).
- edges: Relationships linking nodes (e.g., "Friend", "Hosted By").

Session Logs (session_logs table):

- session_id: Unique identifier for the conversation.
- role: Speaker identity ("user", "agent", "other").
- content: The verbatim transcript.

3.4.4 Dataset Size*:*

The dataset is dynamic and cumulative. It starts empty as Size = 0; and grows linearly with each user interaction. For testing purposes, the system was validated using 30+ trial sessions, generating approximately 350+ lines of transcript and 45+ extracted graph relationships. And it was able to give exact details, provide help with complete context and provide the relevant information around 80% of the time.

3.5 Algorithm/Model Selection

The main goal for this project is to prioritizes latency so that we can get instant responses for real-time feedback and contextual depth for detailed consulting. To achieve this feat, a multi-model architecture is selected, utilizing specific State-of-the-Art models for Audio Diarization, Transcription, and response generation. Note that these modes are placeholders and will be replaced if we find better alternatives.

3.5.1 Automatic Speech Recognition (ASR):

Model Selected: nova-2 (Deepgram)

Justification: Nova-2 was chosen due to its industry-leading speed, which includes sub-300 ms latency in text transcription, high speaker identification accuracy, timestamp generation, and Voice Activity Detection.

Key Algorithm: In Speaker Diarization, the model uses clustering algorithms to separate the "User"—the app owner—from the "Others"—the other participants in the conversation. This is crucial because the "Wingman" feature needs to know who said what in order to generate appropriate responses.

3.5.2 Large Language Models (LLM):

Using two different versions of the Llama 3 architecture and the Groq Language Processing Unit engine for accelerated inference, the system performs a two-tiered Inference Strategy such that we have spped when needed and resoning when necessary.

3.5.2.1 Real-Time Tier (Wingman Mode & Information Extraction):

Model: llama-3.1-8b-instant

Few-Shot Learning algorithm. Because it can produce responses in as little as 50 milliseconds, this 8-billion parameter model is renowned for its speed. It extracts entities (Knowledge Graph Nodes) from transcripts in milliseconds and generates real-time advice.

3.5.2.2 Deep Reasoning Tier (Consultant Mode):

Model: llama-3.3-70b-versatile

Justification: The 70-billion parameter model offers superior reasoning capabilities. It is selected for the "Consultant" feature where latency is less critical than the accuracy, depth, and hallucination-resistance of the answer.

3.5.3 Semantic Search: Sentence Transformers

Model: all-MiniLM-L6-v2

Algorithm: Cosine Similarity.

Process: This model converts text transcripts into 384-dimensional dense vectors. When a query is made, the system calculates the Cosine Similarity between the query vector and stored memory vectors to retrieve the most semantically relevant past interactions (RAG - Retrieval Augmented Generation).

3.5.4 Contextual Retrieval Algorithm (The "Brain" Logic)

A custom Hybrid Retrieval Algorithm was implemented to construct the prompt context:

Vector Search: Retrieves unstructured text logs based on semantic similarity.

Graph Traversal (NetworkX): A Neighborhood Search Algorithm is applied to the Knowledge Graph.

Step 1: Key entities are identified in the user's query.

Step 2: The graph traversing algorithm finds the "Ego Graph" (immediate neighbors) around these entities to retrieve factual relationships (e.g., User -> knows -> Asma), covering gaps that vector focus (similarity) might miss.

Component	Selected Model	Purpose
Speech-to-Text	Deepgram Nova-2	Low-latency transcription & Speaker Diarization
Fast Inference	Llama 3.1 8B (Groq)	Real-time "Wingman" advice & entity extraction

Deep Reasoning	Llama 3.3 70B (Groq)	Complex "Consultant" Q&A
Embeddings	all-MiniLM-L6-v2	Vectorizing text for long-term memory
Retrieval	Hybrid (Cosine + Graph Traversal)	Combining fuzzy semantic history with hard facts

Table 2: summary for all the Models Used

Chapter 4. Implementation

4.1 Work Breakdown Structure (WBS)

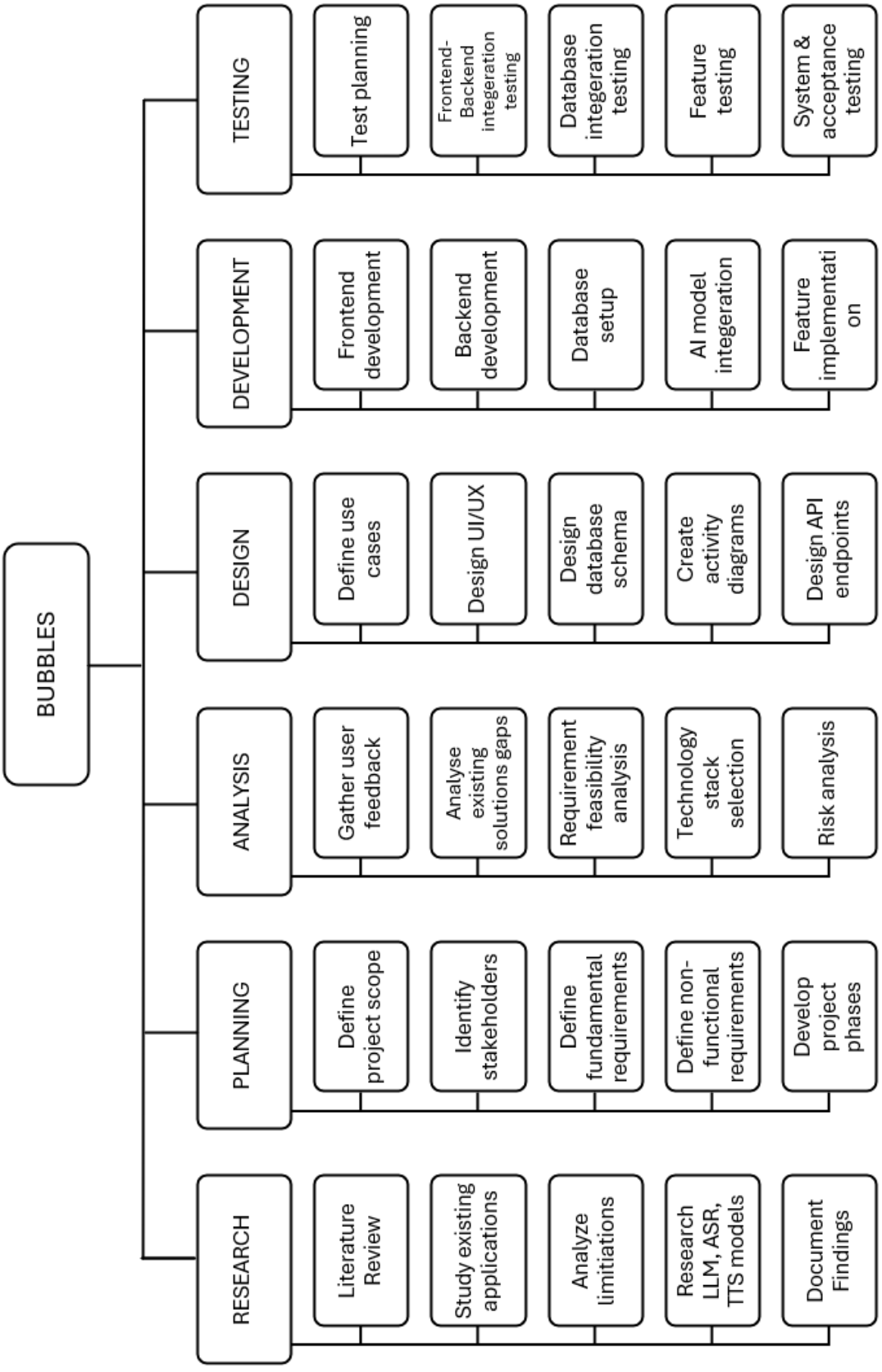


Figure 1: Work Breakdown Structure

4.2 Team Roles and Responsibilities

Team Member	Activity
	Phase 1: Architecture & Setup
M. Ahmad	Designing the high-level System Architecture Diagram (Client-Server-Database).
M. Ahmad	Selecting the Tech Stack (Flutter, Python FastAPI, Supabase) based on requirements.
M. Ahmad	Creating the GitHub Organization and Repository.
M. Ahmad	Setting up .gitignore and Branching rules (Main, Dev, Feature).
M. Ahmad	Configuring the Flutter project structure (Clean Architecture: Providers, Models, Screens).
M. Ahmad	Designing the Relational Database Schema (ERD) for Supabase.
M. Ahmad	Creating SQL tables: users, sessions, messages, memory_logs.
M. Ahmad	Configuring Row Level Security (RLS) policies for data isolation.
M. Ahmad	Setting up Supabase Storage buckets for User Avatars and Audio files.
	Phase 2: UI/UX & Design System
M. Ahmad	Researching modern mobile UI trends (Glassmorphism, Neomorphism).
M. Ahmad	Designing the "Bubbles" Color Palette (Primary, Secondary, Accent colors).
M. Ahmad	Selecting and integrating Google Fonts (Inter, Outfit) for typography.
M. Ahmad	Creating reusable AppButton and AppInput widgets with animations.
M. Ahmad	Designing the custom GlassContainer widget for the frosted glass effect.
M. Ahmad	Implementing Lottie Animations for loading states and empty data states.
M. Ahmad	Designing the application Logo and App Icon variants (Adaptive Icons).
M. Ahmad	Creating the Splash Screen with animated logo reveal.
M. Ahmad	Implementing Dark Mode & Light Mode Theme logic.
	Phase 3: Frontend Feature Development
M. Ahmad	Implementing the AuthProvider for managing User Sessions state.
M. Ahmad	Developing the Login Screen with Form Validation (Email/Password RegEx).
M. Ahmad	building the Signup Screen with secure Password Visibility toggles.
M. Ahmad	Implementing the "Verify Email" OTP screen logic.
M. Ahmad	Creating the App Drawer (Sidebar) with navigation routing.
M. Ahmad	Developing the Main Dashboard Grid layout for features.
M. Ahmad	Building the "Live Wingman" screen with microphone controls.
M. Ahmad	Creating the "Chat Interface" with dynamic bubble message rendering.
M. Ahmad	Implementing Auto-Scroll logic for the chat window.
M. Ahmad	Developing the "History" list view with pagination support.
M. Ahmad	Building the "Settings" screen for User Profile management.
M. Ahmad	implementing "Error Toasts/Snackbars" for network failure feedback.
	Phase 4: Integration & Logic (Upcoming/Ongoing)
M. Ahmad	Integrating supabase_flutter SDK for Auth and Database IO.
M. Ahmad	Writing ApiService class for typed HTTP requests/responses.
M. Ahmad	Integrating LiveKit Client for WebRTC Audio Streaming.
M. Ahmad	Implementing Background Audio permissions for Android/iOS.
M. Ahmad	Handling WebSocket Data Channels for receiving real-time advice.
M. Ahmad	Implementing Local Storage (shared_preferences) for caching settings.
M. Ahmad	Optimizing images using cached_network_image provider.
	Phase 5: Release Engineering (Upcoming)
M. Ahmad	Configuring AndroidManifest.xml permissions (Mic, Internet, Foreground Service).
M. Ahmad	Generating KeyStore for Android App Signing.
M. Ahmad	Optimizing App Bundle size (ProGuard/R8 obfuscation).

M. Ahmad	Creating Screenshots and Graphic Assets for the Play Store listing.
M. Ahmad	Recording and Editing the Final Project Demo Video.
	Phase 1: AI Research & Backend Environment
Attique Rehman	Researching SOTA LLM models (Llama 3, Mistral, Gemma) for reasoning.
Attique Rehman	Evaluating ASR options (Deepgram, Whisper, Google) for speed vs cost.
Attique Rehman	Installing Python 3.9+ and setting up Virtual Environment (venv).
Attique Rehman	Managing dependencies via requirements.txt,
Attique Rehman	Setting up the FastAPI framework structure (main.py, routers/).
Attique Rehman	Configuring uvicorn server for asynchronous request handling.
Attique Rehman	Implementing CORS Middleware for secure API access.
Attique Rehman	Setting up python-dotenv for secure Environment Variable management.
Attique Rehman	Creating Pydantic Models for Request/Response Schema validation.
	Phase 2: Core Intelligence Pipeline
Attique Rehman	Developing the Token Generation endpoint (/getToken) for LiveKit.
Attique Rehman	Integrating Deepgram Python SDK for streaming transcription.
Attique Rehman	Implementing "Speaker Diarization" identifying 'User' vs 'Speaker B'.
Attique Rehman	Integrating Groq Client for ultra-fast Llama 3 inference.
Attique Rehman	Designing the "System Prompt" strategy for the AI Persona.
Attique Rehman	Implementing Context Window management (truncation logic).
Attique Rehman	Handling LLM Rate Limits and Retry logic.
Attique Rehman	Developing the "Wingman" logic for unsolicited advice generation.
	Phase 3: Advanced Memory Systems
Attique Rehman	Integrating sentence-transformers(all-MiniLM-L6-v2) library.
Attique Rehman	Writing logic to chunk text transcripts for embedding.
Attique Rehman	Implementing Vector Database insertion logic for Supabase via pgvector.
Attique Rehman	Developing Cosine Similarity Search algorithms for Memory Retrieval.
Attique Rehman	Integrating NetworkX library for Graph Data Structures.
Attique Rehman	Developing Entity Extraction prompts ("Extract Person, Place, Thing").
Attique Rehman	Implementing Graph Traversal algorithms (finding neighbor nodes).
Attique Rehman	Writing the "Consultant" endpoint combining History + Graph + Vectors.
	Phase 4: Optimization & Refinement (Upcoming)
Attique Rehman	Implementing "Graph-RAG" logic to merge vector/graph contexts.
Attique Rehman	Optimizing Server-side threading for parallel processing.
Attique Rehman	Creating Background Workers for non-blocking database writes.
Attique Rehman	Writing automated scripts to clean up old session logs.
Attique Rehman	Dockerizing the Backend Application for containerized deployment.
Attique Rehman	Hosting the Backend on a Cloud GPU instance (e.g., RunPod/AWS).
	Phase 5: Validation & Thesis (Upcoming)
Attique Rehman	Conducting Load Testing (Locust.io) for concurrent users.
Attique Rehman	Measuring "Glass-to-Glass" latency via high-speed logs.
Attique Rehman	Calculating "Hallucination Rate" metrics for the Thesis.
Attique Rehman	Writing the "Methodology" chapter: "Hybrid Memory Architecture".
Attique Rehman	Documenting API endpoints using Swagger UI / OpenAPI keys.

Table 3: Activity Details

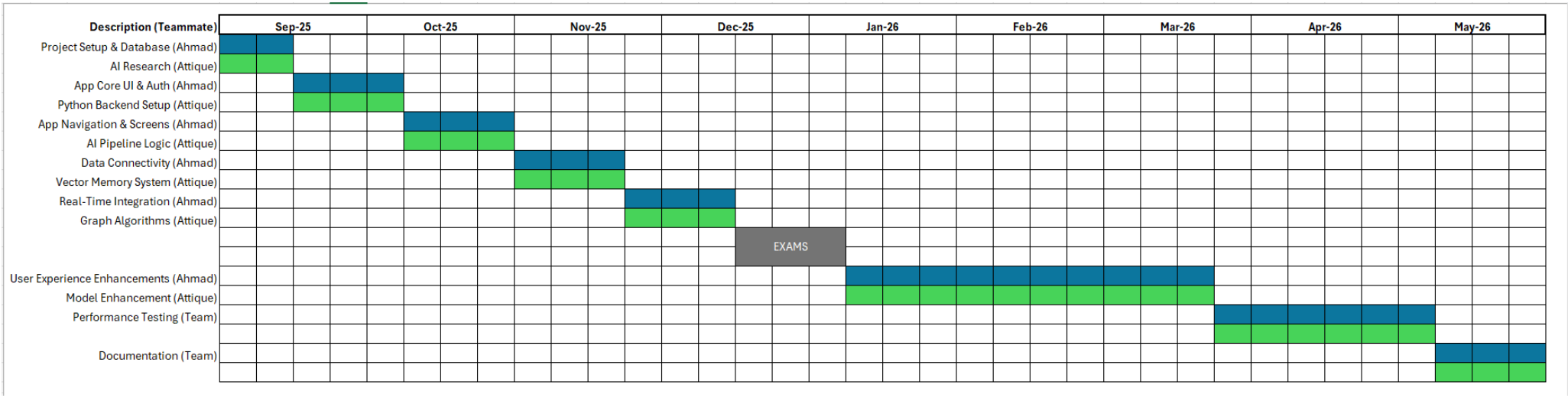


Figure 2: Gantt Chart

4.3 Tools and Technologies

The project was developed using a split-stack architecture, utilizing a cross-platform mobile frontend and a high-performance Python backend for AI processing.

4.3.1 Frontend Development (Mobile Application)

4.3.1.1 *Framework: Flutter (SDK 3.10+)* - Chosen for its single codebase capability for both Android and iOS.

4.3.1.2 *Language: Dart* - The programming language used for the Flutter framework.

4.3.1.3 Key Libraries:

- livekit_client: For handling real-time WebRTC audio streaming.
- provider: For state management across the application.
- supabase_flutter: For managing user authentication and database realtime connections.
- flutter_animate: For UI animations and transitions.
- flutter_markdown: To render rich text responses from the AI.

4.3.2 Backend Development (AI Server)

4.3.2.1 *Language: Python 3.9+* - Selected for its robust support for AI/ML libraries.

4.3.2.2 *Framework: FastAPI* - A modern, high-performance web framework for building APIs with Python 3.6+.

4.3.2.3 *Server Implementation:* uvicorn (ASGI server) running asynchronously to handle multiple concurrent audio streams.

4.3.2.4 *Graph Processing: NetworkX* - Used to build and traverse the in-memory personal Knowledge Graph.

4.3.2.5 *Tunneling: Ngrok* - Used during R&D to expose the local Python backend to the internet for the mobile app to connect.

4.3.3 Artificial Intelligence & Machine Learning

4.3.3.1 *Inference Engine: Groq* - Used as the LPU (Language Processing Unit) to run Llama 3 models with ultra-low latency.

4.3.3.2 *Large Language Model: Llama 3 (via Groq API)* - Used for intelligence, reasoning, and text generation.

4.3.3.3 *Speech-to-Text: Deepgram SDK* - Used for real-time transcription and speaker diarization.

4.3.3.4 *Vector Embeddings: Sentence Transformers (all-MiniLM-L6-v2)* - Python library used to convert text into vector embeddings for semantic search.

4.3.4 Database & Cloud Infrastructure

4.3.4.1 *Primary Database: Supabase (PostgreSQL)* - Used for storing user profiles, structured logs, and session data.

4.3.4.2 *Vector Database: pgvector (on Supabase)* - Used to store high-dimensional vector embeddings for the AI's long-term memory.

4.3.4.3 Real-time Infrastructure: LiveKit Cloud - A managed WebRTC infrastructure used to transport low-latency audio between the mobile app and the Python server.

4.3.5 Development Environment

4.3.5.1 *IDE*: Visual Studio Code.

4.3.5.2 *API Testing:* Postman (for REST API) and LiveKit Connection Tester (for WebRTC).

4.3.5.3 Version Control: Git & GitHub.

4.4 Implementation Details

The implementation of the system is divided into three core modules: the Frontend Application (User Interface), the Backend Brain (Intelligence core), and the Real-Time Communication Layer.

4.4.1 1. System Architecture Overview

With mobile devices with the app serving as input and output terminals and the Python server serving as the central "Brain," for our system the system uses a hub-and-spoke architecture.

4.4.1.1 *Audio Pipeline:* The mobile application records unprocessed audio from mobile's microphone and then transmits it to the server over WebRTC using LiveKit.

4.4.1.2 *Cognitive Pipeline*: In order to produce a accurate and useful response, the server processes the audio, retrieves context (Memory + Graph), and requests the LLM (Groq) to generate an appropriate response

4.4.1.3 *Feedback Loop*: The app receives the response as either text or virtual speech and then diasplays it for the user to see.

4.4.2 Backend "Brain" Implementation (Server-Side)

The backend logic that is currently implemented in server.py is implemented using Python and FastAPI and is structured in four distinct services that run concurrently to provide a seamless experience and are:

4.4.2.1 The Graph Service (Structure Knowledge):

Logic: The NetworkX library is used to implement it, so we can retrieve relations and entities instantly.

Function: It keeps track of each active user's dynamicly in-memory Knowledge Graph. The Llama 3 model is used by the system to extract entities such as "Alice" and "Project X" when a user talks. These entities are included in the graph as nodes. In order to give the AI factual context, the system then searches the graph for direct relationships e.g., Who is Alice? -> Alice is Project Manager. So as a result the response include hard facts and figures.

4.4.2.2 The Vector Service (Long-Term Memory):

Logic: SentenceTransformer and Supabase pgVectors are used to implement this.

Function: A 384-dimensional vector is "embedded" in each chat log. In order to enable the app and LLM "remember" historical details, the system runs a Cosine Similarity Search against the Supabase database whenever a new query is received.

4.4.2.3 The Brain Service (Reasoning Core):

Logic: It creates a "System Prompt" for the LLM on the fly. This prompt is not static, before being transmitted to LLM, it is included with Graph Facts and Vector Memories. This guarantees that the model responds using the user's unique information rather than merely generic training data.

4.4.3 Real-Time "Wingman" Pipeline Implementation

The core feature of the project, the "Wingman," requires a specialized non-blocking pipeline to achieve real-time performance and we achieved it by following:

4.4.3.1 *Ingestion:* The server listens to the LiveKit room to get audio and then we further process that audio.

4.4.3.2 *Diarization:* Using the Deepgram API, the incoming audio stream is split by speaker. The system isolates the voice of the "interlocutor" the person talking to the user and the user.

4.4.3.3 *Parallel Processing*: The transcript of the interlocutor is processed in parallel:

Process A: Sent to specific vector storage to update context or to get the context theat is required to generate LLM response.

Process B: Sent to the 8B Parameter "Fast LLM" to check if advice is needed and if yes then send this back to mobile phone

4.4.3.4 *Instant Advice*: once LLM determines advice is valuable, it is pushed directly to the user's screen via a WebSocket data channel, bypassing standard HTTP request/response delays. It not only delivers advices, but also queries and tips to enrich the users conversation with solid fact and details.

4.4.4 Frontend Integration (Flutter)

The mobile application is designed as a "Thin Client," keeping heavy processing off the device to save battery.

4.4.4.1 *State Management:* The app uses the Provider pattern to manage the connection state (Connecting, Live, Error).

4.4.4.2 *Live WebRTC*: The LiveKit widget handles the complexity of audio encoding and network jitter buffering, ensuring clear audio transmission even on unstable mobile networks.

4.4.4.3 *Secure Auth*: Integration with Supabase Auth ensures that users can only access their own private Knowledge Graphs and Memory Banks on the server.

Please Note that these tools and technologies can be changed as per the technology changes or a new alogoritham or better model or a technique is introduced.

4.5 Screenshots of Prototype

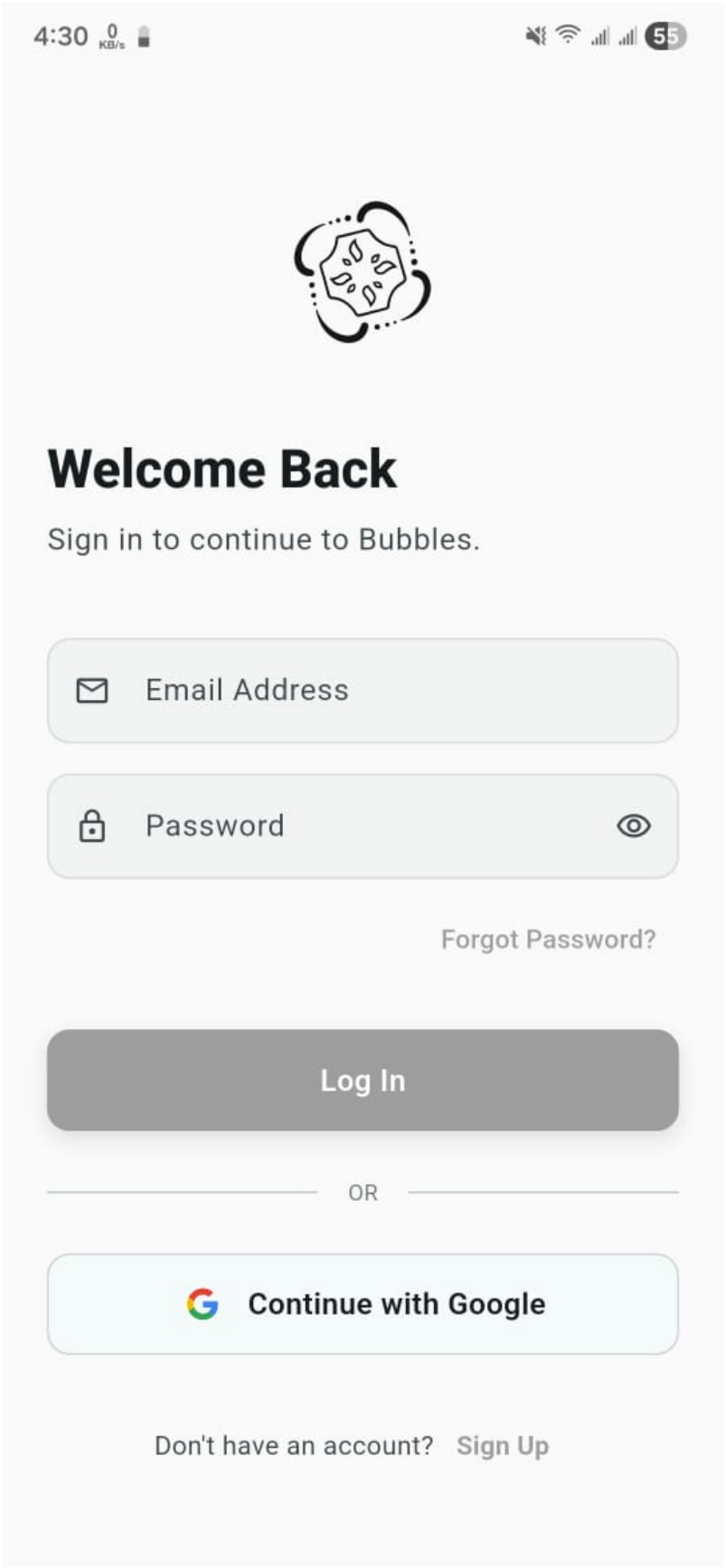


Figure 3: Login Screen

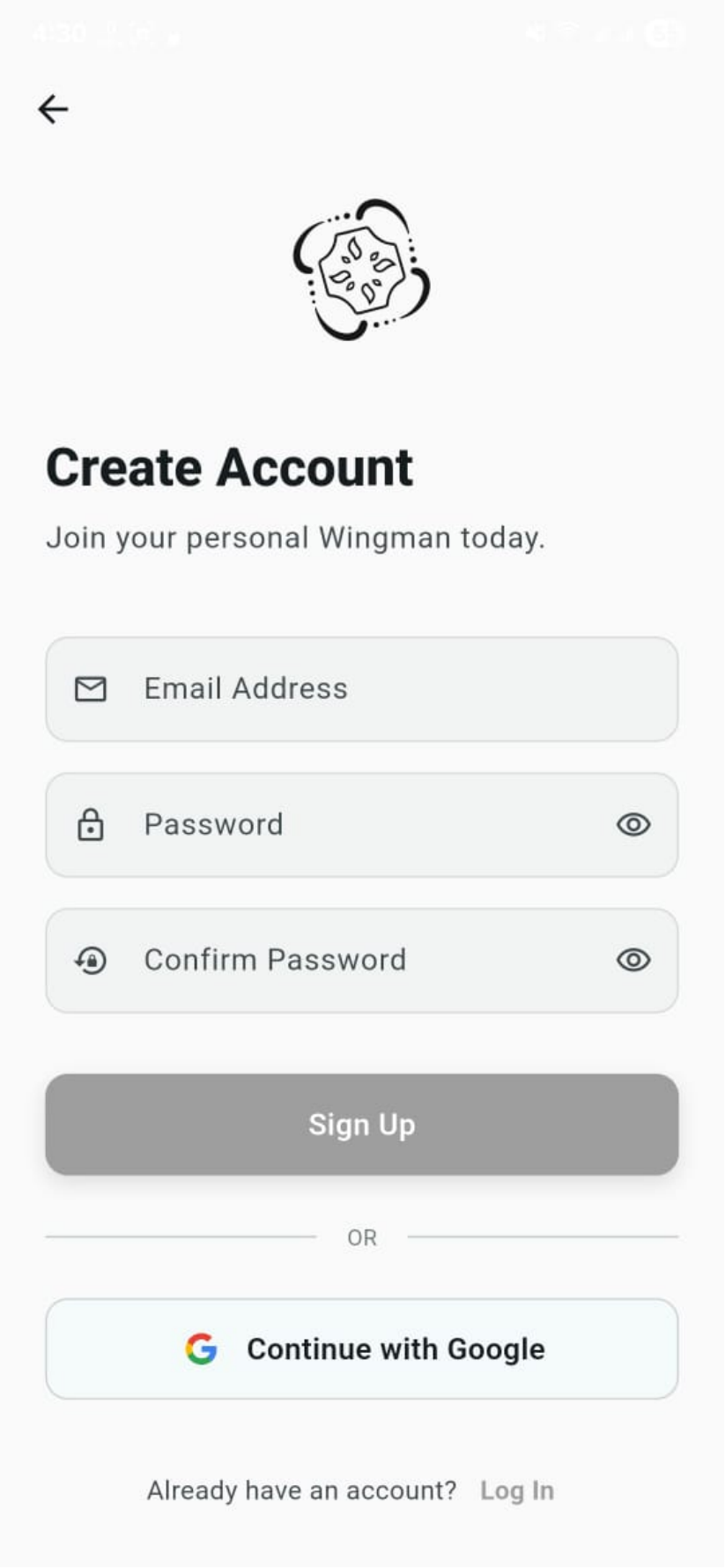


Figure 4: Signup Screen

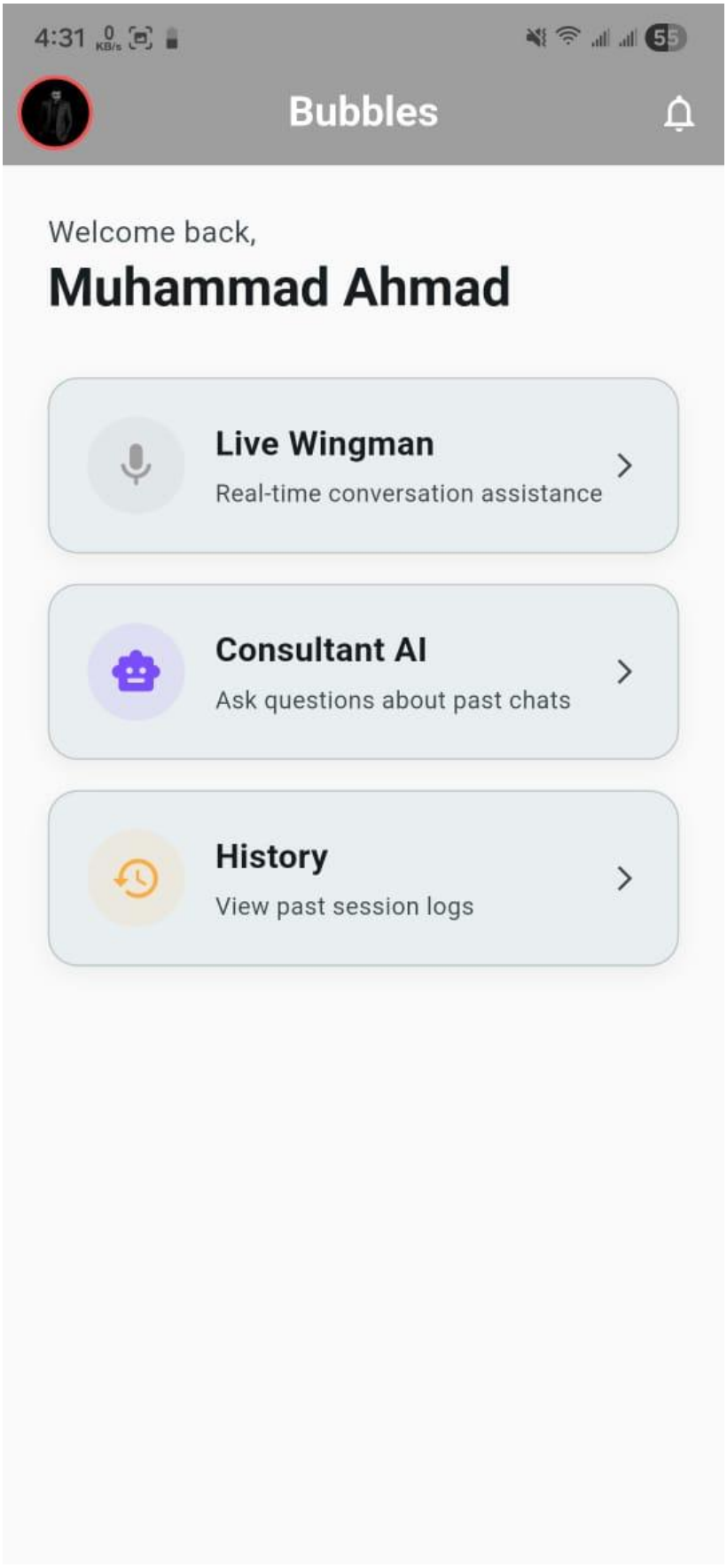


Figure 5: Home Screen (When Server Not Connected)

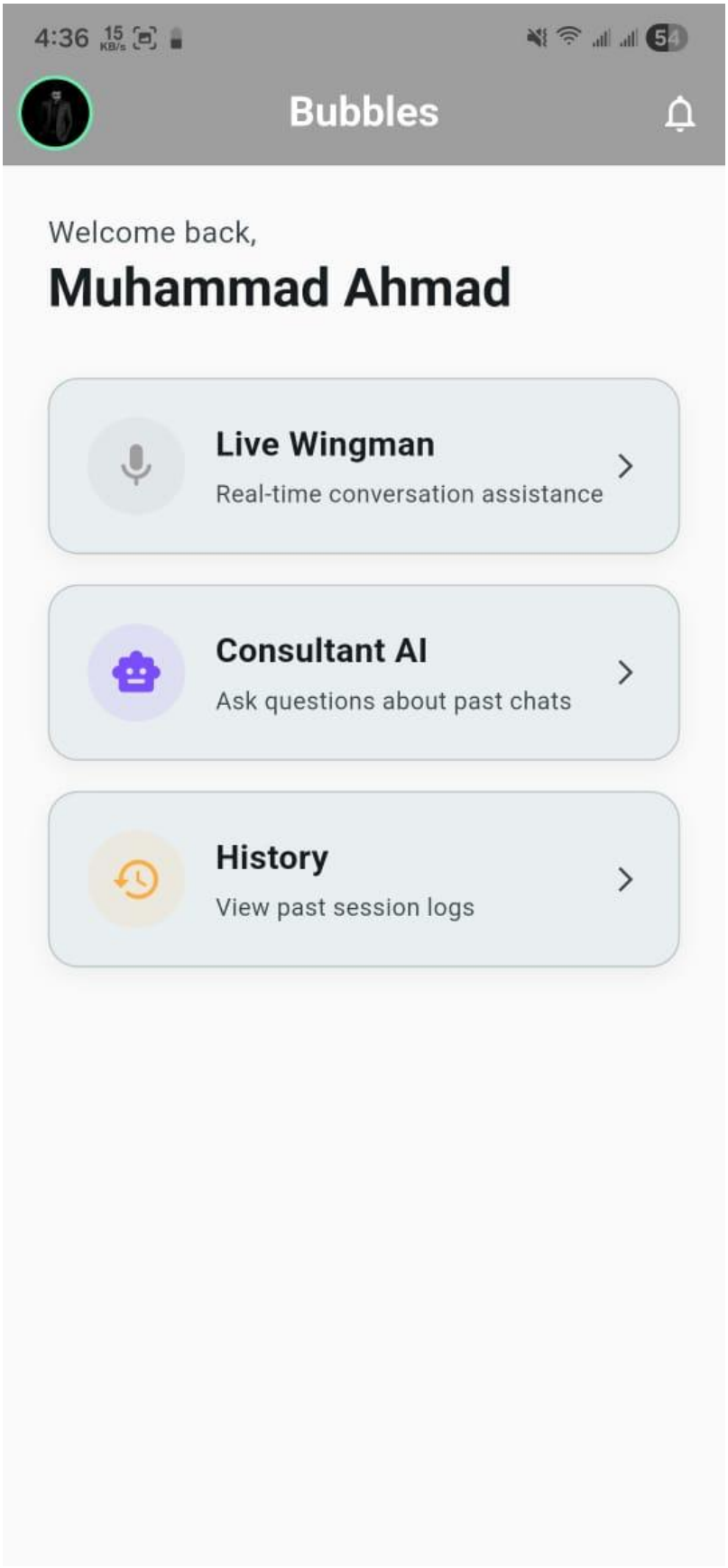


Figure 6: Home Screen (When Server Connected)

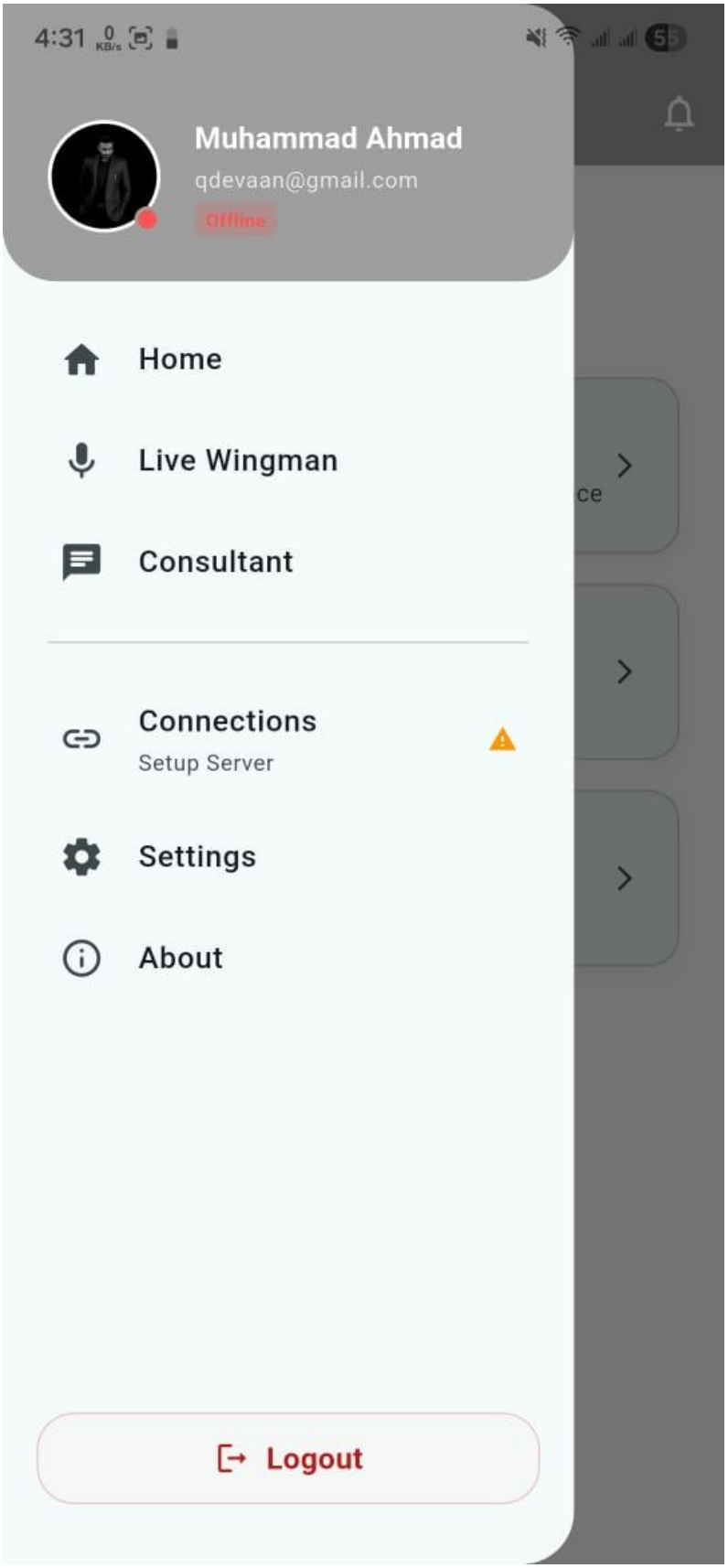


Figure 7: App Drawer (when Server not Connected)

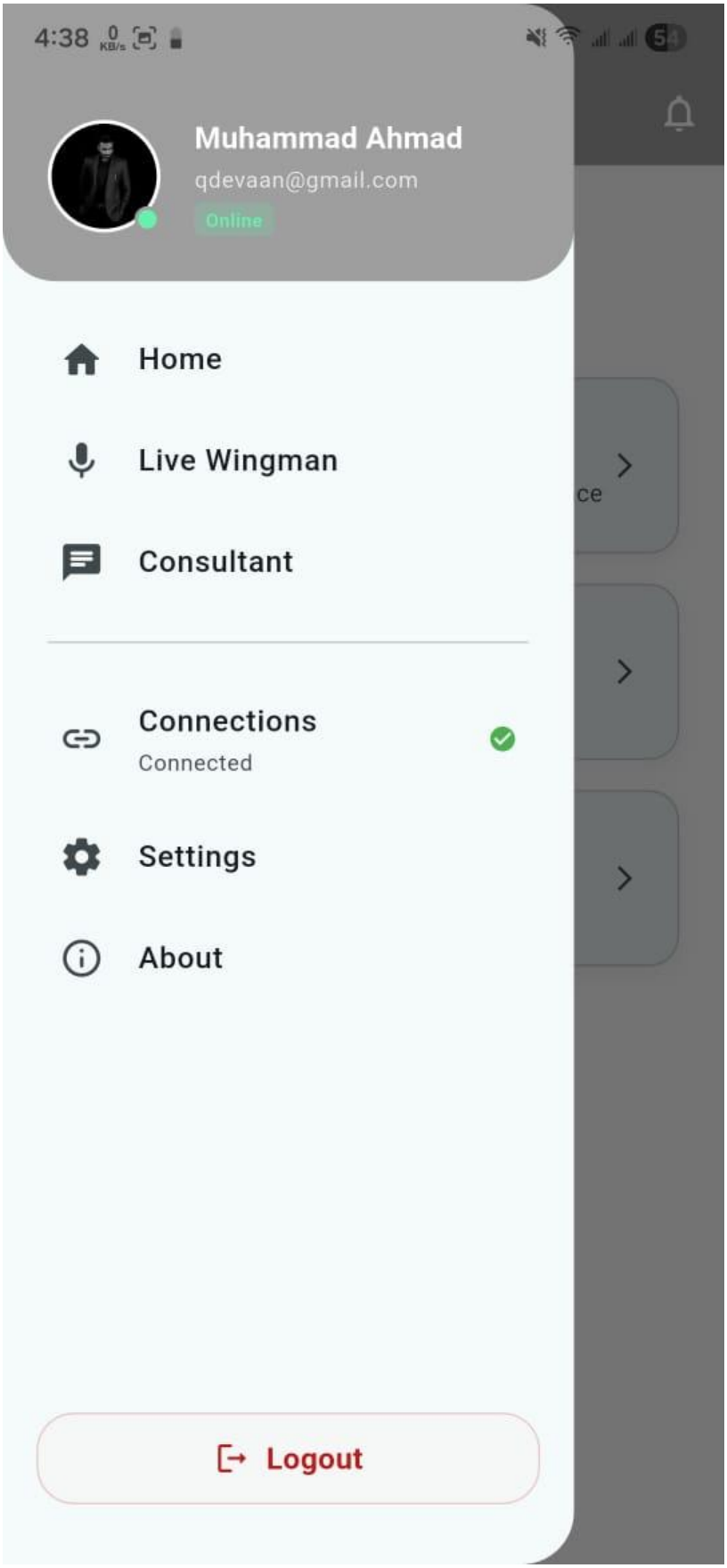


Figure 8: App Drawer(when Server Connected)

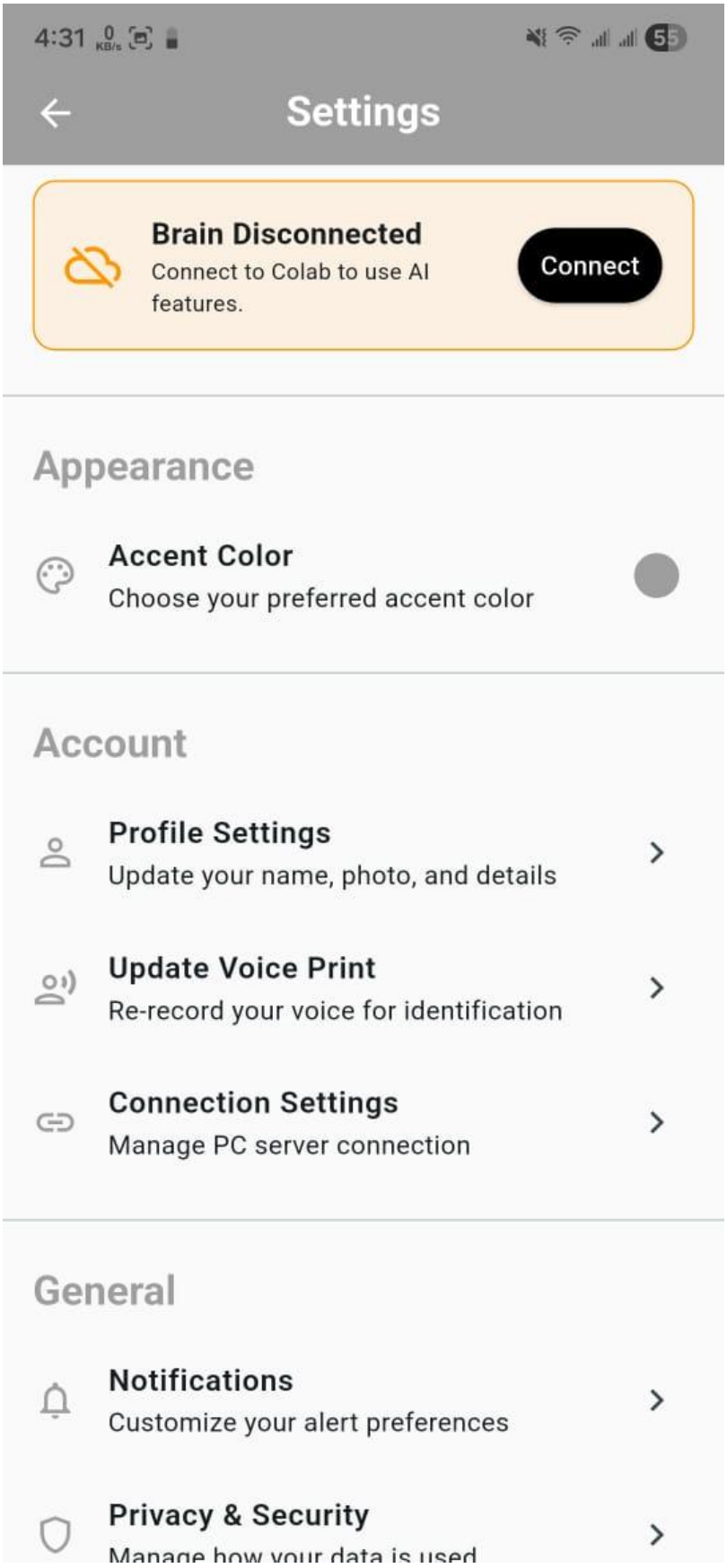


Figure 9: Settings Screen

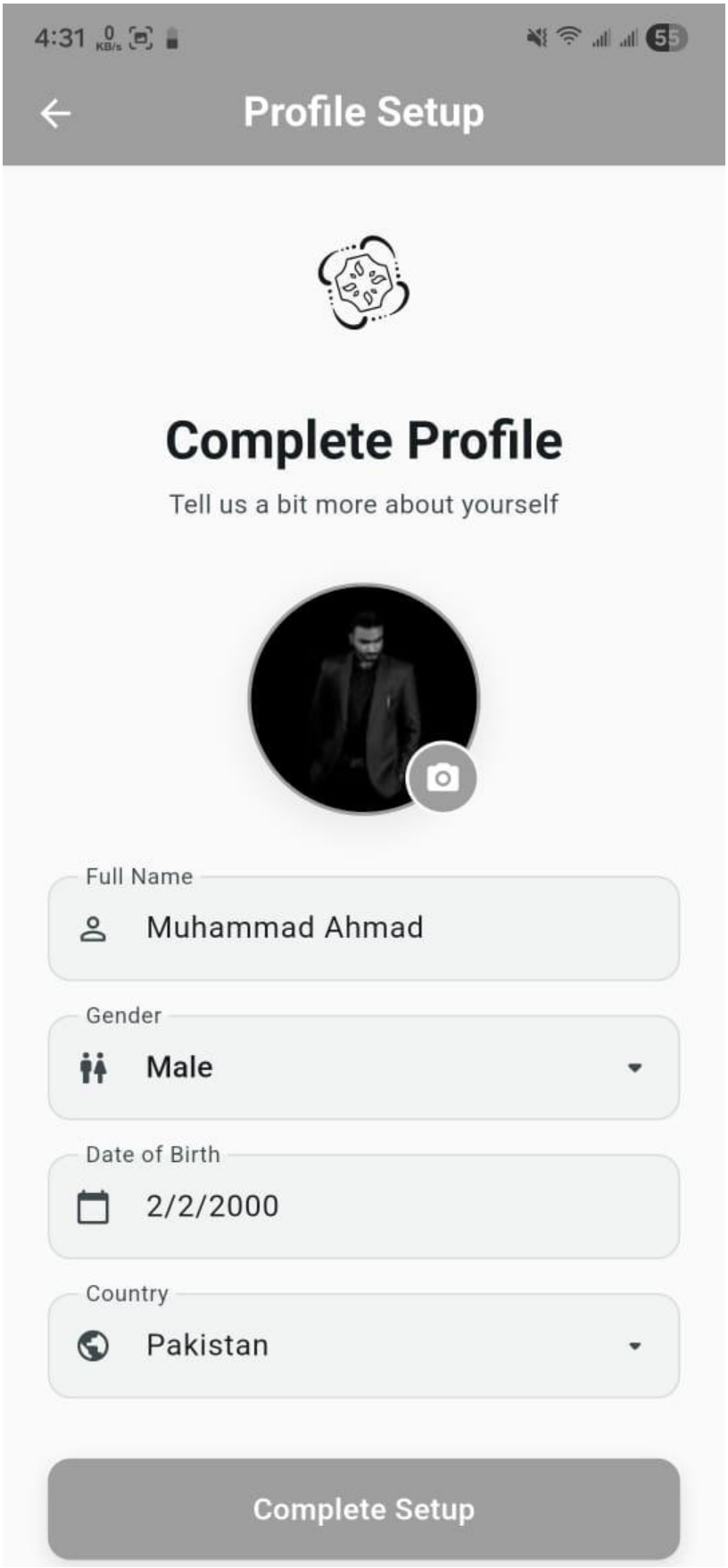


Figure 10: Profile Setup Screen

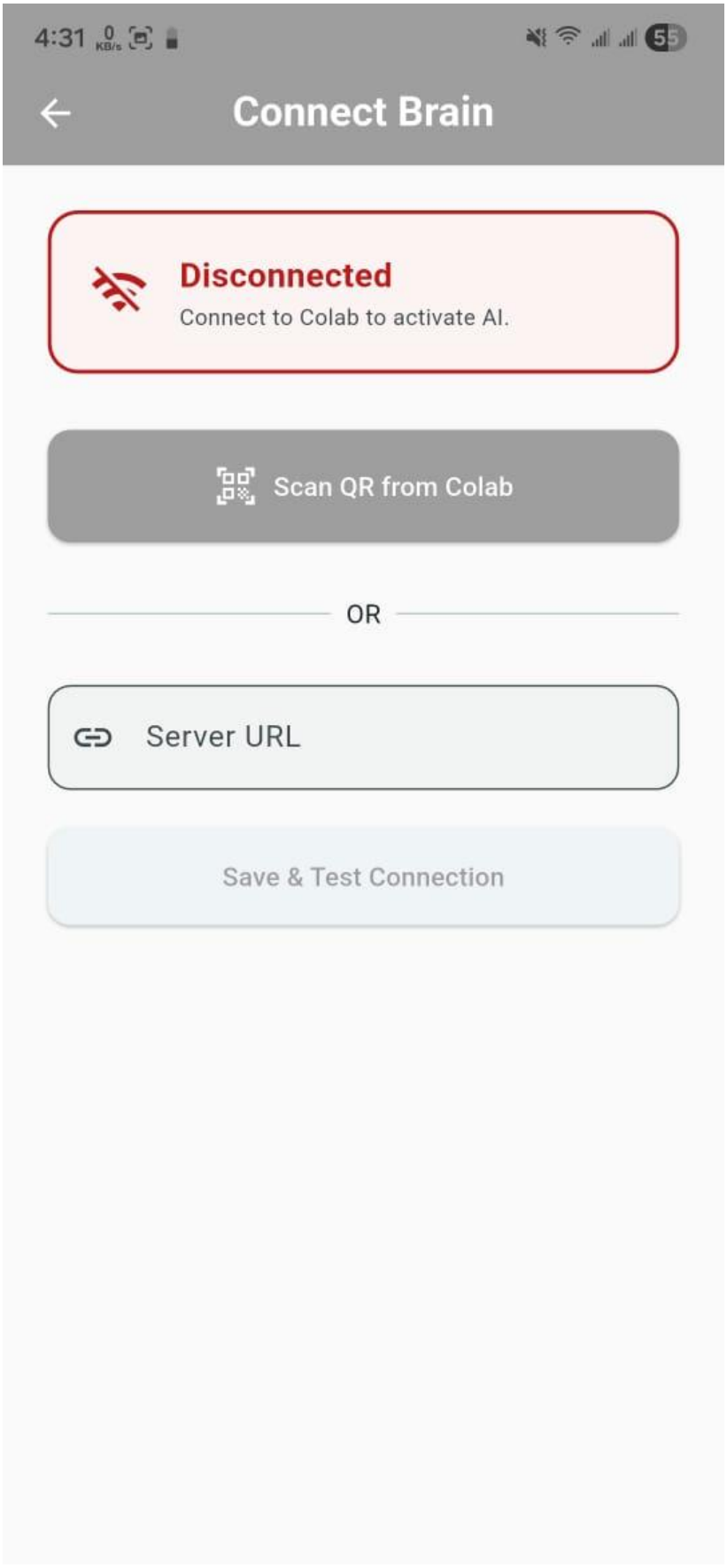


Figure 11: Connection Screen (when server not Connected)

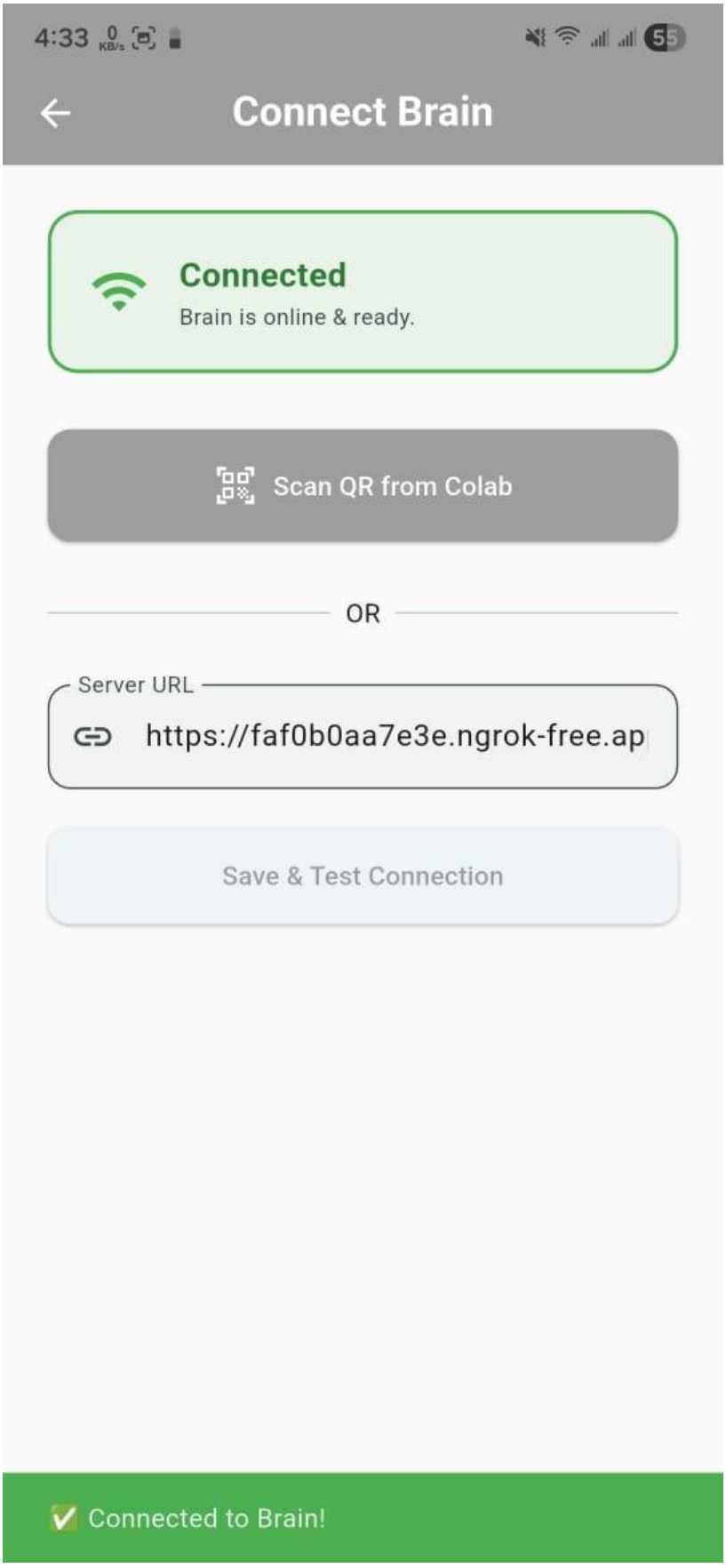


Figure 12: Connection Screen (when server Connected)

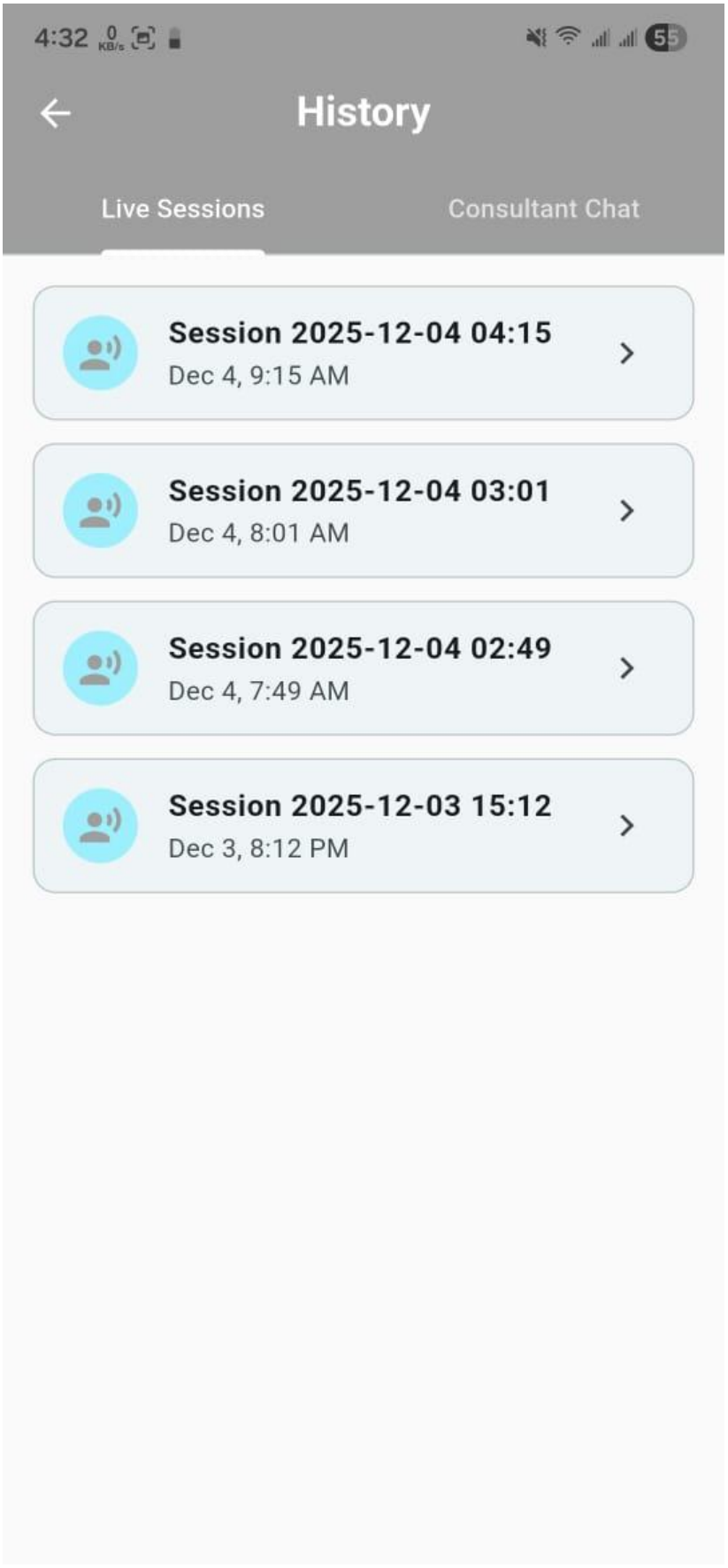


Figure 13: History Screen (Live Session)

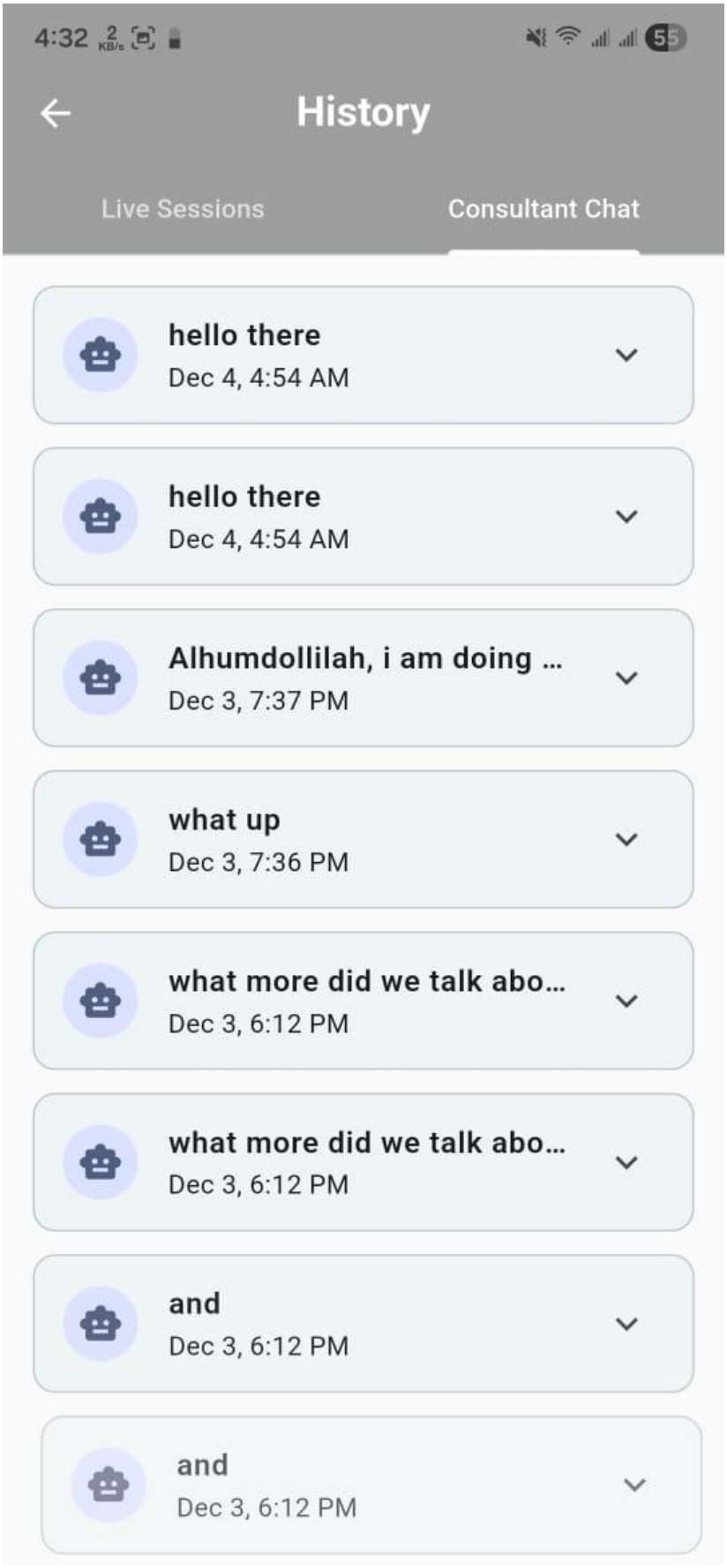


Figure 14: History Screen (Consultant mode)

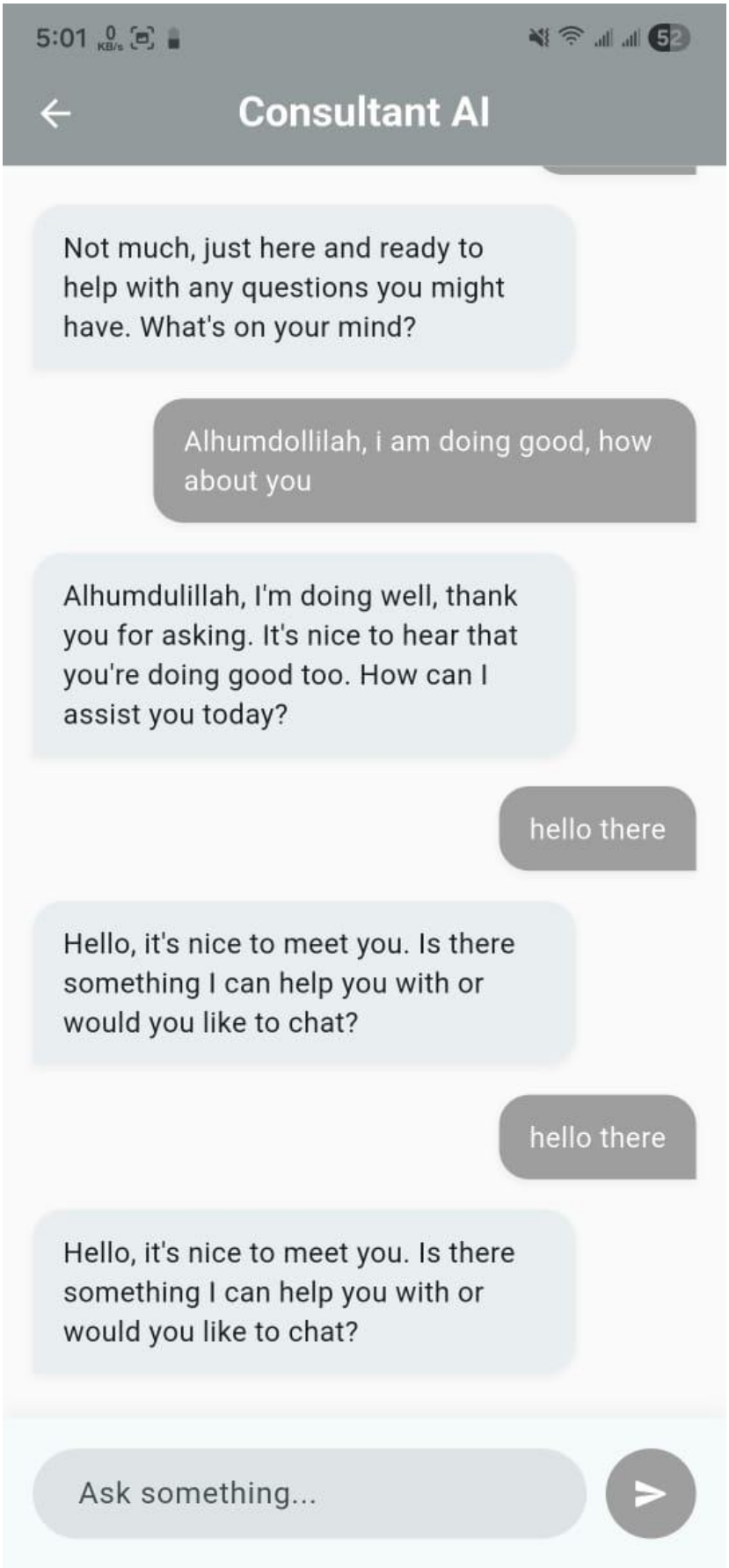


Figure 15: Consultant Screen

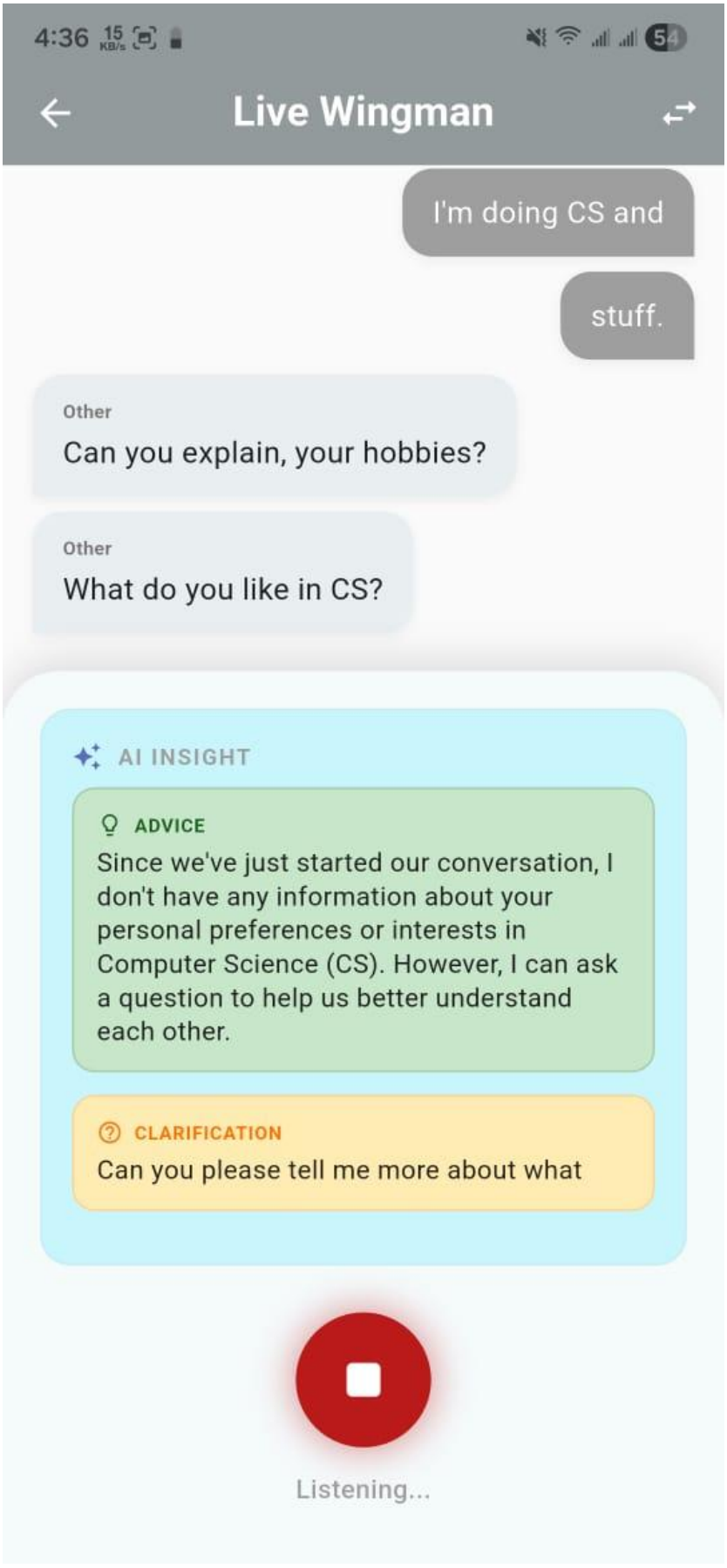


Figure 16: Live Wingman Screen

4.6 Challenges During Implementation

Developing a real-time AI application involving audio streaming and complex reasoning presented several significant technical hurdles:

4.6.1 Latency Management in Real-Time Advice

4.6.1.1 The Problem: The "Wingman" feature requires providing advice while the conversation is happening. Initially, chaining the processes (Speech-to-Text → Vector Search → LLM Generation → Response) caused delays of 3-5 seconds, which was too slow for natural conversation.

4.6.1.2 The Solution: Implemented a Parallel Execution Pipeline. The system was re-architected to run the "Knowledge Extraction" and "Memory Saving" tasks in the background (fire-and-forget), while only the critical "Advice Generation" path was kept on the main thread. Additionally, switching to the faster Llama 3 8B model on Groq's LPU hardware reduced inference time from ~2s to ~300ms.

*4.6.2 Synchronizing Audio Streams & Speak*er Identification

4.6.2.1 The Problem: Differentiating between the "User" and "Other" speakers in a single audio stream was difficult. Standard transcription often merged sentences from both speakers, confusing the AI about who said what.

4.6.2.2 The Solution: Integrated Deepgram's Speaker Diarization. We had to implement logic to parse the word-level timestamps returned by the API to reliably attribute segments to the correct speaker, ensuring the Wingman only advises on what the other person said, not what the user themselves just said.

4.6.3 Hallucinations in the Consultant Node

4.6.3.1 The Problem: The LLM would occasionally invent facts when it didn't have enough context, or it would confuse similar past events.

4.6.3.2 The Solution: Adopted a Graph-Augmented Retrieval (GraphRAG) approach. Instead of relying solely on vector similarity (which is "fuzzy"), we forced the model to look up hard facts in the Knowledge Graph first. We also improved the "System Prompt" to explicitly instruct the model: "If you do not find the answer in the provided Context context, state that you do not know."

4.6.4 Managing State in a Stateless Environment

4.6.4.1 The Problem: HTTP servers (FastAPI) are inherently stateless, but a "Conversation" requires continuous state (graph nodes, recent message history). Losing state meant the AI "forgot" what it said 10 seconds ago.

4.6.4.2 The Solution: Implemented a Session Manager in the server's memory (LIVE_SESSIONS dictionary) and backed it up to Supabase. This allowed the

server to hold active conversation context in RAM for speed, while asynchronously persisting it to the database for reliability.

4.6.5 *Mobile* Connectivity & Audio Quality

4.6.5.1 *The Problem:* Streaming high-quality audio from a mobile device is unstable on cellular networks. Packet loss resulted in "choppy" audio that the AI could not transcribe.

4.6.5.2 *The Solution:* Utilized LiveKit's Adaptive Stream capability. We configured the client to automatically adjust bitrate based on network conditions and implemented a local buffer to ensure smooth transmission even during momentary connectivity drops.

Chapter 5. References

[1] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, *"X-vectors: Robust DNN Embeddings for Speaker Recognition,"* in **Proc. IEEE ICASSP**, 2018.

[2] D. Snyder, D. Garcia-Romero, G. Sell, A. McCree, D. Povey, and S. Khudanpur, *"Speaker Recognition for Multi-Speaker Conversations Using X-Vectors,"* in **Proc. IEEE ICASSP**, 2019.

[3] F. Landini, J. Profant, M. Diez, and L. Burget, *"Bayesian HMM clustering of x-vector sequences (VBx) in speaker diarization: theory, implementation and analysis on standard tasks,"* **Computer Speech & Language**, vol. 71, 2022.

[4] Y. Fujita, N. Kanda, S. Horiguchi, K. Nagamatsu, and S. Watanabe, *"End-to-End Neural Speaker Diarization with Permutation-Free Objectives,"* in **Proc. Interspeech**, 2019.

[5] S. Horiguchi, Y. Fujita, S. Watanabe, Y. Xu, and K. Nagamatsu, *"End-to-End Speaker Diarization for an Unknown Number of Speakers with Encoder-Decoder Based Attractors,"* in **Proc. Interspeech**, 2020.

[6] I. Medennikov *et al*., *"Target-Speaker Voice Activity Detection: a Novel Approach for Multi-Speaker Diarization in a Dinner Party Scenario,"* in **Proc. Interspeech**, 2020.

[7] H. Bredin *et al*., *"Pyannote.audio: Neural building blocks for speaker diarization,"* in **Proc. IEEE ICASSP**, 2020.

[8] A. Radford *et al*., *"Robust Speech Recognition via Large-Scale Weak Supervision,"* in **Proc. ICML**, 2023.

[9] A. Baevski, Y. Zhou, A.-r. Mohamed, and M. Auli, *"wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations,"* in **Advances in Neural Inf. Proc. Sys. (NeurIPS)**, 2020.

[10] A. Gulati *et al*., *"Conformer: Convolution-augmented Transformer for Speech Recognition,"* in **Proc. Interspeech**, 2020.

[11] M. Bain, J. Huh, T. Han, and A. Zisserman, *"WhisperX: Time-Accurate Speech Transcription of Long-Form Audio,"* in **Proc. Interspeech**, 2023.

[12] A. Dawar, *"Why Deep Learning is the Best Approach for Speech Recognition,"* Deepgram Blog, Jun. 2024.

[13] M. Bain *et al*., *"Conformer-1: A Robust Speech Recognition Model Trained on 650K Hours of Data,"* AssemblyAI Blog, Oct. 2022.

[14] D. S. Park *et al*., *"SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition,"* in **Proc. Interspeech**, 2019.

[15] OpenAI, "Introducing Whisper," OpenAI, Sep. 21, 2022. [Online]. Available: https://openai.com/index/whisper/ (accessed Dec. 10, 2025).

Bubble-3

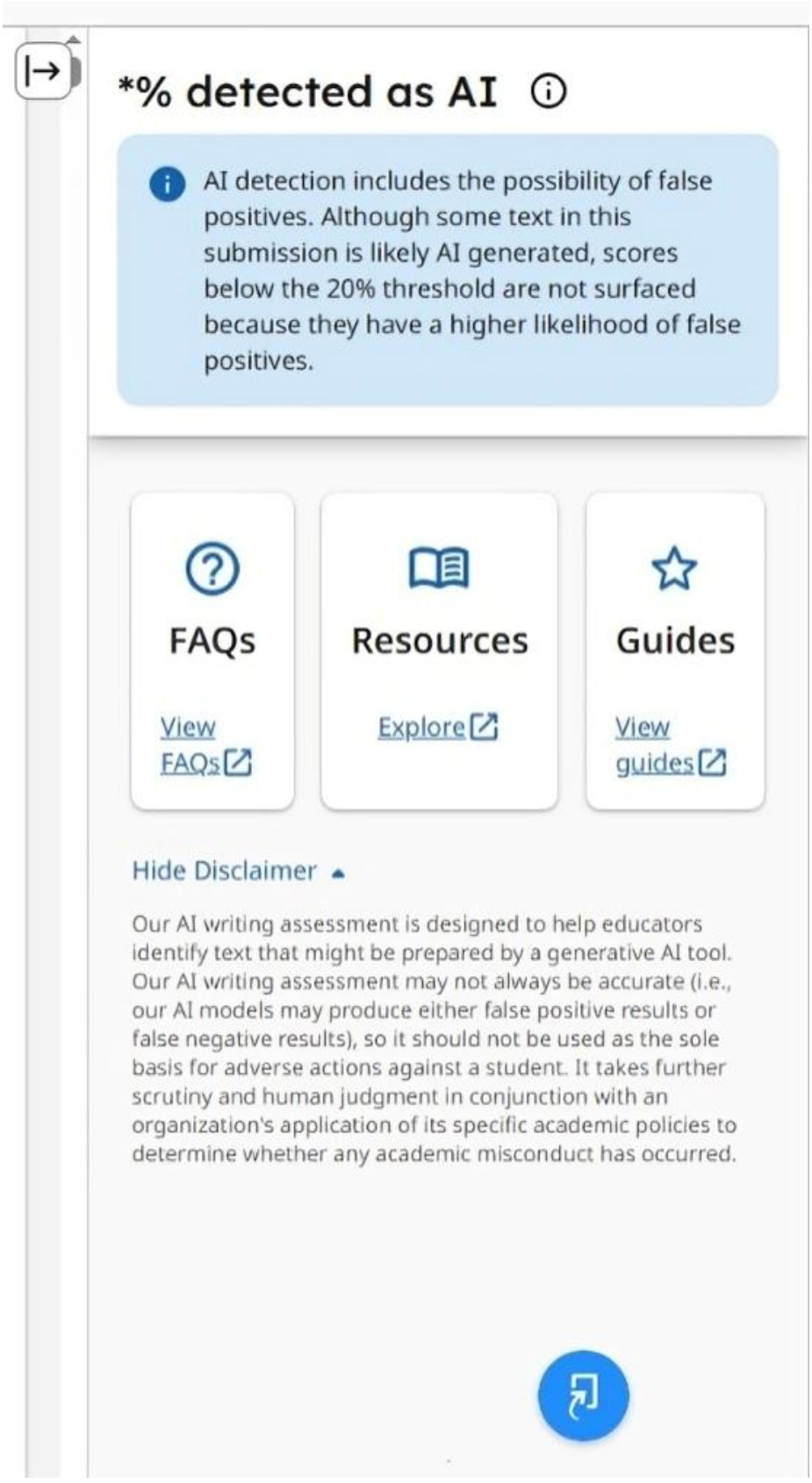
ORIGINALITY REPORT

4%	2%	0%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Higher Education Commission Pakistan Student Paper	2%
2	arxiv.org Internet Source	<1%
3	Submitted to Cranfield University Student Paper	<1%
4	repository.cuilahore.edu.pk Internet Source	<1%
5	Submitted to University of Hong Kong Student Paper	<1%
6	podbay.fm Internet Source	<1%
7	ijcrt.org Internet Source	<1%
8	Submitted to Anna University Student Paper	<1%
9	Submitted to University of Bristol Student Paper	<1%
10	dora.dmu.ac.uk Internet Source	<1%

Appendix B


*% detected as AI
AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.
FAQs
View FAQs
Resources
Explore
Guides
View guides
Hide Disclaimer
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (i.e., our AI models may produce either false positive results or false negative results), so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.