ChatGPT

GemmaVox: Offline Two-Way Voice Translator (Project Plan & Technical Writeup)

Introduction and Vision

GemmaVox is a **real-time two-way voice translation app** built in Flutter for iOS and Android. It empowers two people speaking different languages to carry on a natural conversation without an internet connection. By leveraging Google's **Gemma 3n multimodal AI model**, GemmaVox performs **speech recognition and translation entirely on-device** [1] [2]. This ensures user privacy and reliability even in offline or low-connectivity environments, aligning with the Gemma 3n Impact Challenge's focus on **accessibility** and **privacy-first, on-device innovation** [3] [1]. The app addresses a significant real-world problem – the language barrier – by essentially acting as a personal bilingual interpreter that fits in your hand.

How it Works: Each person speaks in their own language, and GemmaVox transcribes their speech and instantly translates it for the other person. With support for over **140+ text languages and 35 spoken languages** via the Gemma 3n model [4], the app can facilitate conversations between speakers of a vast array of languages. All processing is done locally; **no audio or text leaves the device**, ensuring conversations remain private and secure [1]. This offline-first approach is critical for use cases like travel in remote areas, emergency response, or communities with limited internet access. By combining **multilingual AI** with a simple, intuitive interface, GemmaVox aims to **bridge communication gaps** and demonstrate the power of on-device AI to drive positive impact (accessibility, cross-cultural communication) in line with the Challenge's vision [5] [3].

Project Scope: This document serves as both a project blueprint and a hackathon submission writeup. It outlines GemmaVox's requirements, UX design, technical architecture, and a development plan to build a functional prototype in **3 weeks** by a solo Flutter developer. We detail how the app's design choices prioritize usability and inclusivity (icon-driven UI, minimal text) and how the technical implementation leverages Gemma 3n's features (efficient on-device inference, multimodal input) within the tight timeline. Core milestones and deliverables are defined week-by-week, ensuring a feasible execution plan that meets the competition's criteria of impact, technical depth, and innovation.

Key Features and Requirements

GemmaVox's feature set is focused and purposeful, given the 3-week development timeframe. The key features and requirements include:

- **Real-Time Two-Way Conversation:** Supports **bidirectional voice dialogue**. The screen is split into two halves – one for each speaker – enabling face-to-face conversation mode. Each side has its own microphone button so that **two users can take turns speaking** (one on each end of the device). The UI is mirrored 180° for the two halves, so each person sees their interface oriented correctly from opposite sides.
- **Tap-to-Record with Visual Feedback:** Users initiate recording manually (tap to start/stop) – no automatic voice activity detection (to keep control simple and avoid false cut-offs). A clear visual cue (e.g. a pulsating microphone icon) indicates when recording is active. This way, each speaker

explicitly cues when they are speaking, making turn-taking unambiguous. (An option for **hold-to-talk** could be provided as an alternative interaction, but tap-to-record is the default.)

- **Live Transcription & Translation Display:** As a person speaks, **live transcription** of their speech appears in real time on *their own* half of the screen (in their spoken language). This lets the speaker see that their speech is being correctly recognized. Once the speaker finishes (and stops the recording), that provisional transcript is cleared, and an **translated text output** appears on the *other person's* half of the screen in the listener's language. In other words, each user only sees the **translation of the other person's speech**, not their own words. This makes the experience natural – akin to each person hearing the other's speech in their own language, except via text on screen.
- **Split-Screen Bilingual UI:** The **bottom half UI is in Language A** (for Speaker A) and the **top half is in Language B** (for Speaker B). The **top half UI elements are rotated 180°** so that if the phone is lying flat or held between two people, Speaker B can read their side upright. This split-screen approach (inspired by "conversation mode" in some translator apps) allows two users to face each other with the device in between, each viewing the interface from their side. All on-screen text (labels, prompts) are presented in the respective user's language when possible.
- **Icon-Driven Interface (Minimal Text UI):** To accommodate users of any language, the app's controls use universal icons and visuals instead of text labels. For example, the microphone button icon represents the record function, a language selection icon (globe or flag) represents language settings, a history icon represents past transcripts, etc. The overall theme is a clean **light theme** interface with high-contrast elements and large touch targets, making it usable for people of varying age and tech literacy. No static instructional text is used in the core UI (to avoid assuming any common language); the only text shown to users is the dynamic conversation content (the transcripts and translations themselves).
- **Language Selection with Auto-Detection:** Upon starting a conversation session, users can select Language A and Language B. The selection screen provides a list of supported languages (with native names and flags for clarity). Optionally, GemmaVox can **auto-detect a language**: a user can tap a "Detect Language" mic, speak a phrase, and the Gemma 3n model will identify the language spoken, automatically setting that as Language A or B. This leverages Gemma 3n's multilingual capabilities to simplify setup if a user isn't sure what language their partner speaks. Language preferences are remembered per session and can default to the device's locale. (Note: Only languages supported for audio input by Gemma 3n – 35 languages – will be available for voice conversation. However, the text translation output covers 140+ languages, so in future the app could allow one side to type if their language isn't in the 35 audio-supported list.)
- **Session Transcript Logging:** The app logs the conversation transcript for each session. Every time a speaker finishes speaking and a translation is shown, that exchange is added to a session log (with timestamp and language labels). At the end of a conversation (or at any time), the user can save or share the full transcript via the system share sheet (e.g. export as plain text). This feature is useful for record-keeping (imagine an important conversation at a clinic or business meeting – the transcript can be saved) and also for accessibility (e.g. a deaf user could later read back a conversation). Transcripts are stored privately on-device (and can be deleted by the user).
- **Past Conversations & Home Screen:** A simple home screen will list past conversation sessions (by date, or by manually saved title). Users can tap a past session to view the transcript again or share it. This allows quick access to recent interactions. From the home screen, the user can also start a **New Conversation**, which takes them to language selection for a fresh session. The home screen thus serves as both a history log and the start point for new use. (If privacy is a concern in some contexts, the app might default to not saving transcripts unless explicitly saved – but for the challenge prototype we assume logging for demonstration purposes, with an option to clear history.)
- **Entirely On-Device Processing: No internet or cloud services are required**. All speech recognition and translation is handled by the Gemma 3n model running locally via the

`gemma_flutter` plugin and the `gemma-3n-E2B-it-int4.task` model file [6]. There are **no user accounts, no login/registration, and no data upload**. This not only protects privacy but also speeds up interactions (no network latency). The use of the efficient **Gemma 3n E2B model** (with int4 quantization) ensures the app can run in real time on a typical modern smartphone, within ~2–3GB of RAM [7] [8]. (Text-to-speech output is *not* included due to the complexity of offline TTS and time constraints; users read the translated text. However, if a fast on-device TTS engine is available for a language, it could be integrated in the future as an optional feature to speak the translation audibly.)

- **Constraints and Simplicity:** Given the 3-week solo development limit, the project focuses on a **single core use-case** (two-person live conversation) and avoids extraneous features. All design choices favor simplicity: e.g. one conversation at a time (no complex multi-user or group features), one theme (light mode only), and essential controls only. This ensures a polished, functional prototype can be delivered on time. Optional nice-to-have features (like a dark mode, TTS, or advanced settings) are noted as potential future enhancements but are out of scope for the initial build.

UI/UX Design and Interaction Flow

Home Screen & Language Selection

Upon launching GemmaVox, the **Home screen** greets the user with a simple layout: a prominent button to **"Start New Conversation"** and a list of recent conversation transcripts (if any exist). Each past conversation entry shows the language pair and date/time; tapping it opens the transcript viewer. The home screen uses an icon (e.g. a conversation icon or "+" symbol) for the new conversation action, and a history/list icon for the transcripts section – aligning with the icon-driven design (no text labels). This screen is intentionally minimal to avoid confusion.

Starting a new conversation leads to the **Language Selection** interface. Here the user picks the two languages for the conversation: "Your Language" and "Partner's Language". The UI might present two side-by-side dropdowns or a two-step selection (first select Language A, then Language B). Each language is displayed in its own script (for example, it shows "Español" instead of "Spanish" for Spanish, alongside a flag or globe icon) to ensure recognition by native speakers. An optional **Language Auto-Detect** feature is available: next to a language picker is a small mic icon labeled "Detect Language". If the user taps it and says a sample phrase, the app uses Gemma 3n to identify the spoken language and fills that selection automatically. This is useful in scenarios like meeting someone whose language you don't recognize – you can have them speak a greeting and the app will suggest what language it is. Once Language A and B are set (and an orientation is determined – e.g. Language A = bottom UI, Language B = top UI), the user proceeds into the conversation mode screen.

Conversation Mode UI Design

Figure 1: Conceptual wireframe of the GemmaVox conversation screen. The app's main conversation interface is split into two halves, each oriented towards one of the two speakers. In **Figure 1**, the bottom half is for **Language A (Speaker A)** and the top half (rotated 180°) is for **Language B (Speaker B)**. This mirror-image layout means if two people face each other with the phone between them, each person sees their side of the screen in their own language and upright.

Each side features a large **microphone button** (the circular icons in the figure) – one at the bottom-center of the bottom half for Speaker A, and one at the top-center of the top half for Speaker B (which appears at that person's bottom due to rotation). These buttons are color-coded or labeled with the language name (or flag) to avoid confusion. The **active speaker presses their mic button to speak**.

While recording, the mic icon may glow or pulse to clearly show the **"recording in progress"** state. This avoids accidental overlaps (only one side should record at a time).

Each half of the screen also serves as a display area for text. **When Speaker A is talking**, their speech is transcribed in real-time and shown as overlay text in the bottom half (e.g. a live caption that updates as Gemma recognizes the speech). This text is in Language A (Speaker A's speech). It appears just above the mic button area so Speaker A can easily read it. This live transcript is visible to Speaker A (and optionally visible to Speaker B as well, though flipped upside-down from B's view; however, B doesn't necessarily need to read it). The moment Speaker A finishes and stops recording, the **live transcript on A's side is cleared** (to avoid cluttering the screen with what A just said). Then, after a brief processing moment, the **translated text appears on Speaker B's side** (the top half) in **Language B**. This translated output is typically shown in a speech bubble or highlighted text box, making it clear that it's the result of the other person's speech. Speaker B can now read the translation of what A said, in B's own language.

The roles then swap: if Speaker B replies, they tap their mic (top half) and speak in Language B. B's side will show a live transcription (in B's language) of B's speech, then clear it and display a translated text on A's side in English (or whatever Language A is) for Speaker A to read. In this way, the conversation flows naturally turn-by-turn. Both users only see **incoming messages in their own language**, which makes the experience feel like reading subtitles for the other person's speech in real time.

The UI also provides some additional visual feedback and controls: for instance, a small indicator might show when the system is "translating..." (useful if there's a slight delay after someone finishes speaking, to reassure the user that processing is happening). If the Gemma 3n model's response is quick, this may not be necessary, but a subtle spinner or progress bar can help. Additionally, a **clear session / restart conversation** button might be present (perhaps as a trash bin icon) if users want to reset the conversation and pick new languages. A **back button** (or swipe gesture) would allow returning to the home screen mid-conversation (with a prompt to save or discard the current transcript).

Interaction Flow Example

To illustrate the user flow, here's an example of a **typical exchange using GemmaVox**:

1. **Language Setup:** Alice (an English speaker) meets Bopha (a Khmer speaker). Alice opens GemmaVox, selects English as her language and uses auto-detect for Bopha's speech, which identifies Khmer. The app is now set to English ↔ Khmer mode. They place the phone on a table between them, Alice facing the bottom half, Bopha facing the top half.
2. **Alice Speaks:** Alice taps her microphone button and says, "Hello, how are you?" in English. As she speaks, she sees the words "Hello, how are you?" appearing on her side of the screen (live transcription). Bopha sees the mic icon on Alice's side pulsing, but no text on her side yet.
3. **Translation to Khmer:** Alice taps the mic button again (or it auto-stops after a pause). Alice's transcribed text disappears from the bottom half. A moment later, on Bopha's half, the Khmer translation of Alice's phrase appears (in Khmer script). Bopha reads the translated text on her side. (Alice might see that text too but upside down; however, Alice doesn't need to read it.)
4. **Bopha Responds:** Bopha now taps her (top half) microphone and speaks in Khmer to reply. She sees her Khmer speech being transcribed live on the top half of the screen. Alice sees Bopha's mic icon indicating recording.
5. **Translation to English:** When Bopha finishes, her live transcription clears from the top. Then Alice's side of the screen displays the English translation of Bopha's message (e.g. "I'm fine, nice to meet you.") Alice can now read it in her language.

6. **Turn-Taking Continues:** They continue conversing by alternating turns pressing their respective mic buttons. Each utterance is translated in real time to the other's language and displayed on the appropriate half of the screen. They maintain eye contact and a natural flow, only occasionally glancing at their own side of the screen to read the other's words.
7. **Session End & Save:** At the end of their conversation, Alice taps an "End" icon. The app has logged the full dialogue. Alice can open the transcript (which might show entries like "Alice (EN): Hello, how are you? —> [Khmer translation]" and "Bopha (KM): [Khmer text] —> I'm fine, nice to meet you." for each turn). She taps the Share icon to export the text via email to herself for records. The conversation is saved in the app's history with a timestamp and labeled "English ↔ Khmer with Bopha".

Throughout this flow, the **visual design remains uncluttered and intuitive**: large buttons for microphone, clear text rendering, and a layout that mimics a chat or subtitle display. The use of **two distinct colors** (one for each side's text bubbles) can also help users quickly identify who said what. For example, all of Speaker A's utterances (as seen by B in translation) might appear in blue bubbles on B's side, and Speaker B's utterances (translated for A) in green bubbles on A's side. This color-coding corresponds to the mic button colors, reinforcing the association (useful if both translations appear on one device).

Accessibility considerations in the UI include **readable font sizes** (especially for longer translated text), high-contrast text/background for outdoor use, and simple layouts for **easy touch navigation**. The app avoids any small text or fiddly menus – essentially the user is either on the home screen or in the conversation interface with just one tap action. This makes it usable for people who may not be tech-savvy or literate in the same language as the device's system language. By keeping the interface language-neutral and iconographic, GemmaVox is **universally usable**, relying on the intelligence of the Gemma 3n model to handle all language complexity behind the scenes.

Technical Architecture

Overview

GemmaVox's architecture consists of a Flutter client application augmented by on-device AI capabilities from the Gemma 3n model. The high-level components include:

- **Flutter Mobile App (UI & Logic):** The Flutter framework allows a single codebase for both iOS and Android deployment. The app is structured with a focus on two main flows (the conversation interface and the transcripts/history management) and leverages Flutter's widgets for an interactive UI (split-view, buttons, text display). State management can be handled using a lightweight approach (e.g. Provider or setState for simplicity, given the short timeline) since the app's state is not very complex (mainly the current conversation state and a list of past conversations). Platform-specific integrations (like accessing the microphone) are abstracted via Flutter plugins.

- **Gemma 3n On-Device AI Model:** At the core is the **Gemma 3n E2B model** (with ~5B raw parameters, effectively 2B when using selective loading) running locally [7]. This model is packaged as a `.task` file (e.g. `gemma-3n-E2B-it-int4.task`) and accessed using Google's `gemma_flutter` plugin (v0.9.0) [6]. The plugin likely uses the Google AI Edge runtime to execute the model efficiently on device (possibly leveraging GPU/NN accelerators or optimized CPU paths as available). The model being quantized to int4 means it has a small memory footprint for its size, enabling it to run on typical smartphones without exceeding memory limits. (For context, Gemma 3n's architecture innovations like **PLE caching** and **MatFormer** allow it to

run with as little as ~2GB RAM usage for the E2B variant [7], which is within range of mid-tier phones).

- **Audio Capture & Processing:** The Flutter app uses the device microphone to capture audio. This is done through either the `gemma_flutter` plugin's API (if it directly supports audio input streams) or a separate audio plugin (such as Flutter Sound or just the built-in `Recorder`). The audio is captured in a suitable format (PCM 16kHz mono) expected by the model. Gemma 3n being multimodal means it can accept raw audio waveforms (encoded internally to tokens) and output text [9]. The app will feed the recorded audio buffer to the Gemma model for inference. The **Gemma model performs two tasks** here in sequence: **speech recognition** (convert audio to text in the source language) and then **text translation** (convert the recognized text to the target language). This could be implemented by using the model in a multi-stage prompt or by leveraging any built-in translation capability. For example, if the Gemma model supports instructions, the app can prompt: "Transcribe the following audio in `<Lang A>` and then translate to `<Lang B>`." Since Gemma 3n was designed to handle audio and text and produce text outputs, it can likely perform end-to-end speech translation when prompted appropriately [2]. If needed, the app can break it down: first call Gemma to transcribe audio to text (in source language), then call Gemma again with that text asking for translation to target language. The translation model doesn't require an external API – it uses Gemma's multilingual text generation capabilities. The round-trip is still on-device and fast, especially for short utterances.

- **Real-time Feedback Loop:** To achieve live transcription while the user is speaking (for display on their side), a strategy is required since the model typically processes after receiving the full audio. One approach is to use a lightweight on-device speech recognizer or Gemma in streaming mode (if supported) to get interim results. If Gemma 3n (E2B) is too heavy for true streaming on a phone, an alternative is to break the audio input every second and run partial transcription. However, for simplicity in the MVP, we might simulate "live" transcription by recording audio in short chunks and updating text, or simply show the transcription after the user stops speaking. (The design calls for showing text **while** speaking; implementing that in 3 weeks might be challenging. We assume Gemma's audio input can yield incremental results – if not, a possible compromise is to display the transcription only once recording stops, which is a bit less dynamic but much simpler. Nonetheless, the goal is to attempt near-real-time transcription feedback for a better UX). We ensure that processing is done **asynchronously** so the UI remains responsive – Flutter's isolate or the plugin's internal threading will handle the heavy model inference off the main thread.

- **Data Storage (Transcripts & Settings):** For storing conversation transcripts, the app will use on-device storage. Options include a small SQLite database or simply storing each session as a plaintext file (or JSON) in app documents directory. Given time constraints, a straightforward approach is to append each conversation's lines to a text file and save it with a timestamp in the filename or use a JSON structure for more structured data. Flutter's filesystem API or path_provider plugin will be used. The history list on the home screen is generated by reading these saved files (and maybe parsing the first line or a metadata file to show summary info). User settings like selected languages or last used language can also be stored locally (using SharedPreferences for quick key-value storage). All data stays on the device; **there is no cloud backup or sync**.

- **External Integration:** None required beyond what's mentioned – no external servers or internet calls. If any platform-specific native code is needed (for example, iOS permission for microphone, Android handling of audio focus), those will be handled via Flutter's plugin mechanisms and platform channels. The `gemma_flutter` plugin presumably includes the necessary native code

to load and run the model using either TensorFlow Lite or a similar mechanism behind the scenes. The model file may be included in the app bundle (if size permits) or downloaded on first launch from a provided offline URL (for the hackathon, likely included to simplify judging).

Architecture Diagram: (Conceptually, the flow is as follows)
User A (Lang A) –[speaks]→ **Mic input** (Flutter captures audio) –→ **Gemma 3n Model** (on-device): *speech-to-text (Lang A)* → *text-to-text (Lang B)* –→ **Translation Text** appears on **User B's screen**. The reverse happens for User B's speech. Both original and translated texts are logged in storage. In essence, Gemma 3n functions as the brain in the middle, receiving audio or text and outputting the translated text. Flutter handles the UI, user interaction, and orchestration of these calls to Gemma. The on-device model makes this pipeline low-latency and secure.

From a module perspective, we can imagine the following Flutter app structure:
- `main.dart` – initializes the app, loads the Gemma model (using gemma_flutter's API to load the .task file perhaps at startup asynchronously), and shows the Home screen.
- UI Modules: `HomeScreen` (list of conversations, new convo button), `LanguageSelectScreen`, `ConversationScreen`, `TranscriptScreen` (for viewing a saved transcript). These manage the presentation.
- Model/Service: `GemmaService` – a Dart singleton that wraps calls to the Gemma plugin. It might have methods like `transcribeAudio(Uint8List audioData, String sourceLang)` and `translateText(String text, String targetLang)`, or a combined `translateSpeech(audio, sourceLang, targetLang)`. This service loads the model at startup and keeps it in memory for reuse (to avoid reloading cost each time). It handles formatting prompts or using plugin-provided translation functions.
- Data: `TranscriptStorage` – handles reading/writing transcripts from the file system, and perhaps a small model class for Transcript (with fields like languages, list of utterance pairs, timestamp). This could also handle exporting to share.
- Platform integration: minimal if using existing plugins. Just ensure microphone permission is requested on first use.

Given Gemma 3n's capability, the same model handles both ASR and translation, simplifying the architecture (no need for separate ASR engine or translation API). This unified approach showcases the **multimodal strength of Gemma 3n** – it's explicitly designed to understand audio and text in many languages within one model [9].

Technical Challenges and Mitigations

A few technical considerations in this architecture:
- **Performance on Device:** Running inference for speech transcription and translation in real-time is computationally intensive. Gemma 3n E2B is optimized for mobile but we will still ensure to use int4 quantization and possibly the fastest execution provider available (e.g. utilizing NNAPI or GPU on Android via the plugin, Metal on iOS). We might run the model on a separate isolate or use the plugin's async calls so that the UI stays 60fps responsive. Caching the model in memory after first load will avoid delays on subsequent calls. The expected latency for a short utterance (~5 seconds of speech) might be a couple of seconds for transcription+translation – we will profile early and adjust UI to handle that (perhaps by showing a "translating..." indicator as mentioned).
- **Accuracy of Transcription/Translation:** Gemma 3n is state-of-the-art, but to help it along, we may apply prompt techniques if needed (e.g. instruct it with a system prompt like "You are a translator."). If we find it sometimes misrecognizes speech due to accent or noise, we might consider using a small VAD (voice activity detector) to trim silences or adjust microphone gain. These are stretch goals; the

baseline plan is to rely on Gemma's robust training on diverse audio [10] to handle various accents and languages.
- **Memory Footprint:** Including a multi-gigabyte model in a Flutter app needs caution especially for iOS (app size limits) – we may choose the smaller E2B model (as planned) which is ~2B effective parameters. The int4 quantized model file might be on the order of a few hundred MB. We will make sure to test on a typical device (e.g. Android phone with 6GB RAM) to confirm it can load. In the worst case, we could fall back to running Gemma in a slightly reduced mode or limit concurrency (only one instance loaded at a time) to fit memory. However, as per Google's documentation, the E2B model was designed for 2GB RAM usage which is feasible [7].
- **Cross-Platform Differences:** The gemma_flutter plugin and model should work on both Android and iOS. We'll need to handle microphone permission flows on each platform. Also, file storage paths differ (we use Flutter's path provider to abstract this). UI testing on both platform form factors will be done, but Flutter ensures consistency for the most part.
- **No Cloud Fallback:** Since we commit to offline-only, if the model fails or the language is unsupported, we won't have a cloud service to back us up. We mitigate this by clearly restricting the language choices to what the model can handle (so users can't select an unsupported language). We'll also test a variety of phrases offline beforehand to ensure the model's responses are reasonable. In a scenario where the model yields an unusable result (e.g. gibberish), the app can allow the user to ask to repeat or could display an apology ("Could not understand, please try again" – ideally in the user's language). This edge-case error handling will be built in for robustness, albeit we hope it's rarely needed given Gemma 3n's training breadth.

Development Plan (3-Week Timeline)

To complete GemmaVox in 3 weeks, we break the work into weekly sprints with specific milestones:

Week 1: Foundation and Prototype

- **Project Setup:** Initialize the Flutter project (set up required plugins: `gemma_flutter`, any audio recording plugin, and path_provider for file storage). Obtain the Gemma 3n E2B model file and ensure it's accessible (include in assets or set up download). Verify the license compliance for using the model.
- **Gemma Integration POC:** Write a small prototype function to load the Gemma model and perform a basic test (e.g., transcribe a known audio clip or translate a sample text) to confirm the on-device inference works. This will flush out any issues early (like large file handling, plugin integration, or performance gotchas). For example, feed an English sentence and get a French translation just to test text generation. Ensure logging of model output is working.
- **UI Skeleton:** Design the core screens in Flutter (statically). This means building the layout for the Home screen, Language selection, and Conversation screen without full functionality. Place the buttons, text areas, and any icons. Get the split-screen working: e.g., use a `Transform.rotate` widget for the top half to rotate it 180°. Ensure that text can be displayed upside-down for the top half (which might involve rotating that container). This is also the time to gather or design simple icons (mic icon, share icon, etc.) either from Flutter's Material/Cupertino icons or custom if needed.
- **Microphone & Audio Handling:** Implement basic microphone capture. Likely using `gemma_flutter` directly if it supports an audio input method. If not, use an audio plugin to record a short clip on button press and save to memory. By end of week, be able to record a few seconds of audio from the mic and print out some representation (even waveform data or size) to ensure the pipeline is connected.
- **Milestone (End of Week 1):** A minimal app where you can navigate from Home -> Language Select -> Conversation screen. On the conversation screen, pressing the mic button starts

recording and then stops (dummy implementation with a timer or second tap), and we call the Gemma model to process the audio (perhaps stubbed or actual). Then display either a dummy translated text or real output if possible. Essentially, a *single-turn translation demo* should work by end of week 1 (e.g., user speaks English, app prints out a translated text in Spanish on the other half). This proves the main technical risk (model inference on-device) and sets up the UI framework.

Week 2: Core Functionality Completion

- **Two-Way Conversation Logic:** Expand the prototype to handle alternating turns. Implement the logic to manage whose turn it is, lock the UI for one user while the other is speaking (to avoid simultaneous presses). Automate the flow such that after one side's translation appears, the other side's mic is enabled for reply. This involves state management to track "Awaiting A's speech" vs "Awaiting B's speech" etc.
- **Real-time Transcription Display:** Integrate live transcription if feasible. If the Gemma API supports streaming results, use it to update the text on the fly. If not, simulate by splitting the recording or by showing the final transcript after recording and before translation. Ensure that when one user is speaking, only their side shows text (and maybe also hide any previous translation on that side to avoid confusion). Fine-tune the UI update timing: clearing the text at the right moment and showing the translation on the opposite side.
- **Language Detection Feature:** Implement the language auto-detect function on the language selection screen. Likely this means recording a snippet of audio and calling Gemma in a mode where it outputs the language identity. If Gemma doesn't provide a direct function, an alternative is to use Gemma's text output by prompting: "What language is this sentence: <audio> ?" or using a lightweight open-source language identification if necessary. Since Gemma is trained on multilingual data, it can probably identify language from the audio internally [11] [12]. This feature should be tested with a few languages to verify accuracy.
- **Transcript Logging:** Build out the data model to store the conversation. For each turn, capture the original text and the translated text. As soon as translation is displayed to the user, append that as a line to the in-memory transcript for the session. By end of conversation (or onPause), save the transcript to a file. Implement the Home screen list to read saved transcripts (for now, it could just list files by timestamp). Also implement the Transcript viewer screen if desired, or simply open the file in a text view.
- **UI Polish & Usability:** This week, also refine the UI look. Add the color-coding for text bubbles, animate the mic pulse (using an `AnimationController` in Flutter to scale the icon or glow it when recording). Ensure the rotation aspect looks correct on a physical device (test on an actual phone or emulator rotated). Add any additional guidance in the UI (maybe a quick overlay tutorial arrow pointing to each mic on first use). Keep all text (like tutorial hints) optional or in both languages if possible; perhaps a better approach is to include a pictorial tutorial in the submission rather than in-app, to maintain the no-text UI principle.
- **Performance Testing:** By mid-to-late Week 2, test the end-to-end conversation flow on a real device if possible. Measure the time from speaking to translation appearing. If any step is too slow, consider optimizations (e.g., shorter audio chunks, or limiting audio recording duration to 5 seconds per turn as a trade-off). Also ensure memory usage is stable (no leaks; the model doesn't reload each time unnecessarily).
- **Milestone (End of Week 2):** The app should be fully functional in terms of features: two people can have a conversation across languages with translations appearing appropriately, and a transcript is saved. All primary features (except maybe advanced error handling or fringe cases) are implemented. Essentially, a working MVP that could be demoed.

Week 3: Testing, Optimization, and Submission Prep

- **Robustness & Bug Fixing:** This week is for ironing out any bugs discovered in week 2. For example, fix any UI glitches (text overlapping, rotation issues), race conditions (if two buttons pressed at once), or memory issues (ensure microphone is properly released after use, etc.). Test different language pairs (especially one Latin-script vs one non-Latin like English↔Khmer, or Arabic↔French, etc.) to ensure the fonts display correctly and the UI accommodates different text direction or length. For instance, languages like Arabic or Hebrew are RTL (right-to-left), which might need Flutter's bidi support – we should test and adjust alignment for such cases. If RTL proves too complex to perfect in time, we might restrict to LTR languages in the demo, but ideally show that it can handle it.
- **User Experience Refinement:** Fine-tune the visual elements: adjust font sizes, bubble sizes (e.g. longer translations might need multi-line bubbles), and ensure the mic buttons are easy to reach (maybe allow tapping anywhere on that half to trigger recording, not just the icon, for accessibility). Add a subtle sound or vibration on tap of mic to give feedback (if time allows). Also consider adding an **onboarding screen** or pop-up the first time to explain usage (perhaps just an image illustrating the two halves, since we avoid text). This is optional but can strengthen the submission's clarity.
- **Documentation & Challenge Alignment:** Prepare the required documentation for submission. This includes finalizing this writeup (the technical report) explaining how Gemma 3n is used, the problems addressed, etc. Emphasize the **Impact & Vision** in the context of the challenge – ensure the app's benefit (e.g. a tourist communicating with locals, a humanitarian worker talking to disaster victims with language barriers, etc.) is well articulated. We may include a section in-app or in documentation about privacy (e.g. "All processing on-device with Gemma 3n – nothing is sent to cloud" as a badge of honor).
- **Demo Video Production:** Although not part of coding, the hackathon requires a video. We'll spend time capturing the app in action. This might involve two people demonstrating conversation using GemmaVox in a real scenario. The focus will be on storytelling: e.g. showing how without the app they couldn't understand each other, then with GemmaVox they communicate smoothly. We ensure the video highlights the app UI clearly and any unique features (like the flipped UI, the offline nature – maybe by showing on airplane mode). This will be done towards the end of week 3 to incorporate the polished app version.
- **Final Testing and Optimization:** Run the app on multiple devices if possible (Android phone, an older phone, an iPhone) to ensure cross-platform operation. Optimize any last performance hiccups (for example, if loading the model initially is slow, consider showing a splash screen or a "Loading AI..." indicator at startup). Confirm that the binary size is manageable and the app doesn't violate any app store guidelines (since this is a prototype for judging, not actual release, but still good to know).
- **Milestone (End of Week 3):** Project ready for submission – code on GitHub with documentation, a polished demo video, and the writeup (this document) completed. All challenge requirements (functioning app, use of Gemma 3n, addressing a real problem, etc.) are satisfied. At this stage, GemmaVox will have been tested in a realistic conversation and we can be confident in its demo.

Design Rationale and Accessibility Considerations

The design of GemmaVox was guided by the principles of **accessibility, simplicity, and privacy**:

- **User-Centered Design:** The split-screen conversation mode directly mirrors how two people naturally face each other. This design was chosen so that each user has a dedicated space in the app that feels like "theirs" – reducing cognitive load. The lack of extraneous UI chrome (menus, text labels, etc.) means even users with very little tech experience can understand: *"press the big*

button to talk, then read what the other person said." By minimizing required reading (no menus in small text), we accommodate users who might be illiterate or only comfortable in their native script. The light theme with large text ensures readability for users with visual impairments (within reason – the app assumes users can read in at least one language; those who cannot read would need an auditory output which is future work).

- **Accessibility:** GemmaVox directly contributes to accessibility in communication – it can be considered a tool for both language accessibility and potentially for the **deaf or hard-of-hearing**. For instance, a deaf user could use it to have a conversation with a hearing person of another language: the hearing person speaks, GemmaVox transcribes and translates it to the deaf person's language so they can read it (this addresses both the hearing impairment by providing text, and language barrier by translating) [5]. Similarly, a person who cannot speak could type and have the other read – while our current design focuses on voice, the underlying translation could be adapted. The on-device nature is crucial for certain accessibility scenarios, e.g. rural areas with no internet or in emergencies where infrastructure is down – the app would still function. We also considered physical accessibility: large buttons for those with motor difficulties, and a hold-to-talk mode if keeping a steady tap is hard for some users. The absence of quick timeout VAD means people who speak slowly or have long pauses (e.g. due to speech impairments) won't be cut off – they control the start/stop of recording.

- **Privacy-by-Design:** Because conversations can often be sensitive (personal or business talks, medical consultations, etc.), privacy is paramount. Using Gemma 3n offline ensures that *no audio or text is ever uploaded* to a server [1]. Users don't have to worry about conversations being eavesdropped or saved in the cloud. This could be a key differentiator from mainstream translator apps which often require internet and send data to cloud APIs. Especially in the context of the challenge (e.g. crisis response or healthcare use cases), this privacy can be vital (think of a patient discussing symptoms through a translator app – with GemmaVox, that stays on the device only). We intend to highlight this in the app info (maybe a note "Powered by Gemma – runs entirely on your device"). The app also doesn't collect any personal data or require login, aligning with a privacy-first ethos.

- **Simplicity and Focus:** In a hackathon setting (and for real-world adoption), keeping the app simple was key. We focused on doing one thing well: real-time voice translation for two users. This constraint led to many design decisions: e.g., not overloading the UI with too many options, using the device orientation trick rather than requiring two separate screens, etc. Simplicity also aids performance – fewer background processes and a straightforward UI loop means more resources for the Gemma model to do its job. The timeline constraint (3 weeks) also encouraged simplicity: we prioritized features that directly contribute to a smooth conversation experience, and cut anything that was "nice but not necessary". This lean approach not only makes the project achievable, but also yields a cleaner user experience with less potential confusion or bugs.

- **Visual Aids and Feedback:** Recognizing that dealing with AI and speech might be unfamiliar to some, we incorporate intuitive visual feedback. The pulsing microphone is one example – it gives a clear signal "now recording" which is universally understood (even if one doesn't read English). Another is showing the transcribed text; even though only the speaker needs to see their transcript, showing it reassures them that the app "heard correctly". If the transcription is off, they can retry immediately. Color coding the conversation turns (as mentioned) acts as an implicit guide: e.g., if all of Alice's translations are blue and Bopha's are green, each knows at a glance which language any given text is in (useful if they take turns quickly). We avoid relying on sound (like beeps) as primary cues, because in noisy environments or for hearing-impaired users

those are less useful; visuals are primary, though a gentle beep/vibrate on start/stop can be secondary feedback.

- **Alignment with Challenge Goals:** GemmaVox exemplifies an **"AI for Good"** project – it breaks down language barriers in a private, offline manner, directly answering the Challenge's call for accessibility innovations [3]. It shows how a powerful multimodal model like Gemma 3n can be applied beyond chatbots to a tangible human interaction problem. By being on-device, it also highlights the **on-device innovation** aspect: previously, such real-time translation would require cloud services, but we demonstrate it can now run locally thanks to Gemma 3n's efficiency [8]. This can inspire further applications that require real-time understanding on the edge (for example, in education or crisis communication tools). Finally, the app's focus on a **meaningful real-world use case** – enabling conversation where it was not possible before – aims to score highly on Impact & Vision in the judging criteria. We plan to emphasize a narrative of two individuals connecting across a language divide as the core story, showcasing the *human impact* of this technology.

Conclusion

In summary, **GemmaVox** is a carefully scoped project that merges a simple, effective UX with cutting-edge on-device AI to solve a real-world problem. This document has outlined how we will implement GemmaVox within a tight timeframe, covering everything from UI sketches to the technical pipeline and timeline. By focusing on the essentials (intuitive design, core functionality, and robust use of Gemma 3n's capabilities), the project is well positioned to succeed in the Gemma 3n Impact Challenge.

We have deliberately chosen design and architecture strategies that mitigate risk (using Flutter for fast cross-platform dev, using a single AI model for all tasks, etc.), and we've aligned our plan with the **Challenge's objectives of accessibility, privacy, and innovation**. If executed as planned, GemmaVox will not only be a strong hackathon submission but also a foundation for a product that could be expanded post-challenge (adding features like text input mode, text-to-speech output, more UI polish, etc.).

Most importantly, GemmaVox demonstrates the **transformative potential of on-device AI**: it puts a live translator in the hands of anyone, anywhere – no internet required. This empowerment of users to communicate freely and securely is the kind of impact we believe Gemma 3n was created to enable. We are excited to bring GemmaVox to life and showcase how technology can bring people together across language barriers, in real time, while respecting their privacy and circumstances.

Sources: The development of GemmaVox builds on information and capabilities described in Google's Gemma 3n documentation and the competition materials. Key references include the Gemma 3n model overview (highlighting its audio input support, multilingual range, and on-device optimizations) [13] [4], as well as the Gemma 3n Impact Challenge brief (emphasizing offline, private, impactful applications) [1] [3]. These informed our design and ensure that GemmaVox remains true to the spirit of the challenge while being technically feasible in the given timeframe.

---

[1] [3] [6] [8] [12] Competition Rules.txt
file://file-CQMi1yQQNDCLsF8uhjRMqa

[2] [13] Gemma 3n model overview | Google AI for Developers
https://ai.google.dev/gemma/docs/gemma-3n

4 7 Introducing Gemma 3n: The developer guide - Google Developers Blog
https://developers.googleblog.com/en/introducing-gemma-3n-developer-guide/

5 11 Gemma 3n Comp.txt
file://file-MMRW7jD2yD2YsH73XByV51

9 10 google/gemma-3n-E4B-it-litert-preview · Hugging Face
https://huggingface.co/google/gemma-3n-E4B-it-litert-preview