**Setting Android**

here's how to ensure that "USB debugging" is enabled on your Pixel 3a XL, explained in English:

**Part 1: Enable "Developer options"** (If you've already done this, you can skip to Part 2)

1. **Go to your phone's "Settings":**
   * Swipe down twice from the top of the screen to open the Quick Settings panel, then tap the gear icon ⚙️ (Settings).
   * Alternatively, find and tap the "Settings" app in your app list.
2. **Go to "About phone":**
   * In the Settings menu, scroll down and tap "About phone."
3. **Find "Build number" and tap it repeatedly:**
   * On the "About phone" page, scroll down to find "Build number."
   * **Tap "Build number" repeatedly about 7 times.** During this process, you'll see a toast message counting down how many more taps are needed (e.g., "You are now X steps away from being a developer").
   * When you see the "You are now a developer!" message, Developer options have been successfully enabled. If your phone has a screen lock (password, pattern, PIN), you might be prompted to enter it.

**Part 2: Enable "USB debugging"**

1. **Go to "Developer options":**
   * Go back to the main "Settings" menu.
   * Tap "System."
   * In the "System" menu, you should now see and be able to tap "Developer options." (It's usually located near "System update").
2. **Find and turn on "USB debugging":**
   * Once in "Developer options," scroll down the page.
   * Look for the section titled "Debugging."
   * In this section, find the "USB debugging" option.
   * **Ensure the switch next to "USB debugging" is in the ON position.** If it's off, tap the switch to turn it on.
   * A confirmation dialog will pop up asking "Allow USB debugging?". Tap "OK" or "Allow."

**Part 3: Connect to Your Computer and Verify**

1. **Connect your Pixel 3a XL to your computer using a USB cable.**
2. **Authorization prompt on your phone:**
   * When you connect your phone to a new computer with USB debugging enabled for the first time, a dialog box will appear on your phone's screen, typically titled "Allow USB debugging?". It will also show the computer's RSA key fingerprint.
   * **Check the box for "Always allow from this computer"** (this is recommended so you won't be asked again for the same computer).
   * Then tap "Allow" or "OK."
3. **Verify in Android Studio:**
   * Open your Android Studio.
   * In the Android Studio toolbar at the top, there's a device selection dropdown menu (usually to the left of the green "Run 'app'" button ▶️).
   * If USB debugging is successfully enabled and your phone is correctly connected and authorized, you should see your device name, such as "Google Pixel 3a XL," listed in this dropdown menu.

The application uses Google's speech recognition service, which is accessed through Android's native SpeechRecognizer API.

1. **Speech Recognition (Speech-to-Text)**:

* This functionality converts audio/spoken words into text
* In your app, this is handled by Android's built-in SpeechRecognizer API
* It's a separate technology that works independently of any LLM
* This is why your app can recognize speech even without an LLM

1. **LLM (Large Language Model)**:

* This processes and understands text that has already been converted from speech
* It generates intelligent responses based on the text input
* In your app, the LLM would receive the text after speech recognition is complete

**Whisper TFLite Project Report**

**1. Project Overview**

Whisper TFLite is a fully offline speech transcription system that packages OpenAI's Whisper model into an Android application. The project uses a quantized TensorFlow Lite model running on mobile devices

**2. File Structure, Functions, and Implementation Components**

**2.1 Model Files**

* whisper-tiny.tflite: multilingual support
* whisper-tiny-en.tflite: English-only, optimized for smaller size

These are quantized versions of OpenAI Whisper models in TensorFlow Lite format.

**2.2 Vocabulary Files**

* filters\_vocab\_multilingual.bin: Vocabulary mapping for multilingual model
* filters\_vocab\_en.bin: Vocabulary mapping for English model

These files map model token outputs to readable text.

**2.3 Java & Native Code**

* MainActivity.java: Entry point of the app, handles UI interaction and calls transcription functions
* Whisper.java: High-level wrapper class for Whisper functionality. Handles background audio processing, model loading, and result callbacks using observer pattern.
* Recorder.java: Audio recorder responsible for capturing microphone input in required format, using Android AudioRecord API to record at 16kHz, mono, 16-bit PCM.
* WhisperEngineNative.java: Interface to native C++ engine via JNI

**2.4 Execution Flow Between Java and C++**

1. Java captures user input or button press
2. Java calls nativeTranscribeFile() via JNI
3. C++ loads the audio, extracts features, runs inference
4. C++ returns transcription result to Java
5. Java updates UI using handler

**2.5 Threading with Handler**

* Uses Java Handler to transfer recognition results from background thread to UI thread
* Maintains responsive UI and safe UI updates
* Example:

handler.post(() -> tvStatus.setText("Processing..."));

handler.post(() -> tvResult.append("What's the weather today?"));

**2.6 DTLN Noise Suppression (Optional Extension)**

* Use quantized [DTLN](https://github.com/breizhn/DTLN) model
* Flow: Raw audio → DTLN → Whisper → Text
* Improves recognition accuracy in noisy environments
* Based on [DTLN paper](https://arxiv.org/pdf/2005.07551.pdf)

**2.7 Test Audio Files**

* Files like jfk.wav, english\_test1.wav are included for demonstration and testing

**3. Model Characteristics**

* **Model**: Whisper-tiny, smallest in Whisper family (~39M parameters)
* **Architecture**: Encoder-Decoder Transformer
  + Encoder: Converts audio features (mel-spectrograms) into latent representations
  + Decoder: Autoregressively generates token ID output
* **Output Format**: Token sequence decoded into readable text using vocabulary files

**4. Speech Recognition Pipeline**

**4.1 Input Preprocessing**

1. Capture audio from file or mic
2. Resample to 16kHz, convert to mono 16-bit PCM
3. Normalize waveform amplitudes
4. Segment into 25ms windows with 10ms hop length (400 samples per window, 160 sample shift at 16kHz)

🔁 **Example: From Raw Audio to Mel-Spectrogram**

* Audio Length: ~500ms
* Sample Rate: 16kHz → 8000 total samples
* Window Size: 25ms = 400 samples
* Hop Length: 10ms = 160 samples

Sliding windows:

Window 1: [0-399] → ----window1----

Window 2: [160-559] → ----window2----

Window 3: [320-719] → ----window3----

Window 4: [480-879] → ----window4----

...

Matrix Shape: [48 × 400]

* Rows: 48 (number of windows)
* Columns: 400 (samples per window)

**4.2 Feature Extraction**

1. Apply Short-Time Fourier Transform (STFT) to each window to convert time-domain to frequency-domain
   * Human speech recognition relies more on frequency characteristics than on waveform shapes
2. Apply Mel-filterbanks:

Better matches human hearing perception

Reduces data dimensionality while preserving important features

* Typically 80 filters per frame （window)
* Each 25ms window (400 samples) is transformed into 80 values
* Example (simplified for illustration):
  + Filter 1: average bins 1–10 → Mel band 1 = -2.0
  + Filter 2: average bins 1–18 → Mel band 2 = -1.5
  + Filter 3: average bins 15–30 → Mel band 3 = -0.8
  + Filter 10: average bins 320–400 → Mel band 10 = -1.2

1. Generate 80×T mel-spectrogram matrix representing energy in each band over time （T: number of windows）

**4.3 Inference**

1. Feed spectrogram into encoder
2. Decoder autoregressively generates token IDs
3. Vocabulary .bin file maps token IDs to actual words

**Token Decoding Example** Using the vocabulary file, each token ID is mapped to a word or control token:

50257 → <start>

50362 → <en>

287 → "hello"

1917 → "world"

50256 → <end>

Final output: **"Hello world"**

**4.4 Postprocessing**

1. Assemble tokens into complete sentences
2. Apply punctuation and casing
3. Display final transcribed text

**5. Optimization & Efficiency**

* **Quantization**:

- Int8 quantization for weights

- Float32 for activations

* **Tuned Parameters**:
  + Default: 25ms window, 10ms hop, 80 filters
  + Optimized: 20ms window, 15ms hop, 64 filters
  + Reduces frame count and filter complexity for faster runtime
* **int8 weights**: Minimize model footprint
* **float32 activations**: Maintain accuracy

**6. APK Contents and Deployment**

* APK includes:
  + Java & C++ code
  + TFLite models
  + Vocabulary .bin files