ChatGPT

Debugging & Improving Pixo's Features

To address the issues, we must first **ensure the AI pipeline works** (LLM responses and speech output), then **enhance expressiveness** (more layered emotes) and **fix configuration wiring**. We'll tackle each in turn, citing relevant Apple and third-party documentation.

1. Verify and Log LLMService Responses

If **no response text or audio** is produced, the problem likely lies in the LLMService/network layer. To debug:

- **Add detailed logging:** In `LLMService.swift`, print out the HTTP status and full response body in the completion handler. For example, after receiving `data`, log `response.statusCode` and the raw JSON. The Groq documentation notes that successful requests return "200 OK" [1]. If you're seeing a different status (400, 401, etc.), the logs will reveal why.
- **Handle errors explicitly:** When using `URLSession` or similar, capture any error in the callback and `print(error.localizedDescription)`. For example, Groq's docs mention that error responses include a JSON with details [2]. Logging that JSON can reveal if the request JSON is malformed or the key is invalid.
- **Ensure UI updates on main thread:** When LLMService gets a text result, update SwiftUI state on the main queue. For example:

```
DispatchQueue.main.async {
    self.speechViewModel.currentResponse = responseText
}
```

  Failing to do so can prevent the text from appearing in the UI.
- **Check fallback to Apple TTS:** If ElevenLabs fails, ensure SpeechService calls `AVSpeechSynthesizer`. Log whether the ElevenLabs call succeeded or fell back, and any errors encountered during playback.

These steps (adding `print` or logging and checking HTTP status) are standard practice for debugging API calls [2] [1]. Once logged, you can identify if the issue is a 401 unauthorized (missing API key) or network failure, and fix it accordingly.

2. Confirm ElevenLabs Audio Playback and AVAudioSession

Even with correct text, no sound can mean an AV audio setup issue. To fix this:

- **Activate AVAudioSession:** Before playing back audio buffers, call:

```
try AVAudioSession.sharedInstance().setCategory(.playback, mode: .default,
options: [.allowBluetooth, .defaultToSpeaker])
try AVAudioSession.sharedInstance().setActive(true)
```

This ensures the app can output sound. The example tutorial confirms setting the `.playback` category and activating the session before playing audio [3].

- **Check audio engine start:** If using `AVAudioEngine`, ensure `engine.start()` is called and succeeded (catch any errors). For streaming audio, schedule buffers on `AVAudioPlayerNode` and call `playerNode.play()` only after the engine is running.
- **Volume and mute:** Ensure the device isn't muted and the app's audio is not being silenced by a background audio policy.
- **Verify data handling:** If ElevenLabs returns streaming audio, you must append chunks to a buffer and schedule them on the audio engine (similar to how streaming is handled in the referenced tutorial [4] [3]). If using a simpler playback, ensure the `AVAudioPCMBuffer` is created correctly with valid data.

By following these steps, you activate the iOS audio path properly. As shown in the streaming example, forgetting to set the session active causes no playback [3].

3. Overlay Stack for Layered Emotes (Petting)

To make Pixo **more "cute" with multiple emotes** during petting, use a **ZStack** of overlay views:

- **Use ZStack layering:** SwiftUI's `ZStack` lets you place multiple views (like heart sparkles, blush, etc.) on top of the face. For example:

```
ZStack {
  CompanionFaceView(…)
  if isPetting {
    BlushOverlayView()
    HeartSparklesView()
    ...
  }
}
```

A tutorial notes that `ZStack` "arranges views on top of each other along the Z-axis" [5]. You can stack 2–3 overlay shapes (hearts, blush gradient, sparkles) without cluttering the UI.

- **Timed, layered triggers:** When petting (drag gesture detected), trigger multiple overlays in sequence (e.g. blush appears first, then hearts, then sparkles). Each overlay can be a SwiftUI `Shape` or simple `View` with `opacity` and `scale` animations. For instance, show `HeartView()` with a fade-in, then after 0.1s add `SparkleView()`.
- **Capping overlays:** Limit to ~3 simultaneous overlays so it stays legible. Stack them with different offsets or alignment within the ZStack.

This approach uses SwiftUI layering principles [5] to create rich, combined effects (blush + hearts + sparkles) when the user pets Pixo. It avoids adding new on-screen buttons and maintains minimal UI, yet greatly increases "cuteness."

4. Wire Settings Toggles to Code

The settings toggles must actually **affect behavior**:

- **Play features toggle:** In your code (e.g. `IntentRouter` or `PlayModeManager`), check `UserDefaults.standard.bool(forKey: "pixo_play_enabled")` (or use `@AppStorage("pixo_play_enabled")`) before starting any play mode. If it's false, ignore "Dance", "Copy Me", etc. calls. This connects the UI toggle to the logic.
- **Silent mode toggle:** Similarly, check a flag like `isSilentMode = UserDefaults.standard.bool(forKey: "pixo_silent_mode")` before any `TTS` output. If `isSilentMode` is true, skip the ElevenLabs or AVSpeech calls. This ensures Pixo truly goes quiet when the user asks.
- **Implementation tip:** Use `@AppStorage("pixo_play_enabled")` in a ViewModel or service to auto-read the value. On toggle change, the state updates automatically [6]. Then guard your action code with those flags.

By explicitly reading these toggles in the action handlers, the UI settings will govern the behavior, as intended. Without this, the toggles only change UserDefaults but never influence the logic.

5. Improve Vision Face Landmarks for CopyMe & Mood

The **Copy Me** and **Guess Mood** features can be much more accurate by using Vision's landmarks (instead of rough bbox heuristics):

- **VNDetectFaceLandmarksRequest:** After detecting a face, use a landmarks request to get the eyes, eyebrows, mouth contours, etc. [7]. The documentation shows you can detect features like `leftEye`, `rightEye`, `innerLips`, and `leftEyebrow` [7].
- **Determine smile and brows:** Compute a smile metric by comparing the distance between outer lips or angles of `leftOuterLips` vs `leftInnerLips`. Eyebrow tilt can be found from `leftEyebrow` points. For example, if `leftOuterLips` curve upward relative to `rightOuterLips`, it's a smile. You can also measure eye openness from `leftEye`.
- **Coordinate mapping:** Vision gives normalized points within the face bounding box [8]. To use them, transform those points into view coordinates (as the article explains [8]). Once you have real landmarks, "copy me" can literally map the user's eyebrow tilt and mouth shape to Pixo, rather than guessing from box aspect ratio.
- **Mood guess:** With actual smile and eye openness values, a simple rule can guess mood (e.g. wide eyes + upturned lips = happy). This will be much more reliable than using Pixo's own emotion state.

By leveraging Vision's detailed landmarks, both CopyMe and Mood Guess become grounded in real facial expressions. The demo article notes how to perform a landmarks request and access features like `leftEyebrow` and `outerLips` [7] [9].

References

- Apple AVAudioSession documentation (setting category and activating session) [3]
- Groq API error handling (HTTP status codes and error JSON) [2] [1]
- SwiftUI ZStack layering (stacking multiple views) [5]
- Apple Vision face landmarks (detecting eyebrows, lips, etc.) [7] [8]

These sources outline the techniques needed to debug and enhance Pixo's code as described above.

---

[1] [2] API Error Codes and Responses - GroqDocs
https://console.groq.com/docs/errors

[3] [4] Streaming Audio With AVAudioEngine - Haris Ali
https://www.syedharisali.com/articles/streaming-audio-with-avaudioengine/

[5] ZStacks and Overlays: Building Beautiful Layered UI in SwiftUI | by Mohamed Hamdouchi | Feb, 2025 | Medium
https://medium.com/@mhamdouchi/zstacks-and-overlays-building-beautiful-layered-ui-in-swiftui-a7beb1bc7e10

[6] SwiftUI best practice for using @AppStorage for settings
https://stackoverflow.com/questions/66308518/swiftui-best-practice-for-using-appstorage-for-settings-how-to-read-the-userde

[7] [8] [9] Detecting face landmarks with the Vision framework
https://www.createwithswift.com/detecting-face-landmarks-with-the-vision-framework/