

# Latent: Technical Architecture & Stack Overview

**Latent** is a privacy-first, on-device Conversation Intelligence Engine built primarily to analyze live negotiations, interviews, and pitches in real-time without ever sending audio or transcriptions to the cloud.

---

## Core Technology Stack

- **Application Framework:** React Native (TypeScript)
  - **Target Platforms:** Android & iOS
  - **State Management & UI:** React Hooks, Functional Components, Custom CSS-in-JS (React Native StyleSheets).
  - **Routing:** React Navigation (Stack Navigator)
  - **Local Storage:** AsyncStorage (for saving post-session analysis & settings offline)
- 

## On-Device AI: The RunAnywhere SDKs

The entire artificial intelligence pipeline is powered exclusively by the **RunAnywhere SDK**. This allows massive neural network models (LLMs, STT, TTS) to run locally on the mobile phone's CPU/GPU.

The following SDK modules are integrated into Latent:

1. **@runanywhere/core**
    - The foundational bridging layer that interfaces between React Native JavaScript and the underlying C++ native environment.
  2. **@runanywhere/onnx**
    - The ONNX Runtime backend. This powers the **Speech-to-Text (STT)** engine using the *Sherpa Whisper Tiny* model.
    - Also acts as the execution engine for the **Text-to-Speech (TTS)** pipeline using the *Piper TTS* model.
  3. **@runanywhere/llamacpp**
    - The Llama.cpp backend used for running full **Large Language Models (LLMs)** on-device, such as *Liquid LFM2 (350M)* or *SmolLM2 (360M)*, without cloud latency.
- 

## Data Transmission & Input Flow

The most critical architectural pillar of Latent is the **Zero-Cloud Data Policy**. Here is how data is inputted, transmitted, and processed entirely within the phone's memory.

### 1. Audio Acquisition (Input)

- The application requests local microphone permissions ( `android.permission.RECORD_AUDIO` ).
- Audio is captured via native audio modules ( `react-native-live-audio-stream` / underlying C++ audio capture buffers).
- **Transmission:** Audio data is passed directly from the microphone hardware into local RAM (Float32 buffers or temporary WAV snapshots). **It never leaves the device.**

### 2. Live Transcription Pipeline (STT)

- The `SpeechService.ts` module routinely chunks the incoming audio (e.g., every 1.5 to 5 seconds).

- These localized audio snapshots are handed to the `RunAnywhere.transcribeFile()` C++ bridge.
- The **Sherpa Whisper Tiny ONNX** model analyzes the audio purely on local hardware. We inject a *Domain Bias Prompt* natively to optimize the STT context for words involving finance, startup, and negotiation (e.g., prioritizing "salary" over "celery").
- The C++ bridge returns the recognized text string back up to the JavaScript layer.

### 3. Contextual Auto-Correction & Parsing

- Upon receiving the raw text string, Latent passes it through a zero-latency, synchronous TypeScript `WhisperAutoCorrector` .
- This uses **Levenshtein Distance** algorithms to fuzzy-match phonetic hallucinations against the active *Negotiation Mode* (e.g., Job Interview vs. Investor Meeting) vocabulary.

### 4. Intent & Counter-Strategy Generation

- The sanitized transcript string is injected into the `SessionEngine` , which feeds it to the `IntentClassifier` .
- The engine evaluates the sentence against an offline `Universal Scoring Engine` (powered by structural pattern matching, negation rules, and numeric markers).
- If a specific tactic is detected (e.g., *Authority Pressure* or *Budget Objection*), the `CounterStrategyEngine` instantly surfaces actionable suggestions on the React Native UI.

### 5. Final Output & Offline Storage

- When a session ends, the full dialog map and cognitive metrics are compiled.
- Data is saved directly to local flash storage via `AsyncStorage` .
- The data is then displayed on the *Outcome Replay Screen* for post-meeting analysis, remaining 100% private to the physical device owner.