Hybrid Mode Architecture

Figure 1: Hybrid Mode Architecture

Hybrid Mode (LiveKit + Custom WS)

Overview

Hybrid Mode is a "Legacy" or "Transitional" architecture that demonstrates how to combine **LiveKit's Room Presence** features with a **Custom WebSocket Data Pipeline**.

In this mode, the application connects to a LiveKit room to establish user presence (and potentially receive video/chats), but **bypasses LiveKit for audio transmission**, instead creating a separate, parallel WebSocket connection to sends audio blobs directly to the backend.

End-to-End Architecture

System Flow

1. **Split Routing:** The user device creates two simultaneous connections.
2. **Path A (Signaling Plane):** Connects to **Cloud Node** (LiveKit). This handles "Who is here?", "Is the room active?", and "Permission Tokens".
3. **Path B (Data Plane):** Connects to **Private Server**. This carries the heavy audio payload.
4. **Gateway:** The Frontend creates a logical binding between these two paths implicitly.

Micro-Interaction Walkthrough

1. Dual Handshake

- **Step 1:** The app requests a LiveKit Token from the backend.
- **Step 2:** It simultaneously connects to `wss://livekit.cloud...` (Signaling) AND `wss://myprivateserver.../ws` (Data).
- **Visual:** The "Online" indicator lights up (Signaling), while the "Mic Ready" indicator waits for the second socket.

2. The Split (User Speaks)

- **Signaling:** LiveKit receives "User is Speaking" metadata (Active Speaker bit) via proper API calls, updating the visualizer for *others* in the room.
- **Data:** The *actual* audio bytes are routed *away* from LiveKit, down the orange "Data Path" to the private server. `typescript // In code:`

```
liveKitRoom.localParticipant.isSpeaking = true; // Path A
webSocket.send(audioBlob);                      // Path B
```

3. **Synchronization**

- The system relies on the User ID being consistent across both paths.
- If the Signalling path drops, the User appears "Offline" even if Audio is still streaming to the server.
- If the Data path drops, the User appears "Online" but is silent.

Performance Characteristics

Metric	Value	Notes
Latency	**Medium (~500ms)**	Similar to Direct Mode; limited by MediaRecorder chunking.
Complexity	**High**	Maintaining two independent connections (Room + WS) increases state management overhead.
Use Case	**Migration**	Useful when migrating from a legacy WS app to LiveKit, or when custom audio processing is needed outside WebRTC constraints.

Key Code References

- **Frontend Hook:** `frontend/src/hooks/useLiveKit.ts`
  - Demonstrates the unique `MediaRecorder` + `Room` combination.
- **Comparison:** Unlike Agent Mode, `useLiveKit.ts` handles the audio data **manually** via `ws.send()`.