Here’s the exact, wire-by-wire flow for your stack (Deepgram STT, ElevenLabs TTS, FastAPI WS conversation) with the Analytics Worker. I’ll show the live path, what each component sends/receives, and where state updates happen.

# **1) Live call loop (client ⇄ FastAPI ⇄ Deepgram ⇄ ElevenLabs)**

### **A. Client → FastAPI (conversation WebSocket)**

1. Browser opens wss://…/ws/convo/{call\_id}.
2. Client streams mic frames (Opus/PCM) to FastAPI over this WS.

**Message:** raw audio bytes (binary WS frames).

### **B. FastAPI → Deepgram (STT WebSocket)**

1. FastAPI opens Deepgram Streaming WS wss://api.deepgram.com/v1/listen?....
2. For each mic frame from client, FastAPI forwards it to Deepgram.

**Message:** raw audio bytes → Deepgram.

### **C. Deepgram → FastAPI (transcripts)**

1. Deepgram sends JSON messages containing partial and final transcripts.
2. FastAPI reads each message; when is\_final == true, it:  
   * (i) passes the text to your **bot logic** (for NLU/LLM planning), and
   * (ii) **tees** a TranscriptEvent into the **Analytics Worker** (in-process queue or out-of-process stream).

**Message to bot & analytics (your contract):**

{

"event\_type":"transcript.v1","call\_id":"c\_123","turn\_id":45,

"speaker":"customer","text":"I want to raise a ticket",

"start\_ms":58234,"end\_ms":60410,"final":true,"asr\_confidence":0.92

}

### **D. Bot response → ElevenLabs (TTS) → Client**

1. Your bot decides a reply (LLM or rules).
2. FastAPI sends reply text to ElevenLabs TTS (HTTP/WS) and receives an audio stream.
3. FastAPI streams that TTS audio **back to the same conversation WS** to the browser for playback.

**Messages:**

* To ElevenLabs: { "text": "Sure, I can open a ticket…" }
* To Client: audio bytes (binary WS frames).

Up to here is your existing convo loop. Now the analytics side-chain:

# **2) Analytics side-chain (fed only by Deepgram finals)**

### **E. FastAPI → Analytics Worker (two deployment styles)**

* **In-process:** asyncio.Queue[TranscriptEvent]  
   await queue.put(ev) inside your convo handler.
* **Separate service:** publish TranscriptEvent to a stream (Redis Streams/Kinesis).  
   The analytics service consumes, computes, and pushes updates to dashboards.

Either way, the worker receives the **same** TranscriptEvent.

### **F. Analytics Worker: per-utterance inference**

1. On each final transcript, the worker:

* Runs **sentiment** (tiny model/Comprehend/SageMaker) → numeric score in **[-1, 1]**.
* Detects **intent/entities** (optional now, easy later).
* Checks **ticket detection** (keyword/model).
* Updates **per-call keyed state** (rolling EMA, last-N window mean, ticket\_count, bullets).

**Internal state (per call\_id):** ema, window\_scores[], ticket\_count, summary\_bullets[], phase, interruptions, hold\_time\_ms.

### **G. Analytics Worker → Live snapshot**

1. Throttled (e.g., every 250 ms) the worker emits a **live snapshot**:

{

"event\_type":"agg.update.v1",

"call\_id":"c\_123",

"rolling\_sentiment":{"ema":-0.31,"win\_mean":-0.42},

"ticket\_count":1,

"phase":"diagnosis",

"hold\_time\_ms":12000,

"interruptions":{"agent":2,"customer":1},

"summary\_bullets":[

"Ticket requested for modem drops",

"Agent verifying account",

"Line test underway"

],

"updated\_at":"2025-08-14T10:24:11Z"

}

### **H. Delivering the snapshot to the UI**

You have two push routes (can use both):

* **Direct WS fan-out:** the worker calls push\_live(call\_id, snapshot) → FastAPI maintains /ws/live/{call\_id} rooms and pushes JSON to connected dashboards.
* **Persist + pull/push:** the worker writes the snapshot to **DynamoDB (PK=call\_id, SK=“live”)** with updated\_at; FastAPI either:  
  + streams via **SSE** by polling Dynamo every 500 ms, or
  + subscribes to a **Redis pub/sub** and fans out to /ws/live/{call\_id}.

**Message:** the JSON above to all dashboards watching that call\_id.

# **3) Post-call finalization (after hangup)**

1. Your convo WS detects end of call (client disconnects / telephony event).
2. FastAPI (or EventBridge/Step Functions) triggers a **finalizer** that:

* Collects **salient turns** from the worker (top-K bullets / key moments).
* Calls **Bedrock/LLM** once with transcript snippets + metrics to produce a high-quality final summary & compliance.
* Writes a **CallRecord** with SK="final" to DynamoDB and archives raw data to S3 (Parquet).
* Optionally updates CRM with ticket IDs/notes.

**Stored record (example):**

{

"call\_id":"c\_123","sk":"final",

"summary":"Customer reported modem drops. Ticket SR-5512 opened…",

"outcomes":["ticket\_opened"],"ticket\_ids":["SR-5512"],

"metrics":{"aht\_ms":459000,"silence\_ratio":0.12}

}

## **End-to-end sequence (compact diagram)**

Client → FastAPI /ws/convo → Deepgram

│ audio frames ───────────────▶ │

│ ◀────── partial/final transcripts ────

│

│ (bot reply text) → ElevenLabs TTS → audio ▶ back to client

│

└─(on each FINAL)→ Analytics Worker ingest

│ sentiment + intent + ticket

│ update per-call state (EMA, bullets)

└─ every ~250ms → agg.update.v1

FastAPI /ws/live/{call\_id} ◀──────── push snapshot ───────── Analytics Worker

Next.js dashboard ◀────────────────────────────────────────── FastAPI

## **What runs where (clear boundaries)**

* **FastAPI (conversation WS):**
  + Bridges client audio ⇄ Deepgram.
  + Runs your bot loop (LLM) and ElevenLabs TTS.
  + On Deepgram **finals**, publishes TranscriptEvent.
* **Analytics Worker:**
  + Consumes TranscriptEvent.
  + Maintains per-call state & computes metrics.
  + Emits throttled agg.update.v1 to UI (and optionally persists to DynamoDB).
* **FastAPI (live WS / SSE):**
  + Hosts /ws/live/{call\_id} (dashboards subscribe).
  + Pushes each agg.update.v1 to dashboards.
* **Finalizer (HTTP task / Step Functions):**
  + Runs once on call end to produce a rich CallRecord and store it.

## **Backpressure & failure behavior**

* **Never block** the convo loop: only send **final** transcripts to analytics; use Queue(maxsize=…) and drop oldest if overwhelmed.
* **WS disconnects:** dashboard clients can reconnect; your worker continues; snapshots are stateless.
* **Crash tolerance:** if worker restarts, it rebuilds state from subsequent finals; optional: also persist rolling state in DynamoDB if you need continuity.

That’s the whole pipeline: audio in → Deepgram finals → analytics side-chain → live snapshots to dashboards → post-call finalization. If you want, I can slot the queue.put() and push\_live() into your current FastAPI WS handler function with your Deepgram message shape.