How it works

The system is a voice bot that places outbound calls to a test clinic number as the patient. Twilio initiates the call and, when the line answers, connects it to our server over a bidirectional WebSocket media stream. We receive 8 kHz mulaw audio from the clinic, buffer about 1.5–2 seconds of it, then run Whisper for speech-to-text. That text is fed into a patient LLM (GPT-4o-mini) that's conditioned on a scenario example "schedule new appointment" or "refill prescription" plus the conversation history, and it returns the next thing the patient should say. We synthesize that with OpenAI TTS, convert the result to 8 kHz mulaw, and send it back over the WebSocket so Twilio plays it to the other party. That loop (buffer → STT → LLM → TTS → send) continues for the duration of the call. We log both sides of the conversation and save transcripts to disk—incrementally during the call and again when the stream ends—so we can review calls and run a separate bug-analysis step over the transcripts.

Why I made the main design choices

I used Twilio plus a WebSocket media stream so we own the full pipeline (STT, LLM, TTS) and can change models or add logic without touching Twilio's telephony config. The bot is scenario-based: each run gets a scenario ID (schedule, reschedule, cancel, refill, office hours,...) with a goal and optional first utterance, so we can systematically test how the clinic's IVR and staff handle different intentions. I chose batch STT on ~2 seconds of audio instead of streaming STT to keep the implementation simple and reduce the chance of the bot talking over the agent; we accept a bit of latency for more predictable turn-taking. Transcripts are written during the call as well as on hangup so that if the process is killed mid-call we still have the conversation so far for debugging and analysis. Patient identity (name and DOB) is fixed in config so every call presents the same patient to the clinic and we can consistently test how they handle that information.