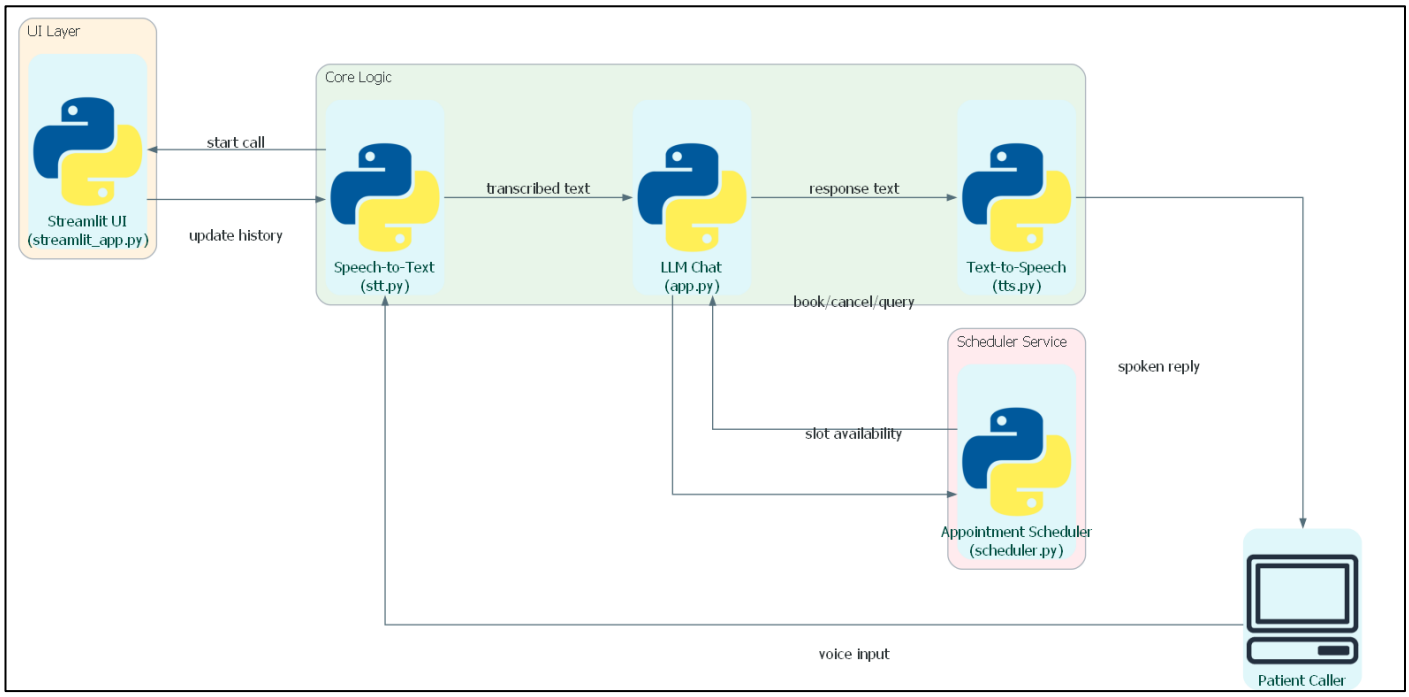
System Design Document – Confido AI Frontdesk Assistant

1. Architecture Overview



Architecture Diagram

This AI application is composed of the following integrated components:

- **Streamlit UI**: Provides an interactive web interface to simulate voice calls. It manages session state, handles button clicks, and renders the conversation flow.
- **STT (Speech-to-Text)** – stt.py: Uses the speech_recognition library with Google's speech API to capture and transcribe user speech into text.
- **LLM (Gemini 2.5)** – app.py: Interprets user intent using a system prompt and structured prompting. Communicates with the Google Generative AI API.
- **Scheduler Logic** – scheduler.py: Manages internal appointment slots, handles booking, rescheduling, cancellations, and availability queries.
- **TTS (Text-to-Speech)** – tts.py: Uses Google Cloud TTS and pydub to convert AI responses to spoken audio.
- The flow begins when the user initiates a call from the Streamlit UI. Audio input is passed to STT, the text is analyzed by the LLM, logic is executed if needed, and responses are spoken via TTS. This cycle continues until the user ends the call.

2. Tech Stack and Tools

S.No	Component	Tool/Library	Reason
1	Web Interface	Streamlit	Prototyping
2	STT	speech_recognition + Google STT	Accurate real-time speech transcription with fallback error handling
3	LLM	google-generativeai	Access to Gemini 2.5 LLM with structured prompting and fast responses
4	Prompt Mgmt	system_prompt.txt	System behavior logic for LLM guidance
5	Scheduler Logic	Python	Booking logic and simulating state tracking
6	TTS	Google Cloud TTS + pydub	

3. Prompt Engineering

To guide the LLM effectively, I designed a structured system prompt that defined:

- Exactly four actionable intents (schedule, reschedule, cancel, slots)
- A strict JSON response format
- Clear rules around memory, slot collection, and follow-up behavior.

This helped ensure the assistant:

- Collected only the missing information,
- Didn't re-ask for known details,
- Responded with structured JSON when ready to execute, and
- Stayed within scope (no medical advice or unrelated replies).

I also included natural language examples to show the desired flow. These steps made the LLM outputs predictable and easier to parse in code.

1. **Problem**: The LLM used to re-ask for information that was already provided (e.g. asking for the name again).
   Fix: Added this rule — *"Ask only for information that is missing. Do not re-ask for information unless the caller changes it."*
2. **Problem**: LLM responded with inconsistent intent for follow-ups (e.g. using schedule when rescheduling).
   Fix: Clarified intent logic with — *"Use 'reschedule' only after an appointment has been confirmed; otherwise continue using 'schedule' when offering alternative times."*
3. **Problem**: LLM sometimes responded in 24-hour time even outside of JSON, confusing the user.
   Fix: Added this rule — *"Outside of JSON, speak times in 12-hour format with AM/PM."*

4. Challenges Faced

- Gemini occasionally repeated already collected details.

- Fixed by enforcing the prompt to remember known values and only ask for missing ones.
- While building the assistant, I needed to make sure I always collected four pieces of information (slots) from the caller, name, date, time, and reason—before actually booking or modifying an appointment. In early tests, the LLM sometimes:
  1. Skipped slots (e.g. gave date and time but never asked for the name)
  2. Re-asked slots it had already confirmed, or
  3. Wrapped its JSON in extra text or code fences so our parser couldn't pull out the fields.

  To fix this, I built a strict slot mechanism in code:
- JSON-only responses: I wrote a regex in llm_turn to strip out exactly one JSON object and parse it into a Python dict of slots.
- Missing-slot prompts: After each LLM reply, I checked which keys (name, date, time, reason) are still empty and ask *only* for the first missing one , "May I have your full name?", "What date would you like?".
- Context carry-forward: For rescheduling or follow-up slots, I kept a current_booking object so if the user says "move it to 2 PM," AI already knows the date and doesn't ask again.
  These changes ensured the assistant reliably gathers all required fields in order, never repeats itself, and always has a clean JSON payload to drive the scheduler logic.

Initially, I used Gemini 1.5 Flash, which worked well for simple queries like booking an appointment. However, it struggled with more complex queries like rescheduling or cancellation. Switching to Gemini 2.5 resolved this, as it handled context, memory, and slot extraction more reliably.

Assumptions & Limitations

- The system assumes all appointments are in the year 2025 if the user does not specify a year.
- It assumes a single active caller/session at a time (no multi-user or concurrency).
- There is no real calendar or backend integration — all bookings are stored in memory and reset on restart.
- No persistence of appointment history or user profiles across sessions.
- STT may fail for strong accents, background noise, or unusual names.

Future Improvements

- Connect to real-time calendars (e.g. Google Calendar) to persist and validate appointments.
- Add session tracking to support multiple users and retain context.
- Improve STT reliability with confidence thresholds, spell correction, or Whisper-based fallback.
- Support multi-language interaction for both STT and TTS.
- Deploy using telephony APIs like Twilio or VAPI AI for live call support.