WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Yojana Seva Kendra: Real-Time Multilingual AI Voice Helpline for Indian Government Schemes

Saran B[a,*], Sanjay J[a], Sriram R[a], M Deepa[b]

[a]Department of Information Technology, Sri Shakthi Institute of Engineering and Technology, Coimbatore, Tamil Nadu, India. Register numbers: Saran B (714025205141), Sanjay J (714025205137), Sriram R (714025205168)

[b]Assistant Professor (Guide), Department of Information Technology, Sri Shakthi Institute of Engineering and Technology, Coimbatore, Tamil Nadu, India

A B S T R A C T :

Hundreds of government welfare schemes exist in India to support citizens across agriculture, health, housing, pensions, education, and small business. However, information about these schemes is scattered, difficult to find, and written in administrative language. Citizens who need them most often cannot get quick, clear answers. This paper introduces Yojana Seva Kendra, a real-time voice helpline that citizens can call or access online to ask scheme questions in English, Hindi, or Hinglish and receive grounded answers. The system combines LiveKit Agents running Gemini Live for natural conversation, a FastAPI backend for token management and logging, a React operator console, and a SIP trunk to reach people on ordinary phones. The critical design choice is that every scheme answer is anchored to a retrieval system: questions search a FAISS index over 25 schemes and 25 citizen-style examples, so answers stay tied to actual eligibility rules, amounts, and deadlines rather than invented details. Outbound calls follow an explicit state machine with answer detection and voicemail handling. The entire system runs on one machine, keeping deployment simple. We present the architecture, implementation, testing approach, and outline a path toward user evaluation.

Keywords: Voice AI; Retrieval-augmented generation; Government welfare schemes; Multilingual helpline; LiveKit; Gemini Live; SIP telephony; FAISS

Introduction

In India, there are hundreds of welfare programs managed by different government levels, each designed to support agriculture, healthcare, housing, pensions, education, and small business development. While these schemes exist to help citizens, the information about them is often scattered across government websites, PDF documents, and state-level notices written in administrative language. A farmer in Bihar trying to find out when the next installment of PM-KISAN will arrive should not have to navigate complex government portals just to get a straightforward answer. Similarly, traditional helplines come with their own limitations: they operate during fixed hours, maintain limited language support, and often struggle with call volumes. We saw an opportunity to address these gaps.

The recent advances in real-time speech models have made it possible to have natural spoken conversations with AI systems. However, directly using these models for government scheme information presents a serious risk. These models can confidently state outdated benefit amounts or make up eligibility rules that sound plausible but are incorrect. Since citizens might act on such information, we decided to build the system differently. We designed it so that the language model never answers from memory alone. Instead, every scheme question goes through a retrieval system, and responses are built from the actual data we retrieve. This approach keeps answers grounded and factual.

This paper describes Yojana Seva Kendra, a voice-based helpline that implements this retrieval-first approach. The system lets citizens speak naturally in English, Hindi, or Hinglish and get answers in their own language. Beyond handling incoming calls from citizens, it can also place outbound calls to reach people proactively. The system is designed to be observable and operable, giving us detailed logs of what the AI hears, retrieves, and says. We also built it to run on a single machine, keeping the infrastructure simple and cost-effective.

Our key contributions are these:

- A speech-to-speech helpline system where every scheme answer is grounded in a curated knowledge base of 25 central and state schemes, implemented using FAISS vector search over sentence embeddings.
- An outbound calling system that uses a defined state machine with answer detection and voicemail handling, ensuring the agent never speaks to a ringing phone or a voicemail box.
- An operator dashboard and backend control system that provides a unified log, live monitoring, and process supervision, all accessible through a single interface.

- A complete system design that prioritizes reliability and security while remaining deployable on minimal infrastructure.

The rest of this paper is organized as follows. Section 2 covers related work in helplines, voice systems, and retrieval-based generation. Section 3 presents the overall system architecture. Section 4 walks through the inbound call flow, the retrieval mechanism, and outbound calling. Sections 5 and 6 detail the monitoring and security aspects. Section 7 discusses our implementation and testing approach. Finally, Sections 8 and 9 discuss limitations and future directions.

Related Work

Government scheme portals. Platforms like MyScheme let citizens search for applicable schemes and check their eligibility. While valuable, they assume the user has literacy in English, internet access, and knows how to search for what they need. Our goal is to serve the opposite end: someone with just a basic phone who can simply ask a question out loud.

Traditional helplines and voice systems. Call-center based helplines rely on human staff, which means they have fixed hours, limited language options, and face queue challenges. Interactive voice response systems offer 24/7 availability but are rigid and menu-driven. Voice-based AI agents change this equation by offering natural conversation, instant availability, and the ability to switch languages mid-call. The challenge is making them reliable and accurate.

Retrieval-augmented generation. RAG combines a language model with a separate knowledge store that is searched at query time, rather than relying only on the model's training data. This is particularly valuable when answers need to stay current and precise. We use Sentence-BERT embeddings and FAISS indexing to efficiently find relevant passages from our scheme documents when a citizen asks a question.

Real-time communication and telephony. WebRTC handles audio between browsers and servers in real time. LiveKit builds on WebRTC to create rooms and relay media. SIP is the standard protocol for phone calls, and tools like Vobiz let us reach regular mobile phones. Our contribution is integrating these pieces into a grounded, observable, and operational helpline system.

System Architecture

The system has four main components, as shown in Figure 1. The first three run as separate processes on the same machine, all communicating over localhost. The fourth component reaches out to the public telephone network.

1.1. Voice agent worker

This is a LiveKit Agents worker that runs the actual conversation. When a citizen starts a session, the worker joins a LiveKit room, starts a session with Gemini Live (the realtime speech model), and manages the dialogue. A separate, simpler text-to-speech model plays the opening greeting to ensure it is consistent. After that, the realtime model handles both understanding and speaking.

1.2. Control plane

This FastAPI backend is the central coordinator. It issues short-lived authentication tokens to the console, collects all events and logs from the worker, streams those logs to the console over SSE, and supervises the worker process. It also handles process lifecycle (start, stop, restart) and serves as a gateway to the LiveKit API.

1.3. Operator console

This React application is what an operator uses. They can start voice sessions from the Monitoring tab, watch the live audio feed and agent state in real time, and tail the structured logs. There's a separate Outbound tab where they enter phone numbers and initiate calls to reach citizens proactively. The console treats the connected and idle states as two separate render trees for performance.

1.4. Telephony edge

A LiveKit SIP trunk connects to Vobiz, a CPaaS provider that bridges to real phone networks. This lets the same agent that answers citizens on the web also speak to people calling on ordinary mobile phones.

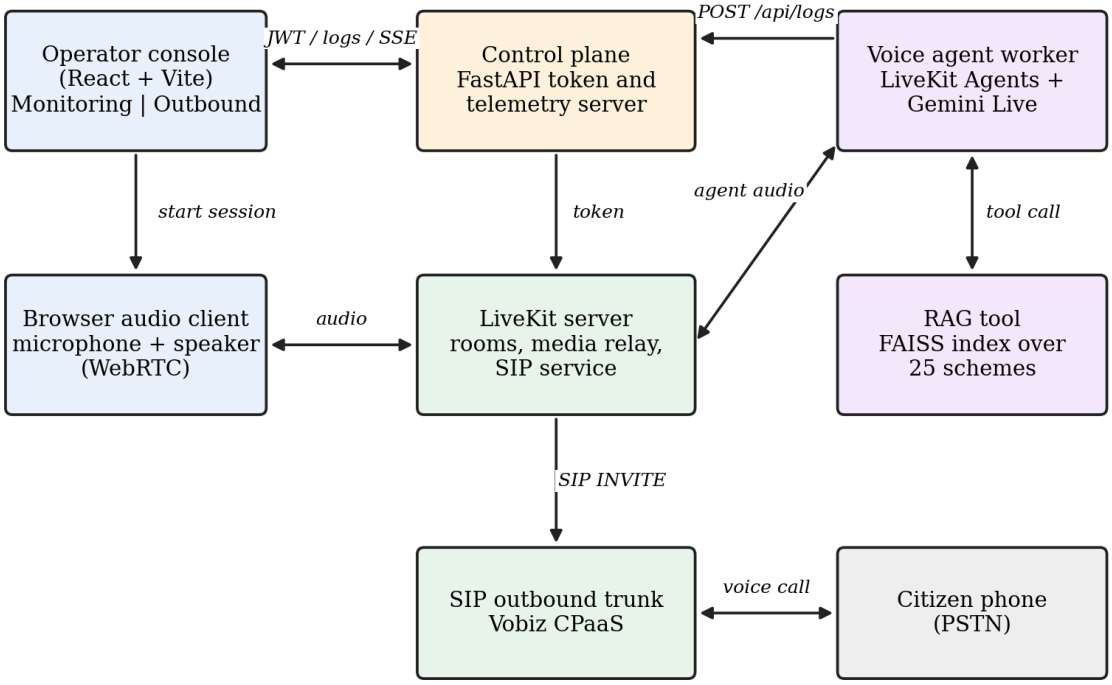


Fig. 1 - Overall system architecture showing the four main components and their relationships.

Table 1 - Core technologies and their roles.

Component	Technology & Version	Purpose
Speech model	Gemini Live (gemini-3.8-live); voice: Fenrir	Listens and speaks in real time with low latency
Opening line TTS	Gemini Flash TTS (gemini-3.8-flash-tts)	Ensures greeting is always the same
Call answering detect	Gemini Flash (gemini-3.5-flash)	Classifies if a human or machine answered
Agent framework	LiveKit Agents 1.8.8; plugin: livekit-plugins-google 1.8.8	Manages sessions and communication
Media transport	LiveKit server, client library 2.16.0, SIP trunk	WebRTC rooms and phone calls
Backend	FastAPI 0.143.2, Uvicorn 0.54.0, Pydantic 2.14.1	Tokens, logs, and supervision
Retrieval & search	FAISS 1.15.1, sentence-transformers 6.1.0	Finds relevant scheme information
Frontend UI	React 19, Vite 7, custom CSS	Operator dashboard
Testing	pytest 8.4.2, Playwright for browser tests	Automated quality checks

How It Works

1.5. When a citizen calls in

The process starts when someone clicks "Start Voice Chat" on the console. The console requests a token from the backend, connects to a LiveKit room using that token, and activates the microphone. At the same time, a worker process in the background connects to the same room and starts listening for audio. The worker sends an opening greeting through the pre-recorded TTS model, then hands off to Gemini Live. From that point on, citizen and agent have a natural spoken conversation.

One detail that took some debugging: the browser audio rendering must have a single owner, otherwise you get doubled audio or echo. We made the room audio renderer component the sole owner of playback. We also treat three things separately: whether the microphone is on, whether the agent is speaking or thinking, and whether the agent is actually present and responsive. This separation let us debug issues that would have been invisible if we mixed these signals together.

1.6. How answers stay accurate

Every time the citizen asks about a scheme, the agent calls a tool named lookup_government_scheme_info. This tool searches a FAISS index that we built from our knowledge base: 25 central and state schemes plus 25 example citizen questions. The search returns up to five results, filtering out anything below a similarity threshold of 1.35. The system prompt instructs the agent to ground its answer in the retrieved amounts, eligibility rules, and interest rates. It also forbids the agent from asking for sensitive information like Aadhaar passwords, bank PINs, or claiming to see a citizen's account status.

1.7. Calling citizens

An operator enters a 10-digit mobile number in the Outbound tab. The backend normalizes this to E.164 format and creates a record in the call registry. It then tells LiveKit to set up a SIP participant. The agent is automatically dispatched to a room named after the call ID, so it knows which call is being handled. Critically, the agent does not speak until it detects that the SIP call has been answered. This means the line stays silent while ringing, so if someone does not answer, they just hear the phone ring—no weird voicemail message from a silent agent.

Once answered, the agent runs a fast classification model to determine whether a human or machine answered. If it is a machine, the call is logged as voicemail and the agent hangs up. If it is a human, the agent speaks the greeting and launches into the scheme-based dialogue. The agent has 4 minutes 30 seconds to finish the conversation, after which it plays a closing message and ends the call.

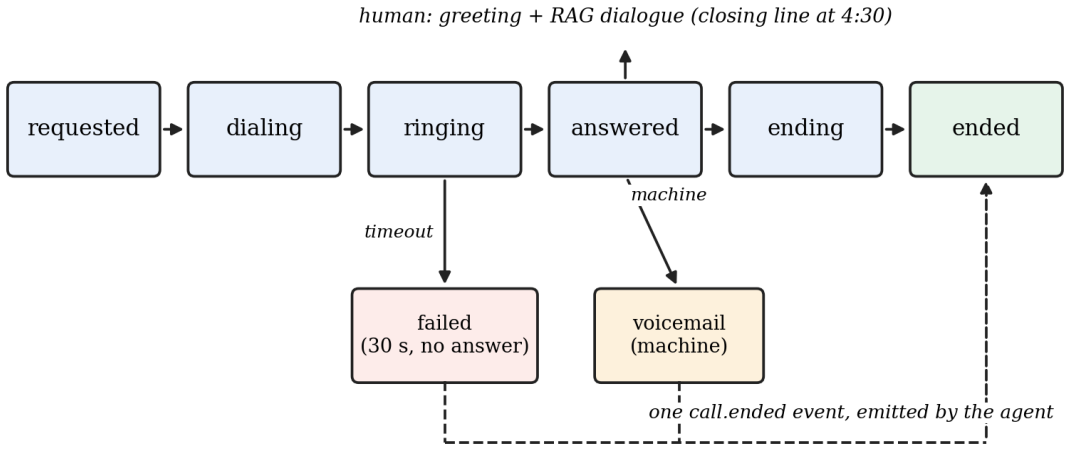


Fig. 2 - Outbound call lifecycle, from request through completion or failure.

Table 2 - Call parameters and timing.

Parameter	Behavior
Phone number	10-digit format, normalized to international E.164 (+91…)
Room identifier	call-{call_id}
Timeout if unanswered	30 seconds from initial INVITE
Call duration limit	270 seconds (4 min 30 sec) after agent greeting
Duration billing	Measured from INVITE to disconnect (includes ring time)
Talk time	Measured from answer to disconnect
Call end event	Exactly one per call, always sent by the agent
Operator hang-up	Available from console; second hang-up is idempotent

Observability and Control

1.8. A single log stream

Everything that happens gets written to a bounded in-memory log buffer. Each entry has a generated ID, a severity level, a channel (SYSTEM, WEBRTC, RAG, TTS, STT, CALL, TASKS), structured data, and correlation metadata. The agent cannot write to this buffer directly; instead, it posts events to the same /api/logs endpoint that the console reads. This creates a single source of truth. The console can tail the logs in real time, filter by channel or level, and search the text.

For outbound calls, we correlate by call_id rather than by room session ID. This means we can pull the full history of a phone call with a single API request: GET /api/logs?session_id=<call_id>. Before logging, we redact any sensitive information that might leak into the logs or be displayed on the operator dashboard.

1.9. Process monitoring

The backend can report on and control the three local processes: the token server, the agent worker, and the Vite development server. GET /api/tasks returns the PID and status of each. POST /api/tasks/{id}/control lets an operator start, stop, or restart them. This matters because the agent framework and the Gemini plugin must stay on matching versions, and pinning them in a virtual environment keeps them together.

Security and Reliability

Several design choices support security and prevent common mistakes:

- Authentication tokens are short-lived and created on the server, so the client never sees the API secret.
- CORS is restricted to loopback addresses by default, with an explicit allow-list in the environment for overrides.
- Credentials for LiveKit and Vobiz are kept in a .gitignore-d .env file and never bundled into the frontend.
- Real-time sessions fail loudly if there is a version mismatch or configuration problem, rather than silently reporting as "connected."
- The RAG embedding cache is versioned, so updating the knowledge base cannot silently reuse stale vectors.
- We deliberately refuse to dial if no worker is running, preventing silent calls to real phones.

Building and Testing

The system is developed and tested on Windows 10 and 11. Python runs in a virtual environment with all dependencies pinned in requirements.txt. The frontend uses npm. A PowerShell script opens the three console windows to start each process. Backend tests use pytest with markers for default tests, tests that need a live agent, and UI tests. Browser tests run in Playwright with Chromium, using a synthesized audio file so we can test voice sessions without actually speaking. We also have an intentional "canary" test: it checks that the app never touches the participant audioTracks object directly, because doing so throws an error in livekit-client 2.x and corrupts the React tree.

Discussion

1.10. What we hoped to achieve

Our goal is that a citizen with a phone and no English reading ability can get an accurate, specific answer about a government benefit in under a minute, in their own language, without opening a website or waiting in a queue. On the operator side, they should be able to see exactly what the agent heard, what it retrieved, and what it said. And we wanted to keep the infrastructure simple: one FAISS index, three local processes, and a SIP connection. A single machine can run the whole system.

Abbreviations

RAG – Retrieval-Augmented Generation
SIP – Session Initiation Protocol
PSTN – Public Switched Telephone Network
CPaaS – Communications Platform as a Service
AMD – Answering-Machine Detection
TTS – Text-to-Speech
SSE – Server-Sent Events
JWT – JSON Web Token

1.11. Limitations and next steps

Our knowledge base currently covers 25 schemes, so questions about schemes outside this set will not find a match and the agent will say so. The system depends on third-party services for the speech model and for the phone network connection, so we do not control latency or availability there. We deployed it as a single machine on Windows, not as a cloud service. Most importantly, we have not yet done a measured study with real citizens to evaluate answer accuracy, latency, and success rates. That is the most critical next step.

Conclusion

We built Yojana Seva Kendra to demonstrate that voice-based government scheme helplines can be grounded, observable, and operational. The key insight is that language models should not be trusted to recall facts about real government schemes; instead, every answer should come from an explicit retrieval step. This architecture is more work upfront but it is much safer when the stakes are high. Future work will grow the scheme corpus, add more Indian languages, run user studies to measure accuracy and latency, and explore cloud deployment.

Acknowledgements

We thank our guide M Deepa (Assistant Professor, Department of Information Technology, Sri Shakthi Institute of Engineering and Technology) for her mentorship and guidance throughout this work. We also thank the college for providing the resources and environment for this research.

REFERENCES :

- Alvestrand, H. (2021). *Overview: Real-time protocols for browser-based applications* (RFC 8825). Internet Engineering Task Force.
- Google. (n.d.). *Gemini API: Live API*. Retrieved from https://ai.google.dev/gemini-api/docs/live
- Johnson, J., Douze, M., & Jégou, H. (2021). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data, 7*(3), 535–547.
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems, 33*, 9459–9474.
- LiveKit. (n.d.). *LiveKit documentation*. Retrieved from https://docs.livekit.io
- Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of EMNLP-IJCNLP 2019* (pp. 3982–3992).
- Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., et al. (2002). *SIP: Session Initiation Protocol* (RFC 3261). Internet Engineering Task Force.
- Schulzrinne, H., Casner, S., Frederick, R., & Jacobson, V. (2003). {*RTP: A transport protocol for real-time applications*} (RFC 3550). Internet Engineering Task Force.
- van der Geer, J., Hanraads, J. A. J., & Lupton, R. A. (2000). The art of writing a scientific article. *Journal of Science Communication, 163*, 51–59.