AgoraAI: A Novel Open Source Voice-to-Voice Solution to Multiagent-Multihuman

Antonio Concha-Sánchez , Jose Bernal , Alfredo Hernandez-Muñiz , Suresh Kumar Gadi *

Posted Date: 30 October 2025

doi: 10.20944/preprints202510.2309.v1

Keywords: multi-agent systems; voice-to-voice communication; large language models (LLMs); human-ai collaboration; open-source software; gemini

 doi:10.20944/preprints202510.2309.v1

Article

AgoraAI: A Novel Open Source Voice-to-Voice Solution to Multiagent-Multihuman

Antonio Concha-Sánchez [1,‡], Jose Bernal [2,‡], Alfredo Hernandez-Muñiz [2,‡] and Suresh Kumar Gadi [2,3,‡*]

[1] Facultad de Ingeniería Mecánica y Eléctrica, Universidad de Colima, Coquimatlán Colima 28400, Mexico
[2] Departamento de Investigación y Desarrollo, Industria Sigrama S.A de C.V., Torreón, Coahuila 27020, Mexico
[3] Facultad de Ingeniería Mecánica y Eléctrica, Universidad Autónoma de Coahuila, Torreón, Coahuila 27276, Mexico
* Correspondence: Research@SKGadi.com; Tel.: +52-n449-182-0153
‡ All authors contributed equally to this work.

Abstract: This article presents AgoraIA, an open-source framework designed to enable dynamic, multi-participant conversations by integrating multiple AI personas within a shared conversational environment. Unlike traditional single-agent LLM interactions, AgoraIA allows users to configure and deploy distinct AI agents that can engage in simultaneous, turn-based dialogues—both with human participants and with each other—through voice or text interfaces. The system supports a wide range of use cases, including formal panel discussions, interactive educational settings involving teachers and students, and informal conversations with friends or family members. By enabling diverse AI perspectives, roles, and communication styles, AgoraIA facilitates rich, context-aware exchanges that enhance engagement and collaboration. Key innovations include: (1) a modular architecture for creating customizable AI personas with unique roles, knowledge bases, and interaction styles; (2) support for synchronized turn-taking in mixed human–AI groups, ensuring structured yet fluid conversations; and (3) scalability across domains such as education, research, and decision-making. The effectiveness of AgoraIA is demonstrated through examples of its application in various professional and academic conversations involving users of different ages and experience levels.

Keywords: multi-agent systems; voice-to-voice communication; large language models (LLMs); human-ai collaboration; open-source software; gemini

1. Introduction

The initial development of Artificial Intelligence (AI) relied on complex algorithms and extensive data pre-processing, requiring specialized skills in programming and data preparation that created a high barrier to entry for non-experts. The advent of Large Language Models (LLMs) democratized access to AI, enabling natural language interactions through a technique known as prompt engineering [1–3]. This shift allows users of all technical levels to effectively communicate with AI models without the need for data pre-processing marking a transition toward accessible, general-purpose AI systems [4,5]. Recent advancements such as GPT-4 [4] and LLama [5] further illustrate how foundation models have become the backbone of modern AI applications, fostering open innovation and rapid adoption across research, education, and creative domains [6,7].

This accessibility was further enhanced by the integration of voice-to-voice communication interfaces in consumer applications, making AI interactions more intuitive and expanding their reach [8,9]. Voice-based LLMs, such as SpeechGPT [8], Whisper [10], and VoiceLLM [11], have demonstrated that combining speech and language understanding leads to more natural, multimodal dialogue systems. Through prompt engineering, these models can be assigned distinct personalities or roles, such as a lawyer, teacher, or student, giving rise to the concept of AI "agents." Platforms like Character.ai exemplify this trend, focusing on creating engaging one-on-one conversations with a single, personality-driven AI agent, showcasing highly naturalistic voice outputs [12].

 doi:10.20944/preprints202510.2309.v1

Despite the remarkable progress in conversational systems based on LLMs, the vast majority of implementations—especially those involving voice—remain limited to single-agent architectures interacting with a single human user [13]. This paradigm poses inherent constraints in contexts that demand multiple perspectives, specialized roles, or dynamic group interactions, such as debates, panel discussions, collaborative learning environments, or decision-making scenarios [14]. Although multi-agent approaches are emerging [15], and educational settings have begun to explore multiple conversational-agent interlocutors [16], these remain the exception rather than the rule.

To address the need for more collaborative forms of human–AI interaction, several frameworks have emerged, such as CrewAI [17], ChatDev [18], AutoGen [19], MetaGPT [20], and ChatArena [21], which enable structured collaboration between multiple LLM-based agents. These frameworks have demonstrated the power of coordinated multi-agent systems in completing complex tasks, such as automating software development. Empirical evaluations, such as those presented in the MAgIC framework [22], further demonstrate how LLM-powered agents can exhibit cognitive and collaborative behaviors in controlled environments. Similarly, Park et al. [14] explored how generative agents can simulate human-like social behaviors and long-term memory formation in open environments, highlighting the emergence of autonomous and context-aware interactions among LLM-based entities. However, their design remains primarily focused on backend agent-to-agent communication, requiring programming expertise for setup and operation. Moreover, most of these systems are constrained to text-based interfaces, limiting accessibility and user immersion [14].

Recent works have emphasized the transition from single-agent conversational systems to collaborative, multi-agent frameworks that leverage Large Language Models (LLMs) to coordinate and reason collectively. Tran et al. [23] provide a comprehensive survey of collaboration mechanisms among LLM-based agents, identifying key dimensions such as cooperation, competition, coordination strategies, and orchestration architectures. In parallel, Park et al. [14] demonstrated how LLM-based agents can simulate complex social behaviors and exhibit human-like interactions in open environments, highlighting the emergence of theory-of-mind capabilities within multi-agent systems. Efforts have also focused on improving cooperation between dialogue agents in task-oriented contexts. For instance, Sun and Kou [24] proposed a multi-agent collaborative dialogue (MACD) algorithm based on reinforcement learning, which enhances coordination and joint policy learning among conversational agents, significantly improving task success rates in complex dialogue environments. This line of research underscores the potential of multi-agent collaboration for achieving more coherent and efficient conversational behavior in AI systems.

A significant gap therefore persists in the current landscape of AI communication. While existing platforms enable either one-to-one (single-agent/single-human) or many-to-one (multi-agent/single-human) interactions, there remains a lack of user-friendly systems that support dynamic, multi-participant, real-time, voice-based collaboration. Even recent advances in embodied multi-agent cooperation emphasize text-based coordination rather than naturalistic, voice-driven interaction [25].

To overcome these limitations, we introduce AgoraAI, an open-source platform that bridges this gap by enabling multi-agent, multi-human, voice-to-voice collaboration. AgoraAI allows users to configure and simultaneously deploy multiple AI agents with distinct personas, roles, and communication styles within a shared conversational space. This design aligns with recent studies that investigate theory of mind and role awareness in multi-agent LLM collaboration [26,27], while extending these cognitive principles to real-time, voice-based interactions involving multiple humans and AI participants. Comparable multi-agent systems have been explored in domain-specific applications—such as AI-driven language-learning environments [28]—yet these implementations remain limited to text-based exchanges and lack real-time, multi-human participation. In contrast, AgoraAI generalizes the concept to a fully open, voice-enabled, and extensible framework designed for diverse collaborative scenarios across education, research, and social communication.

Building upon the collaborative principles of frameworks such as ChatDev, AgoraAI advances these ideas toward fluid, context-rich, and multi-perspective dialogues, suitable for formal panels,

 doi:10.20944/preprints202510.2309.v1

interactive educational sessions, and informal group discussions. The main technical contributions of AgoraAI include:

1. **Multi-Agent Architecture:** Enables the simultaneous deployment of distinct AI personas within a single conversational event, fostering a diversity of viewpoints.
2. **Context-Aware Generation:** Agents generate responses based on a comprehensive and dynamically constructed prompt, ensuring interactions are coherent and contextually relevant.
3. **Comprehensive State Management:** The backend maintains a complete state of the simulated event, acting as a *single source of truth* to guarantee conversational consistency.
4. **Control Mechanisms and State Transition:** The platform includes features to transition from a setup state to a conversational state, as well as mechanisms to forcibly interrupt an agent's speech or trigger a response, granting users greater control over the dynamics.
5. **Transcript and Reporting Management:** An administrative interface allows for the generation of complete event reports (with details, roles, and history) and the management of transcripts, including downloading for backup and uploading to restore a previous conversation.
6. **Modern Full-Stack Technology:**
   - **Frontend:** Developed with Vue.js 3 and the Composition API, atop the Quasar framework for Progressive Web Applications (PWA), and using TypeScript for static typing.
   - **Backend:** Built on Node.js with Socket.IO for real-time, bidirectional communication and high scalability, also implemented in TypeScript. LLM integration is handled centrally.
7. **Simplified Deployment:** The platform is delivered as a complete solution, with Docker as the primary method for straightforward and consistent deployment.

In summary, AgoraAI contributes to the field of multi-agent systems by providing a robust, ready-to-use, open-source infrastructure that democratizes the creation of complex conversational simulations and diverse, AI-driven perspectives.

This article is structured as follows: Section 2 presents the AgoraAI interface and design, detailing both the admin and audience interfaces. Section 3 describes the overall system design, including the technological architecture and implementation decisions. Section 4 details case studies demonstrating the platform's application and effectiveness in various contexts. Section 5 discusses proposed enhancements and future directions for the platform. Finally, Section 6 provides the conclusions of the work. Additionally, Appendices A and B provide a detailed algorithmic description of the front-end and back-end components, respectively.

2. AgoraAI Interface

AgoraIA is an open-source platform for multi-agent simulation, offering real-time visualization of conversations, meeting assistance, and comprehensive reporting. The prerequisites for running AgoraIA are Docker installed on the system and a Google Generative AI API Key to utilize the LLM model. The source code is available in the GitHub repository [29], and the Docker image for easy installation is hosted on Docker Hub [30].

AgoraIA is designed to be operated by an administrator (referred to as the operator) responsible for configuring, managing, and facilitating the event. The application provides two access points: the Audience Interface (Index Page) for public display at http://localhost:3000/ and the Admin Interface for event control at http://localhost:3000/admin. Access to the admin page is secured by the environment variable `MY_PAI_ADMIN_PASSWORD`, with `123456` as the default password when none is specified. The Gemini API key is configured through the `MY_PAI_GEMINI_API_KEY` environment variable.

These settings, along with the port configuration, can be easily managed using Docker. The `docker-compose.yml` file shown in Listing 1 demonstrates a sample configuration of AgoraIA.

```
version: '3.8'
services:
```

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 October 2025 doi:10.20944/preprints202510.2309.v1


```yaml
  agora:
    image: skgadi/sigrama-agora-ai:latest
    container_name: temp-suresh-agora-ai
    restart: unless-stopped
    environment:
      MY_PAI_GEMINI_API_KEY: "AaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaA"
      MY_PAI_ADMIN_PASSWORD: "1234"
    ports:
      - "8080:3000"
```

Listing 1: `docker-compose.yml` example to configure docke container.

The Admin Interface empowers the operator to set up event parameters and manage the flow of conversation during the event. This interface operates in two distinct states: a Setup State for initial configuration and a Conversational State for real-time event execution.

During the Setup State shown in Figure 1, the operator prepares all necessary details for the event. This involves configuring three main categories of information: Event Details, Roles, and Participants, alongside managing system settings and data.

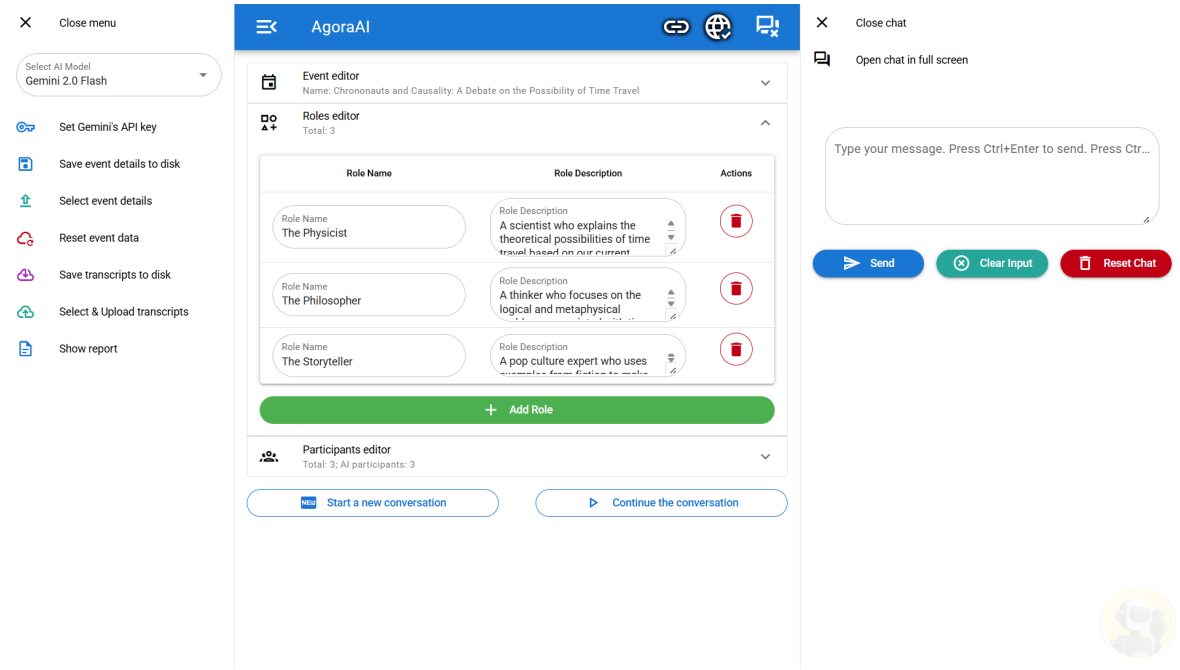

Figure 1. The Admin Interface in the Setup State. This view allows the operator to access various configuration editors (e.g., Event, Roles, Participants) and management functionalities (e.g., API setup, data backup/restore).

2.1. Configuration Editors

The operator utilizes the following editors, shown in Figure 1, to define the event structure:

- **Event Editor:** Allows the operator to define global event parameters, including the background image, event language, event name, event description, and event dynamics.
- **Roles Editor:** Facilitates the creation and definition of various roles within the event (e.g., moderator, Agora, audience). There is no predefined limit on the number of roles that can be created.
- **Participants Editor:** Enables the operator to add and configure all participants, which are listed in a table. The first column specifies whether the participant is human or AI, as shown in Figure 2. For a human participant, `Human` is selected. For an AI participant, a specific Text-to-Speech (TTS) voice must be chosen. Subsequent columns allow the operator to register the participant's name, assign a role (from those defined in the Roles Editor), and provide a biography. The final column is for uploading an avatar image for each participant. For AI participants, three distinct avatar

 doi:10.20944/preprints202510.2309.v1

images are required to represent their idle, thinking, and talking states, see Figure 2. Any standard image format is supported; using animated GIF formats can enhance the realism of the AI's visual representation during its thinking and talking phases. It is important to mention that for AI participants using a TTS voice identified as a non-native speaker of the event language, individual speaking turns are limited to a maximum duration of 15 s in order to preserve the flow of the conversation.

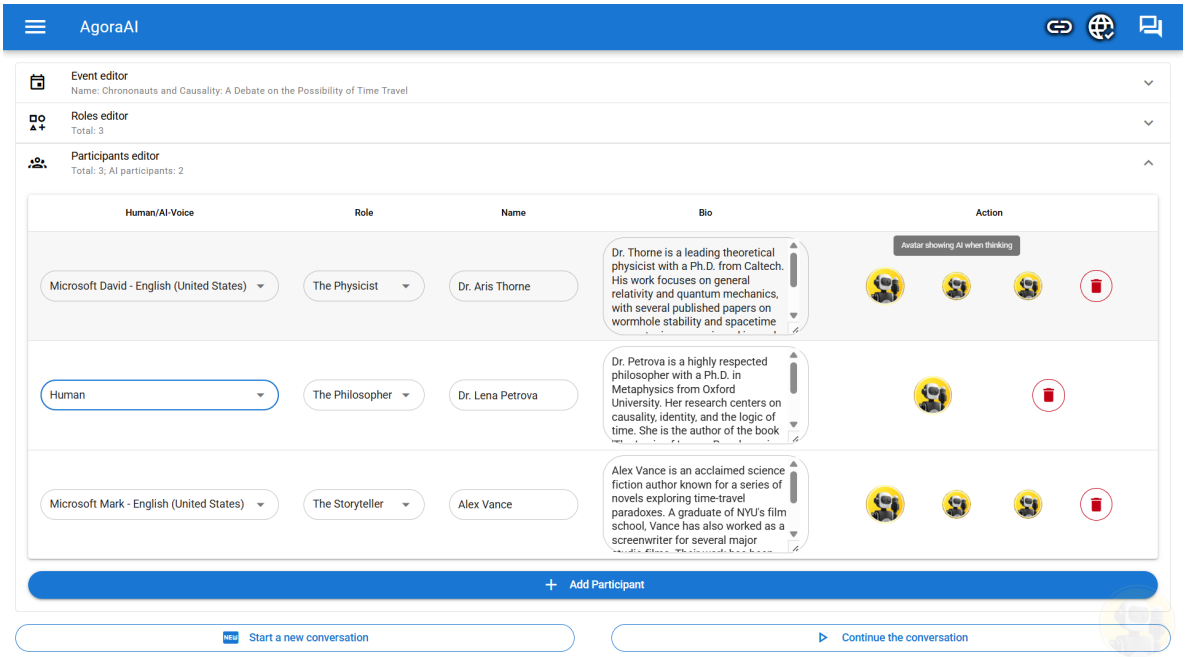

Figure 2. The Admin Interface in the Setup State with the `Participants editor` expanded.

2.1.1. Configuration Management and Data Operations

Beyond editing event details, the Setup State provides the following buttons, illustrated in Figure 1, for system and data management:

- **Select AI Model:** The application allows the user to select from a list of available AI models, which is updated automatically from Gemini. By default, the app uses Gemini 2.0 Flash. This flexibility enables users to choose the most suitable model for their specific task. It is also useful for maintaining a smooth conversation, as switching to a different model can resolve issues when one is busy or overloaded. Recommended models for AgoraAi include Gemini 2.0 Flash, Gemini 2.5 Flash, Gemini 2.0 Pro, and Gemini 2.5 Pro.
- **Set Gemini's API key:** Opens a dialog box for the operator to input the Gemini API key. This key is essential for the server to perform Speech-to-Text (STT) processing for human speakers and to obtain responses from the AI.
- **Save event details to disk:** Allows the operator to download all event configuration details, including images, as a single backup file.
- **Select event details:** Enables the operator to restore a previously downloaded event configuration. This action overwrites any current event details; thus, caution is advised.
- **Reset event data:** When a user edits an event, two versions exist: the original on the server and the new, edited version. This button discards all user changes and resets the form by reloading the original data from the server.
- **Save transcripts to disk:** Downloads the conversation history, comprising STT results from human speakers and AI-generated responses, in a raw format from the server. This serves as a backup of the event's dialogue.
- **Select & Upload transcripts:** Allows restoration of a previously downloaded transcript history. This action replaces the existing history and should be used with care.

 doi:10.20944/preprints202510.2309.v1

- **Show report:** Generates a JSON comprehensive file, human-readable report that includes event details, defined roles, participant information (including biographies), and the full transcript of the event.

2.1.2. Conversational State

Upon pressing the `Send data to AI` button in the Setup State (see the blue button on the right in Figure 1), the configured event data is transmitted to the server, triggering the Admin Interface to transition to the Conversational State (Figure 3). In this state, the interface displays all participants alongside speaker designation buttons, with visual indicators distinguishing human and AI participants.

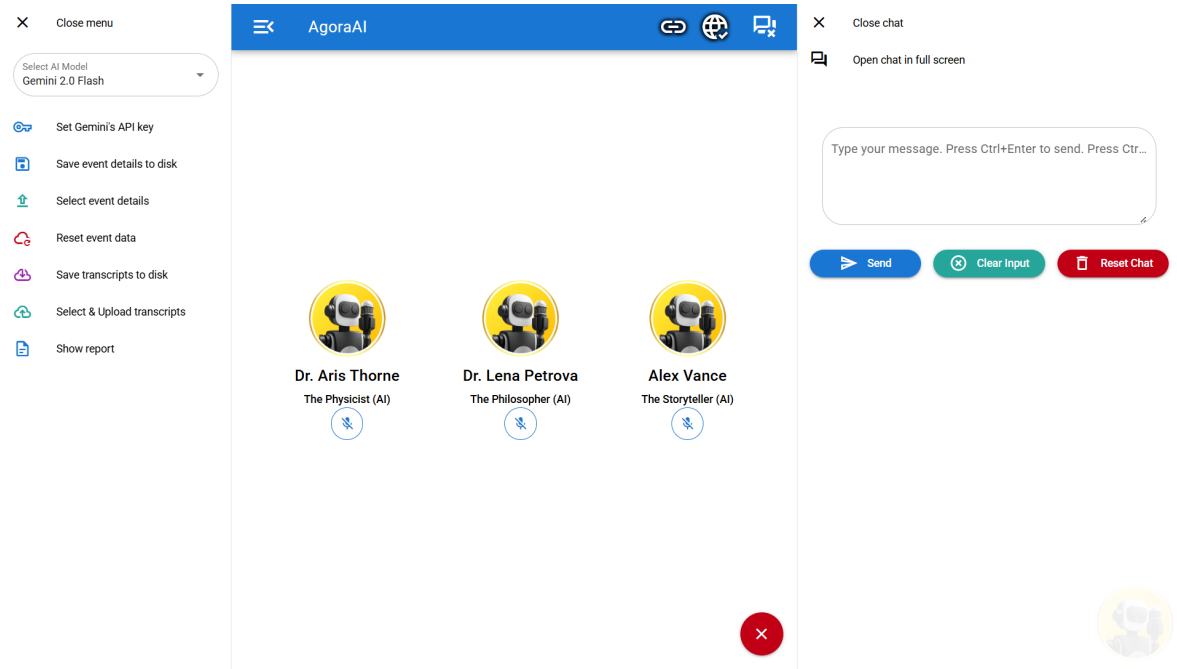

Figure 3. Screenshot of the Admin Interface in the Conversational State. The operator selects the current speaker from the list of participants to manage turn-taking.

The operator manages turn-taking by selecting the next active speaker (only one participant may speak at any time). For human participants, this activates voice recording followed by automatic STT processing of the captured audio. When selecting an AI participant, the system immediately requests a generated response from the corresponding agent, which is then streamed to the Audience Interface for simultaneous TTS playback and text display. This bidirectional workflow ensures seamless transitions between speaker types while maintaining strict turn-based protocol.

2.2. Audience Interface (Index Page)

The Audience Interface, shown in Figure 4 is also referred to as the Index Page. It is accessible at http://localhost:3000/#/, and designed for public display systems like projectors or large-format screens. It visually represents the ongoing event and the AI participants' activities.

 doi:10.20944/preprints202510.2309.v1

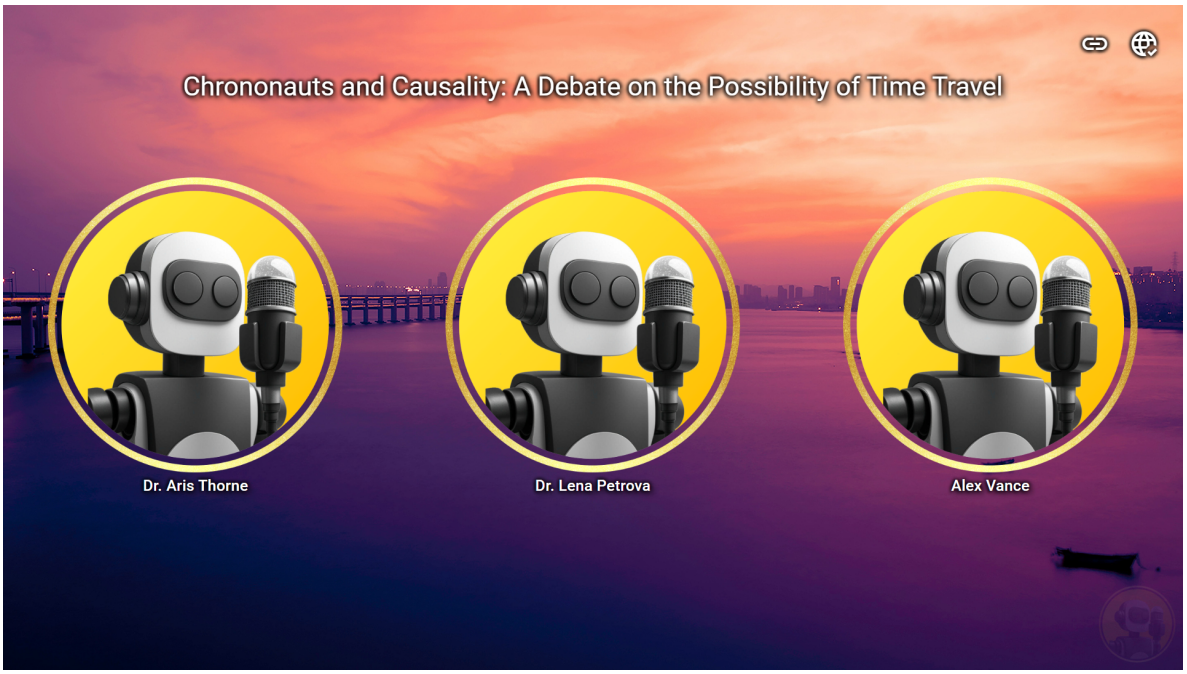

Figure 4. Screenshot of the Audience Interface (Index Page) displaying AI agents. The active AI agent changes its visual state (idle, thinking, talking) based on operator selection and LLM processing, providing dynamic feedback to the audience.

The background image and event name displayed on this page are those configured by the operator in the Admin Interface. A server status indicator is shown in the top-right corner, where a broken link icon signifies a disconnected server, while arrow icons indicate data reception from the server.

When an AI participant is selected by the operator to speak, the Audience Interface follows the next sequence:

1. The interface receives a command, and the avatar of the designated AI agent transitions from its 'idle' state to a 'thinking' state, visually indicating that a response is being processed.
2. Once the server has processed the request and the Large Language Model (LLM) has generated a response, this response is sent to the Audience Interface.
3. The corresponding AI agent's avatar then changes to a 'talking' state, and the integrated TTS system vocalizes the AI's response.
4. Upon completion of the speech, the AI agent's avatar reverts to its 'idle' state, awaiting further interaction or the next turn.

This dynamic visual feedback, coupled with TTS vocalization, aims to provide an engaging and informative experience for the audience observing the AI's participation in the event.

3. System Design

AgoraAI demonstrates scalability by supporting operation on both local machines and future central servers capable of hosting multiple Agoras, maintaining core functionality while handling concurrent requests. The platform exclusively employs mature technologies, including Google Cloud's AI services and Quasar Framework, to ensure robustness and reliability, deliberately avoiding experimental features. The entire project was developed using TypeScript to facilitate early error detection during development and to ensure smooth data transfer between the front-end and back-end through shared types/interfaces.

Figure 5 presents the technologies and communication protocols used in this project. The front-end files can be served either by a local server or by the back-end, which hosts the public folder containing the built front-end files. In this setup, the back-end serves the web application from the

 doi:10.20944/preprints202510.2309.v1

public folder and connects to the front-end via Socket.IO, with both HTTP and WebSocket services running on the same port (3000). Additionally, the server communicates with Google Cloud through HTTPS API requests. Gemini from Google Cloud was chosen becouse of the following reasons (1) It handles real-time voice to text translation, (2) extensive context capacity (up to 1 million tokens), making it ideal for prolonged debates, and (3) It gives free tockens to test the Gemini, which is suitable for evaluating AgoraAI.

The Quasar framework (built on Vue.js) was selected for two key reasons: its comprehensive framework capabilities that extend Vue.js's core functionality, and its built-in PWA support enabling Progressive Web App development with minimal source-code configuration changes.

The client-side application implements a structured architecture to manage its complexity:

- **State Management:** Global application state is managed by **Pinia**, the official state management library for Vue.js. Its modular store-based architecture is employed to manage distinct domains of state, such as socket connectivity (`socket-store`), real-time event data (`main-room-store`), and text-to-speech (TTS) operations (`speech-store`).
- **Real-time Communication:** Bidirectional, low-latency communication with the back-end server is achieved using the **Socket.io** library. This is fundamental to the application's real-time nature, enabling the instantaneous propagation of state changes, conversational turns, and control commands.
- **Styling: SCSS** is utilized for its advanced features over standard CSS, allowing for nested rules, variables, and mixins to create a consistent and maintainable design system.

Moreover, we use Node.js with Socket.IO instead of Python for this project due to four key advantages. First, Node.js's asynchronous architecture efficiently handles API calls to cloud-based LLMs, ensuring reliable data transfer. Second, the team's experience demonstrates that a TypeScript/-JavaScript codebase enables superior cross-platform performance as a Progressive Web App (PWA) while maintaining development efficiency. Third, Node.js's native non-blocking I/O model provides inherent scalability advantages for multiple concurrent users. Finally, Socket.IO simplifies implementation of bidirectional WebSocket communication, enabling robust real-time interactions that surpass traditional web server capabilities.

On the other hand, Google Cloud was was preferred over SaaS component due to two strategic advantages: (1) it offers a comprehensive AI service suite that combines Speech-to-Text and LLM functionalities, with future Text-to-Speech integration planned (though currently using native OS TTS for superior speed/reliability), and (2) it provides a developer-friendly free tier via API keys, enabling risk-free testing while allowing user-specific key updates as detailed in Section section 2. Finally, the front-end's algorithms are shown in Appendix A.

 doi:10.20944/preprints202510.2309.v1

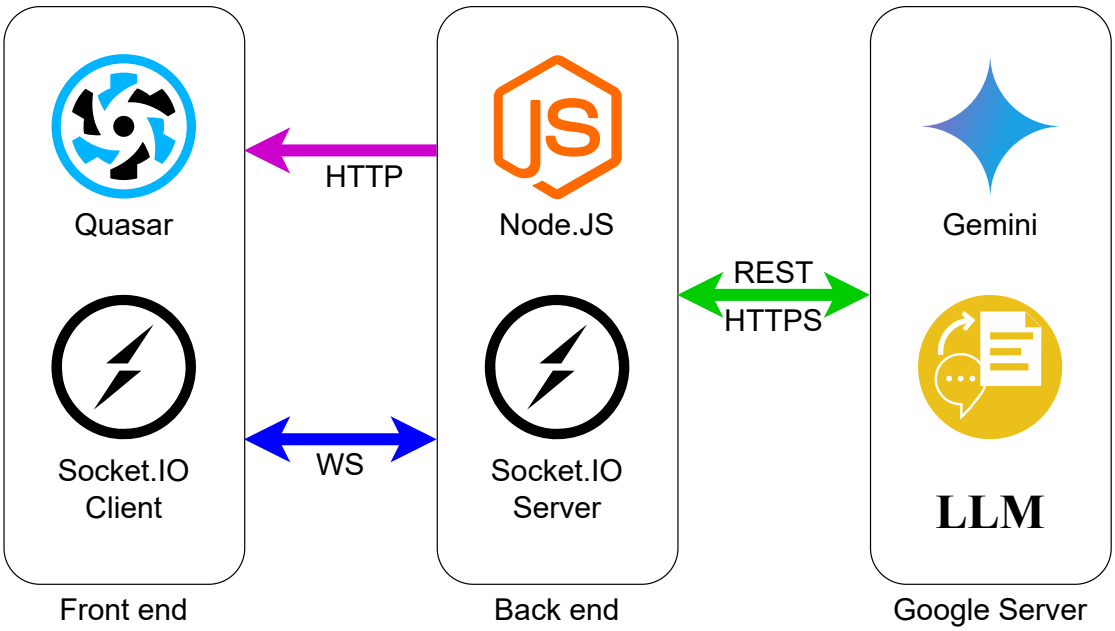

Figure 5. Block diagram showing the system design.

3.1. Back-end server

The back-end of the application is engineered as a robust, real-time, stateful server designed to facilitate dynamic, multi-participant interactions driven by generative artificial intelligence. The architecture is modular, leveraging modern technologies to manage complex conversational flows, process multimodal inputs, and maintain contextual coherence throughout an event's lifecycle.

3.1.1. Server and Communication Layer

The back-end is built using an `Express.js` web server developed in `Node.js` with `TypeScript`, which provides the necessary HTTP infrastructure. Real-time, bidirectional communication between the server and clients is handled via **Socket.IO**, enabling low-latency, event-based messaging essential for transmitting voice data, broadcasting AI-generated responses, and synchronizing client states in real-time. When the client loads, it emits the `app-init` event, prompting the server to subscribe the client to the `main-room`. This room is responsible for broadcasting AI-related events, including: (1) the AI model is thinking, (2) the AI model's response — the text-to-speech (TTS) content spoken by the AI on the index page, and (3) a signal to stop the AI from speaking, which is useful when an operator needs to interrupt the AI for reasons such as excessive speech length or other anticipated conditions. The server logic is modularized into separate socket-handling files: `admin-activities.ts` for administrative tasks and `app-main.ts` for general application events, ensuring a clean separation of concerns.

The client transitions from Setup-state to Conversational-state by sending full event details to the server via the `admin-activities-init-ai` Socket.IO event, with the payload containing the complete event data. The server receives and stores this information in the `event` variable. During the Conversational-state, once a human participant finishes speaking, their voice recording is sent to the server using the `admin-activities-voice-input-to-server` Socket.IO event, which includes both the participant's details and the audio recording blob. Upon receiving this event, the server proceeds to process the input accordingly. The back-end's algorithms for communication are described in Appendix B.

3.1.2. Generative AI Engine

The back-end connects to Google's cloud services through the npm package `@google/genai`, which is also used to manage the integration of the application's cognitive capabilities. These services

 doi:10.20944/preprints202510.2309.v1

power two critical functionalities using the `gemini-2.0-flash-001` multimodal Large Language Model (LLM): (1) *Speech-to-Text (STT) Transcription*, which converts incoming audio streams from participants into text, and (2) *Context-Aware Response Generation*, which produces conversational replies for AI participants based on a comprehensive, dynamically constructed prompt. The integration is coordinated through a centralized initialization module (`ai/initialization.ts`) that manages API key handling and access configuration. Additionally, the full event data and transcript history are stored in the variables `event` and `fullTranscript` within `pai/server/src/ai/full-history.ts`, enabling persistent tracking of AI interactions.

3.1.3. State and History Management

A core feature of the back-end is its ability to maintain a complete state of the simulated event. The `ai/full-history.ts` module acts as the single source of truth, storing event metadata (e.g., name, dynamics), participant profiles (e.g., name, role, bio), and a chronological, turn-by-turn transcript of the entire conversation. This stateful design is fundamental to the system's ability to generate contextually relevant AI responses.

3.1.4. Features and Data Flow

The back-end orchestrates a sophisticated data flow to create a seamless conversational experience. The process for a single conversational turn can be broken down into the following stages:

1. **Session Initialization:** An administrative client initiates or continues an event by sending `GSK_SETTINGS_TO_INIT_AI` or `GSK_SETTINGS_TO_CONTINUE_AI_WITH_HISTORY` payloads. This populates the server's state with the event's parameters, participant data, and, if applicable, a pre-existing transcript.
2. **Voice Input Processing:** Human participant voice input is captured on the client, encoded, and transmitted to the server via a `GSK_VOICE_INPUT_TO_SERVER` socket event. The server receives this audio data, saves it as a temporary `.wav` file, and adds a reference to it into a dedicated processing queue.
3. **Asynchronous Transcription:** The audio processing is handled asynchronously by a queue (`toAppendQueue` in `ai/audios.ts`). For each audio file, the system uploads it to the Google AI platform, invokes the Gemini model for transcription, and appends the resulting text to the main event transcript (`fullTranscript`), associating it with the correct speaker.
4. **AI Response Generation:** An AI participant's turn to speak, triggered by a `GSK_REQUEST_AI_TO_START_TALKING` event, enqueues the request into a separate response queue (`queueToGetResponse` in `ai/main.ts`). Processing of this queue is deliberately delayed until all pending audio transcriptions are complete (`isAllTasksDone()`), guaranteeing the AI has full context.
5. **Dynamic Prompt Engineering:** When an AI response is generated, the system constructs a detailed prompt. A *System Prompt* (`getSystemPrompt`) primes the LLM with its persona, while a *Main Prompt* (`getPromptForAI`) provides the full event context and conversational history.
6. **Response Delivery and State Update:** The LLM's response is appended to the `fullTranscript` and broadcast to all clients via the `main-room-ai-response` socket event (`emitAIResponse`), enabling client-side text-to-speech synthesis and UI updates.

4. Case Studies and Implementation

AgoraAI has demonstrated significant utility across diverse professional and academic fields and has been adopted by users of various ages and expertise levels. Beyond enabling complex multi-agent and multi-human conversations, the platform excels at facilitating activities that require structured dialogue, such as presenting arguments or debating topics from opposing viewpoints. Example events showcasing these capabilities have been generated and are available in the folder `example-events` of the GitHub repository [29]. A brief description of these events is provided in Table 1.

 doi:10.20944/preprints202510.2309.v1

The following sections detail the successful deployment and observed efficacy of AgoraAI in structured environments.

4.1. Simulated Multi-Agent Panel

AgoraAI was specifically engineered and successfully deployed to function as one or multiple virtual panel members. This capability is frequently utilized by educators, conference organizers, and professional facilitators seeking to enrich discussions by incorporating diverse, AI-driven perspectives into a shared conversational environment. Its core differentiation from standard single-agent conversational systems lies in its architecture, which allows for the simultaneous configuration and deployment of several distinct AI personas. This design choice effectively simulates the dynamics and complexities of a real-world multi-viewpoint panel discussion.

4.2. Augmented Academic Tutoring

In university settings, AgoraAI was deployed to augment tutoring sessions, demonstrating utility in reducing student attrition rates and improving both academic knowledge and essential soft skills. The platform participated alongside a human teacher and a student, enabling an interaction that was consistently informative and current. The system's effectiveness was observed in its capacity to help professors structure their pedagogical thoughts while providing students with greater clarity on the concepts discussed. A key differentiating operational advantage is the platform's automatic generation of a complete conversation transcript, referred to as "Notes." This feature serves students by allowing them to review the tutoring activity asynchronously in their personal space, thereby reducing the necessity for continuous manual note-taking during the session and enabling greater focus on the content.

4.3. Real-time Meeting Assistant

AgoraAI was successfully employed as an intelligent AI meeting assistant, serving project managers and participants by helping to structure and conduct professional proceedings efficiently. Its effectiveness is evidenced by its capacity to generate comprehensive summaries and prepare necessary, tailored action items for each participant immediately upon conclusion. A key differentiating feature is the integrated chat window, which allows users to inquire about custom details and retrieve additional contextual information useful for post-meeting documentation, extending the system's utility beyond basic transcription and summary generation.

4.4. High-Stakes Examination Simulation

As an experimental case study, AgoraAI was implemented to facilitate rigorous preparation for a master's student's high-stakes oral examination. Serving the student in an intense practice setting, the platform was configured to simulate a panel composed of three distinct examiners. All examiners specialized in the same field but represented different areas of expertise, which is the crucial multi-agent differentiation that accurately simulates a complex assessment environment. The observed effectiveness was significant: the questions generated by the AgoraAI agents were highly relevant and precise, leading to a demonstrable improvement in the student's final presentation quality and overall preparedness for the formal defense.

 doi:10.20944/preprints202510.2309.v1

Table 1. Summary of Example Event Configurations available in the AgoraAI repository [29] in the folder `example-events`, where N. Part, represents the number of participants.

Filename	Categories	N. Part.	Core Function	Dynamics
`formal/event-peer--tutoring.-json`	Education	2	Focused, one-on-one tutoring session on complex concepts (Circuit Analysis Study Session).	Interactive, collaborative, and supportive session, where the tutor guides the student through problems.
`formal/event-project--discussion.-json`	Corporate	3	Formal meeting to discuss and evaluate a new project proposal (Project Mercury Proposal Review).	Project leader presents, followed by a moderated discussion and secretary documenting, key points and action items.
`informal--chat/chrononauts--and--causality`	Academic Debate	3	Debate on the scientific and philosophical feasibility of time travel (Chrononauts and Causality).	Structured debate format with three rounds: presentation, moderated discussion, and audience Q&A.
`informal--chat/even--scroll--fortune.-json`	Social/Economy	3	Lively debate about the rise of the influencer economy, covering business and mental health implications (The Scroll of Fortune).	Panel discussion format where each student presents a different perspective (optimist, skeptic, analyst), followed by Q&A.
`informal--chat/even--scroll--fortune.-json`	Career Guidance	4	Unique career guidance session for primary school students (Dream Big: A Career Exploration for Young Minds).	45-minute interactive, fun, light-hearted Q&A between a student, parent, CEO, and neurosurgeon.
`informal--chat/event--hero-mind.-json`	Pop Culture/Ethics	3	Discussion exploring the motivations, ethical dilemmas, and psychological traits of iconic heroes and villains (The Hero's Mind).	Moderated discussion with students presenting arguments and counterpoints based on specific character case studies.
`informal--chat/event--pages-to--pixels.json`	Arts/Media	3	Comparative analysis of storytelling across different mediums, from literature to TV series (From Pages to Pixels).	"Show and tell" format: each student presents a book/film adaptation case study, followed by group discussion.
`informal--chat/event--sweat.json`	Health/Lifestyle	4	Dialogue exploring diverse approaches to health and fitness, balance, and mental health impact (Sweat, Balance, & Thrive).	60-minute panel discussion: 30 minutes moderated Q&A, followed by 30 minutes open forum for audience participation.

Continued on next page

 doi:10.20944/preprints202510.2309.v1

Table 1. Summary of Example Event Configurations available in the AgoraAI repository [29] in the folder `example-events`, where N. Part, represents the number of participants.

Filename	Categories	N. Part.	Core Function	Dynamics
`panel--discussions/sigrama--day-2025`	Industry/AI	6	Panel discussion on AI in the industry, producing industrial transformation, and broad societal impacts (Sigrama's day 2025).	Panel session in Spanish (es-MX) with dedicated time for presentations, audience Q&A, and conclusions.
`panel--discussions/supply--chain`	Logistics/ Ethics	5	Exploring the future of global supply chains, focusing on disruption, ethical sourcing, and digital transformation.	Structured discussion including opening remarks, a 55-minute moderated panel, and audience Q&A with a live poll.
`panel--discussions/event--algorithmic--judge.json`	Ethics/Tech Policy	4	Confronting the challenges of bias, accountability, and fairness as AI automates decisions (The Algorithmic Judge).	Town Hall style debate: initial statements from experts followed by moderator-facilitated questions directly from the audience.
`panel--discussions/event-bits--to-beings.-json`	Science/ Interdisciplinary	4	Deep dive into the fundamental concept of information across physics, biology, and human language (From Bits to Beings).	Each panelist gives a 15-minute presentation, followed by a 45-minute moderated debate and audience discussion.
`panel--discussions/event--casual-web.-json`	Science/Data	4	Discussion tackling the problem of distinguishing cause from correlation in the age of big data (The Causal Web).	Socratic Dialogue format: the moderator poses foundational questions to encourage direct engagement and debate among the panelists.
`panel--discussions/event--climate--action.json`	Environment/ Policy	5	Bridging scientific consensus, policy implementation, and community engagement for climate change action.	Structured discussion probing the intersection of science and policy, followed by audience Q&A and a final call to action.
`panel--discussions/event--future-of--work.json`	Labor/ Technology	5	Exploring the impact of AI and automation on the global workforce and strategies for human adaptability (The Future of Work).	Structured discussion including moderated debate, audience Q&A, and final key takeaways from experts.

Continued on next page

 doi:10.20944/preprints202510.2309.v1

Table 1. Summary of Example Event Configurations available in the AgoraAI repository [29] in the folder `example-events`, where N. Part, represents the number of participants.

Filename	Categories	N. Part.	Core Function	Dynamics
`panel--discussions/event--life-in--netrowks.-json`	Neuroscience/ Sociology	4	Exploration of the science of networks in brains, societies, and ecosystems (Connectomics: Life in the Network).	Interactive session featuring live network visualizations on which panelists comment, followed by moderated discussion.
`panel--discussions/event--unseen--hand.json`	Philosophy/ Systems	4	Interdisciplinary panel exploring how complex, organized systems arise from simple, individual actions (The Unseen Hand).	A 90-minute session comprising an initial moderator Q&A, a cross-panel debate, and a final open Q&A

5. Proposed Enhancements and Future Directions

While AgoraAI is currently a complete and readily implementable open-source solution, the platform's potential can be further realized through enhancements that leverage advancements in supporting technologies. The primary limitation identified in the current deployment is the reliance on a human operator to manage turn-taking in real-time. Specifically, a human administrator must manually select the active speaker via the Admin Interface, a necessary step due to the current constraints in real-time diarization. Existing diarization techniques, while capable of detecting voice activity, often struggle to reliably identify and distinguish individual speakers in a multi-human context.

Our highest priority for future development involves integrating mature, real-time speaker diarization technologies to automate this manual process. As these algorithms advance, their integration will allow the system to automatically identify the current speaker, thereby eliminating the need for human intervention in turn management. This automation will significantly enhance the system's operational efficiency, reduce the complexity of event facilitation, and move the platform toward full conversational autonomy in complex multi-human environments.

Beyond automation of speaker selection, future efforts will focus on expanding the multimodal capabilities and flexibility of the agents. This includes revisiting the Text-to-Speech (TTS) implementation. Currently, the system prioritizes the speed and reliability of native operating system TTS engines. Future development will explore incorporating advanced, high-fidelity, cloud-based TTS solutions (such as Google's Text-to-Speech service) to further personalize and refine the acoustic presence of the AI agents. Furthermore, subsequent work will investigate protocols for enhanced multi-agent synchronization and decision-making, allowing for more nuanced collaborative responses and greater operational scalability in large, distributed conversational simulations.

6. Conclusions

This paper introduced AgoraAI, a novel open-source solution designed to bridge the gap in tools supporting voice-to-voice conversations involving multiple distinct AI agents and multiple human participants within the same session.

AgoraAI provides a comprehensive and user-friendly framework that allows operators to configure and deploy multiple AI agents—each with a defined persona, role, and biography—to participate in turn-based discussions alongside humans. The core implementation utilizes robust and mature technologies, including Node.js/TypeScript, the Quasar framework for the Progressive Web Application (PWA) front-end, and Socket.IO for establishing low-latency, real-time communication channels. The

 doi:10.20944/preprints202510.2309.v1

system leverages Google's Gemini LLM for critical multimodal tasks, specifically Speech-to-Text (STT) transcription and context-aware response generation.

A key technical innovation lies in the back-end's Asynchronous Dual-Queue Processing architecture, which employs a strict synchronization condition to ensure AI responses are computed only after all pending audio transcriptions are complete. This guarantees full conversational coherence. Furthermore, the Dynamic Context-Injection and Persona Templating algorithms ensure the LLM receives the maximum possible context, resulting in highly relevant and accurate outputs.

Initial evaluations in university and industrial contexts demonstrated significant utility and positive acceptance, showcasing successful applications in structured environments such as panel discussions, academic tutoring, meeting assistance, and oral exam evaluation. While the current implementation requires a human operator to manually select the active speaker, future work will focus on integrating advanced real-time diarization technologies to automate speaker identification and further reduce human involvement. AgoraAI provides a significant step forward in facilitating fluid, multi-perspective human-AI collaboration.

Appendix A Front-end's core Algorithms

The front-end's innovative capabilities are powered by the following algorithms.

Algorithm 1: Real-time Conversational State Machine

The core of the application is a client-side state machine, managed by Pinia stores and driven by server-sent events via Socket.io. This architecture ensures the UI is always a direct reflection of the conversational state.

1. **Initialization:** On connection, the client receives the complete event model (`GSK_AI_FULL_EVENT_DATA_TO_CLIENT`) from the server, populating the `main-room-store`.
2. **State Definition:** The primary state variables are `speakerIdx` (the index of the current or next speaker) and `speakerState` (an enum of 'idle', 'thinking', 'talking').
3. **State Transitions:**
   - A `main-room-ai-is-thinking` event from the server sets the `speakerIdx` and updates the `speakerState` to 'thinking', triggering a visual change in the corresponding avatar.
   - A `main-room-ai-response` event delivers the generated text. This text is passed to the TTS engine. The `onstart` callback of the speech utterance transitions the `speakerState` to 'talking'.
   - The `onend` callback of the speech utterance resets the speaker state, preparing the system for the next turn.

This event-driven model provides a robust and reactive system for visualizing a dynamic, multi-agent conversation.

Algorithm 2: Client-side Audio Processing Pipeline

To capture human input, a multi-step audio processing pipeline is implemented within the `SendInputsToServer.vue` component.

1. **Capture:** The `navigator.mediaDevices.getUserMedia` API is invoked to request microphone access and capture an audio stream.
2. **Encoding:** The stream is processed by the `MediaRecorder` API. The application first queries for the most desirable supported MIME type (preferring lossless formats like `audio/wav`) and falls back to the widely supported `audio/webm`.
3. **On-the-fly Conversion:** A critical step is the `convertWebmToWav` function, executed if the browser defaults to WebM. This function performs the following sub-steps:
   (a) The received WebM blob is read into an `ArrayBuffer`.

 doi:10.20944/preprints202510.2309.v1

(b) The `WebAudio API`'s `audioContext.decodeAudioData` method is used to decode the buffer into raw PCM audio data.
(c) A new `ArrayBuffer` is created and a valid 44-byte WAV file header is manually written to it, specifying format, channel count, sample rate, and bit depth.
(d) The raw PCM data from the `AudioBuffer` is then appended to the header.
(e) The final buffer is converted into a `Blob` with the `audio/wav` MIME type.

This ensures that the back-end receives a consistent, high-quality audio format, simplifying the speech-to-text processing pipeline.

4. **Transmission:** The final audio blob is transmitted to the server via the `admin-activities-voice-input-to-server` Socket.io event.

Algorithm 3: TTS Lifecycle and Reliability Management

The application implements a robust wrapper around the browser's `SpeechSynthesis` API to manage its lifecycle and handle cross-browser inconsistencies.

1. **Voice Loading:** Available system voices are loaded asynchronously on the `window.speechSynthesis.onvoiceschanged` event and stored in the `speech-store`.
2. **Utterance Execution:** When an AI is to speak, a `SpeechSynthesisUtterance` is created with the designated text and voice.
3. **State Synchronization:** The utterance's `onstart` and `onend` event handlers are used to precisely synchronize the application's visual state ('talking' vs. 'idle') with the audible speech, as described in Algorithm 1.
4. **Keep-Alive Mechanism:** To counteract a known issue in some browser implementations where the speech synthesis engine can time out and halt during long pauses, a proactive "keep-alive" mechanism is implemented. A `setInterval` is initiated at the start of an utterance, which periodically calls `window.speechSynthesis.resume()`. This ensures that the speech queue remains active and prevents premature termination of long-form AI responses. The interval is cleared when the utterance completes successfully.

Appendix B Communication of Innovation: Back-end's Key Algorithms

The innovation of this back-end system is not derived from a single novel algorithm, but from the orchestration of existing technologies through two key architectural patterns that can be described algorithmically.

 doi:10.20944/preprints202510.2309.v1

Algorithm 1: Asynchronous Dual-Queue Processing for Conversational Coherence

Algorithm A1 Asynchronous Dual-Queue Processing for Conversational Coherence

```
1: Global State: Queue_Audio, Queue_Response, isProcessingAudio, isProcessingResponse, MainTranscript
2: procedure OnAudioInput(audioData, speaker)
3:     filePath ← Save audioData to a local file
4:     Enqueue {filePath, speaker} into Queue_Audio
5:     ProcessAudioQueue
6: end procedure
7: procedure ProcessAudioQueue
8:     if Queue_Audio is empty or isProcessingAudio then return
9:     end if
10:    Set isProcessingAudio ← true
11:    task ← Dequeue from Queue_Audio
12:    transcript ← async LLM.Transcribe(task.filePath)
13:    On completion, append transcript to MainTranscript
14:    Set isProcessingAudio ← false
15:    ProcessAudioQueue
16: end procedure
17: procedure OnRequestAIResponse(participant)
18:    Enqueue participant into Queue_Response
19:    ProcessResponseQueue
20: end procedure
21: procedure ProcessResponseQueue
22:    if isProcessingResponse or not Queue_Audio.isEmpty or isProcessingAudio then
23:        return                                          ▷ Synchronization Condition
24:    end if
25:    if Queue_Response is empty then return
26:    end if
27:    Set isProcessingResponse ← true
28:    participant ← Dequeue from Queue_Response
29:    prompt ← GenerateFullPrompt(MainTranscript)
30:    response ← async LLM.Generate(prompt)
31:    On completion, append response to MainTranscript and broadcast
32:    Set isProcessingResponse ← false
33:    ProcessResponseQueue
34: end procedure
```

The **Synchronization Condition** (line 22) is the most critical step, ensuring that the system's state is fully updated before an AI computes its response.

 doi:10.20944/preprints202510.2309.v1

Algorithm 2: Dynamic Context-Injection and Persona Templating

Algorithm A2 Dynamic Context-Injection and Persona Templating

procedure GENERATEFULLPROMPT(*MainTranscript*)
    Input: Current AI participant P, Event details E.
    ▷ 1. Persona Templating
    SystemPrompt ← Format a string: *"You are {P.name}, a {P.role} in event '{E.name}'. Your bio is: {P.bio}..."*
    ▷ 2. Context Aggregation
    Initialize *MainPrompt* with event, role, and participant metadata.
    for each *utterance* in *MainTranscript* **do**
        Format and append utterance to *MainPrompt*:
        `"- Name: {utterance.name}, Role: {utterance.role}, Content: {utterance.content}"`
    end for
    ▷ 3. Execution Packet
    return (*SystemPrompt*, *MainPrompt*)
end procedure

This algorithmic approach to prompt engineering ensures that the model receives the maximum possible context, encompassing both the immediate conversational history and the overarching rules and identities of the simulation, leading to highly relevant and coherent outputs.

References

1. Bommasani, R.; Hudson, D.A.; Adeli, E.; Altman, R.; Arora, S.; von Arx, S.; Bernstein, M.S.; Bohg, J.; Bosselut, A.; Brunskill, E.; et al. On the Opportunities and Risks of Foundation Models, 2022, [arXiv:cs.LG/2108.07258].
2. Guo, T.; Chen, X.; Wang, Y.; Chang, R.; Pei, S.; Chawla, N.V.; Wiest, O.; Zhang, X. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680* **2024**.
3. Li, X.; Wang, S.; Zeng, S.; Wu, Y.; Yang, Y. A survey on LLM-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth* **2024**, *1*, 9.
4. Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F.L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774* **2023**.
5. Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* **2023**.
6. Gilardi, F.; Alizadeh, M.; Kubli, M. ChatGPT outperforms crowd workers for text-annotation tasks. *Proceedings of the National Academy of Sciences* **2023**, *120*, e2305016120.
7. Chen, P.Y.; Liu, S. *Introduction to Foundation Models*; Springer Nature, 2025.
8. Zhang, D.; Li, S.; Zhang, X.; Zhan, J.; Wang, P.; Zhou, Y.; Qiu, X. SpeechGPT: Empowering Large Language Models with Intrinsic Cross-Modal Conversational Abilities, 2023, [arXiv:cs.CL/2305.11000].
9. Saravana, C.P.M.K.S.; Mandyam, S.K.; Thondamanati, C.; Akkim, D.; Ramapogu, R. Contrive the chat & voice bot based on personal automation through artificial intelligence. In Proceedings of the AIP Conference Proceedings. AIP Publishing LLC, 2025, Vol. 3237, p. 020009.
10. Radford, A.; Kim, J.W.; Xu, T.; Brockman, G.; McLeavey, C.; Sutskever, I. Whisper: Robust speech recognition via large-scale weak supervision. *arXiv preprint arXiv:2212.01234* **2022**.
11. Palbou, L. VoiceLLM: A Modular Python Library for Voice Interactions with AI Systems. https://www.voicellm.ai/, 2025. Accessed: October 26, 2025.
12. Lee, O.; Joseph, K. A large-scale analysis of public-facing, community-built chatbots on Character.AI, 2025, [arXiv:cs.SI/2505.13354].
13. Seaborn, K.; Miyake, N.P.; Pennefather, P.; Otake-Matsuura, M. Voice in human–agent interaction: A survey. *ACM Computing Surveys (CSUR)* **2021**, *54*, 1–43.
14. Park, J.S.; O'Brien, J.; Cai, C.J.; Morris, M.R.; Liang, P.; Bernstein, M.S. Generative agents: Interactive simulacra of human behavior. In Proceedings of the Proceedings of the 36th annual acm symposium on user interface software and technology, 2023, pp. 1–22.

15. de Aquino e Aquino, G.; da Silva de Azevedo, N.; Okimoto, L.Y.S.; Camelo, L.Y.S.; de Souza Bragança, H.L.; Fernandes, R.; Printes, A.; Cardoso, F.; Gomes, R.; Torné, I.G. From RAG to Multi-Agent Systems: A Survey of Modern Approaches in LLM Development. *Preprints* **2025**. https://doi.org/10.20944/preprints202502.0406.v1.
16. Cox, S.R. The use of multiple conversational agent interlocutors in learning. *arXiv preprint arXiv:2312.16534* **2023**.
17. Taulli, T.; Deshmukh, G. CrewAI. In *Building Generative AI Agents: Using LangGraph, AutoGen, and CrewAI*; Springer, 2025; pp. 103–145.
18. Qian, C.; Liu, W.; Liu, H.; Chen, N.; Dang, Y.; Li, J.; Yang, C.; Chen, W.; Su, Y.; Cong, X.; et al. Chatdev: Communicative agents for software development. *arXiv preprint arXiv:2307.07924* **2023**.
19. Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Li, B.; Zhu, E.; Jiang, L.; Zhang, X.; Zhang, S.; Liu, J.; et al. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In Proceedings of the First Conference on Language Modeling, 2024.
20. Hong, S.; Zhuge, M.; Chen, J.; Zheng, X.; Cheng, Y.; Wang, J.; Zhang, C.; Wang, Z.; Yau, S.K.S.; Lin, Z.; et al. MetaGPT: Meta programming for a multi-agent collaborative framework. In Proceedings of the The Twelfth International Conference on Learning Representations, 2023.
21. Wu, Y.; Jiang, Z.; Khan, A.; Fu, Y.; Ruis, L.; Grefenstette, E.; Rocktäschel, T. ChatArena: Multi-Agent Language Game Environments for Large Language Models. GitHub repository, Farama-Foundation/chatarena, 2023. Accessed: October 26, 2025.
22. Xu, L.; Hu, Z.; Zhou, D.; Ren, H.; Dong, Z.; Keutzer, K.; Ng, S.K.; Feng, J. MAgIC: Investigation of Large Language Model Powered Multi-Agent in Cognition, Adaptability, Rationality and Collaboration. In Proceedings of the Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing; Al-Onaizan, Y.; Bansal, M.; Chen, Y.N., Eds., Miami, Florida, USA, 2024; pp. 7315–7332. https://doi.org/10.18653/v1/2024.emnlp-main.416.
23. Tran, K.T.; Dao, D.; Nguyen, M.D.; Pham, Q.V.; O'Sullivan, B.; Nguyen, H.D. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322* **2025**.
24. Sun, J.; Kou, J.; Shi, W.; Hou, W. A multi-agent collaborative algorithm for task-oriented dialogue systems. *International Journal of Machine Learning and Cybernetics* **2025**, *16*, 2009–2022.
25. Guo, X.; Huang, K.; Liu, J.; Fan, W.; Vélez, N.; Wu, Q.; Wang, H.; Griffiths, T.L.; Wang, M. Embodied llm agents learn to cooperate in organized teams. *arXiv preprint arXiv:2403.12482* **2024**.
26. Li, H.; Chong, Y.; Stepputtis, S.; Campbell, J.; Hughes, D.; Lewis, C.; Sycara, K. Theory of Mind for Multi-Agent Collaboration via Large Language Models. In Proceedings of the Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing; Bouamor, H.; Pino, J.; Bali, K., Eds., Singapore, 2023; pp. 180–192. https://doi.org/10.18653/v1/2023.emnlp-main.13.
27. Kosinski, M. Theory of mind may have spontaneously emerged in large language models. *arXiv preprint arXiv:2302.02083* **2023**, *4*, 169.
28. Aleedy, M.; Atwell, E.; Meshoul, S. A Multi-Agent Chatbot Architecture for AI-Driven Language Learning. *Applied Sciences* **2025**, *15*. https://doi.org/10.3390/app151910634.
29. Gadi, S.K. AgoraAI: A Multi-Agent Conversational Framework. https://github.com/skgadi/agora-ai, 2025. Available on GitHub. Last accessed: October 18, 2025.
30. Gadi, S.K. AgoraAI: Docker Hub instllation source. https://hub.docker.com/r/skgadi/sigrama-agora-ai, 2025. Available on GitHub. Last accessed: October 18, 2025.