

Building LLM-Based Voice Agents for Requirements Elicitation: An Experience Report on Early Prototypes

Oshani Weerakoon
University of Turku
Turku, Finland
osweer@utu.fi

Erkki Kaila
University of Turku
Turku, Finland
ertaka@utu.fi

Tuomas Mäkilä
University of Turku
Turku, Finland
tusuma@utu.fi

Shola Oyedeji
LUT University
Lappeenranta, Finland
Shola.Oyedeji@lut.fi

Abstract

This paper presents an experience report on the development and pre-testing of voice-based agentic workflows that uses two large language models, OpenAI's GPT-4o-mini and Google's Gemma3:27b to conduct requirement elicitation discussions in software projects. The growing use of independent AI agents for specialized tasks has motivated our exploration of voice agents as "requirements elicitors" within software projects. The paper describes the approaches attempted during development, including those that were successful and those that failed, along with insights gathered from implementing and testing the use cases. We conducted a pre-test round with five participants, comparing the performance of the two agents under two case studies. At this stage, the OpenAI-based agent showed a higher requirements coverage, identifying 77.5% of relevant requirements on average, while the Gemma-based agent captured 35.0%. In terms of usability, participants rated the OpenAI agent 4.0/5, compared to 3.3/5 for the Gemma agent, highlighting a more natural conversational flow, better contextual understanding, and improved responsiveness. We propose deploying this voice agent as the first agent in an extended multi-agent requirements engineering workflow to support requirement elicitation sessions alongside a human requirement engineer, which will be the extension of this work. The methods, design choices, and lessons learned documented in this report aim to guide practitioners and researchers in adapting similar agent-based approaches in their own domains.

CCS Concepts

• **Computing methodologies** → **Speech recognition; Natural language generation**; • **Software and its engineering** → **Agile software development; Requirements analysis**.

Keywords

Requirements Engineering, Natural Language Requirements, Generative AI, Voice Agents



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

AGENT '26, Rio de Janeiro, Brazil

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2399-5/26/04

<https://doi.org/10.1145/3786167.3788420>

ACM Reference Format:

Oshani Weerakoon, Tuomas Mäkilä, Erkki Kaila, and Shola Oyedeji. 2026. Building LLM-Based Voice Agents for Requirements Elicitation: An Experience Report on Early Prototypes. In *International Workshop on Agentic Engineering (AGENT '26)*, April 12–18, 2026, Rio de Janeiro, Brazil. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3786167.3788420>

1 Introduction

In the present literature, there exists more than one definition for what is an "agent" [9, 19, 20]. According to Dorri et al. [9], an agent is entity that senses different parameters in an environment, uses that information to make a decision based on its goal, and performs the necessary action on the environment.

Early voice based systems like chatbots and speech-based assistants relied on predefined scripts, rules, and limited domain operations. Both pre-LLM chatbots and traditional interactive voice response systems were primarily rule-based systems [4], and usually followed pre-coded workflows that progressed through scripted question-answer sequences [1]. They had limited world knowledge base, could not properly follow the context shifts in during conversations [6] or handle interruptions or overlapping speech [21]. In contrast to normal human interaction, the majority of previous voice-based systems were operated in half-duplex mode (data is sent sequentially but not simultaneously (e.g., Walkie Talkie)). Legacy digital assistants and dictation software, for instance, were developed using rule-based algorithms that had been gradually improved upon over decades to reach the statistical and neural models of the 2000s [4].

The capabilities of voice based systems have been significantly transformed due to the emergence of powerful generative AI models, particularly large language models (LLMs). Conversational systems powered by LLMs that can understand and produce natural speech in real time are referred to as generative AI voice agents [1], and are able to generate contextually relevant and original responses [1], unlike their rule-based predecessors.

In Requirement Engineering (RE), understanding the problem area is crucial to usability and usefulness of the software system [14]. However, it depends on the way requirements are communicated [7] and user-involved approaches usually lead to better requirements gathering [7]. Successful requirements elicitation and gathering depend on effective communication between stakeholders and the software development team. Voice agents, which are

capable of natural dialogue, can support and automate parts of this process. However, the potential of such agents to collaborate closely with humans, especially by acting as interviewers or "elicitors" during requirements elicitation discussions, remains underexplored. Our motivation of this work is to address the research gap by examining generative voice agents within structured multi-agent RE workflows utilizing modern high-capability LLMs.

Our study is guided by the following research question: **RQ: How do proprietary and open-source LLM-based voice agents perform requirements elicitation for a software product in interaction with a human client?**

In this paper, we present our initial work on utilizing both open-source and proprietary large language models to develop voice-based agents capable of initiating and sustaining meaningful interactions with humans. For this round of experiments, we employ the OpenAI GPT-4o-mini and Gemma3:27b models; however, any open-source LLM can be easily swappable within our current implementation. We chose GPT-4o-mini, as it has demonstrated a promising potential in the recent voice AI related research, and Gemma3:27b from Gemini family, as it carries the capabilities of a powerful model despite being lightweight, therefore leading to minimal latency and response times in voice based conversation prompting. The voice agent functions as the "elicitor", responsible for conducting elicitation discussions with human clients regarding a software product idea. A preliminary round of testing was conducted with five participants using two case studies. We present comparative results of the two voice agents, evaluated in terms of coverage of extracted requirements and usability when eliciting requirements. The ultimate objective of this research is to integrate an AI-based voice agent as the first agent, in a multi-agent Requirements Engineering workflow that effectively meets human user expectations.

2 Experimental Design

In this section, we present our approach to utilizing two base models, one proprietary and one open-source, for a voice-enabled conversational agent. Note that certain steps may vary based on the operating system and hardware configuration.

We use the following software and hardware configurations, in this experiment.

- Apple M3 laptop running MacOS, 16GB RAM.
- Locally hosted GPU running Ollama (version 0.12.3) with access to several large language models.

Figure 1 illustrates the flow of processing events for both agents. Each voice agent automatically starts, listens for user speech, transcribes it, generates a model response, and speaks the response back through the system speakers. The conversation can continue indefinitely, and logs and audio recordings are created simultaneously.

A 15 seconds fixed duration recording window was chosen instead of continuous listening with voice activity detection (VAD)¹ to avoid capturing unnecessary background voices, in both approaches. To improve usability, an *interrupt mechanism* was implemented, allowing the user to stop ongoing playback instantly by pressing the *Enter* key, to enhance the control and responsiveness.

¹<https://www.tavus.io/post/voice-activity-detection>

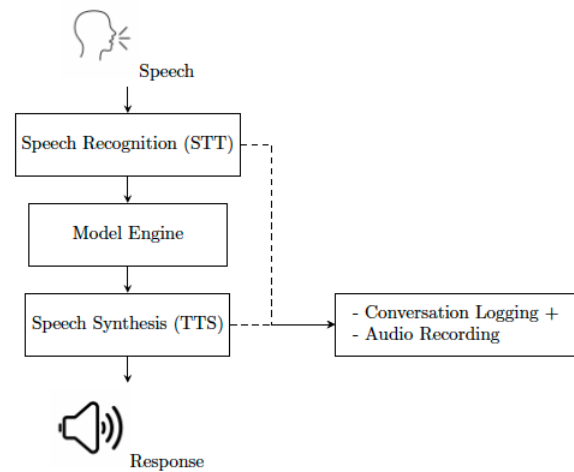


Figure 1: End-to-end processing flow of the voice agents

All configuration data, including API credentials, are externalized in a `.env` file to ensure portability and security across development environments. We chose to maintain logging of every conversational turn, including text transcripts, timestamps, and combined user-elicitor audio in both approaches. The goal is to preserve transparency and repeatability for evaluating conversational AI behavior within experimental or future empirical contexts.

In the following sections, the approaches are explained in further detail. Table 1 presents a side-by-side comparison of the two agentic pipelines.

2.1 Voice Agent 1: Using GPT-4o-mini

In this section, we present our approach on building a human-interactive voice agent using OpenAI's Agents SDK and voice extensions.

2.1.1 Design Decisions and Rationale. The use of the `sounddevice` library provided native access to the system microphone and speakers through the Core Audio interface, ensuring low-latency capture and playback without external dependencies. The decision to use the `VoicePipeline` and `SingleAgentVoiceWorkflow` classes was guided by their compatibility with local event loops and asynchronous input/output handling.

To begin, a Python virtual environment was initialized to manage dependencies in an isolated context, including the following packages.

- `openai`: for accessing OpenAI LLMs
- `openai-agents[voice]`: for using OpenAI's experimental Agents SDK and voice extensions
- `sounddevice`, `numpy`: for audio capture and processing
- `python-dotenv`: for securely managing API keys

2.1.2 Agent Definition. Next, the conversational agent was defined using the `Agent` class from the OpenAI Agents SDK, with a set of instructions (prompt) and using `gpt-4o-mini` as the base model, as shown in the code snippet below.

```
name="Assistant"
```

```
instructions="You're a Requirement Engineer working in a
software company. You are required to elicit requirements from
a client about his latest product idea. Engage in a friendly
conversation with the client and extract as much as
requirements you can based on his product idea. Follow the best
practises of requirement elicitation interviews. Always respond
in English.",
model="gpt-4o-mini"
```

2.1.3 Voice Pipeline Configuration. The project uses `VoicePipeline` and `SingleAgentVoiceWorkflow` classes from the `openai-agents[voice]` package. These components allow integration of voice input and output with the defined agent.

```
from agents.voice import SingleAgentVoiceWorkflow, VoicePipeline
workflow = SingleAgentVoiceWorkflow(agent)
pipeline = VoicePipeline(workflow=workflow)
```

The OpenAI's SDK is handling the text and speech elements through `VoicePipeline`, in a black-box environment². On the positive side, this eliminates the need for integrating external libraries for speech/text synthesis.

2.1.4 Recording and Interaction Loop. The interaction loop was implemented asynchronously to ensure that audio playback, transcription, and model inference occurred without blocking one another. This is achieved in the order of:

- The user's speech was recorded from the system microphone and converted to a `numpy` array.
- The recording was wrapped as an `AudioInput` object and submitted to the voice pipeline.
- The loop allowed for real-time interruption using the `Enter` key. A background thread monitored for the key press, which could stop playback and return the system to listening mode without terminating the session.

2.1.5 Integrating Speech Transcription. To include accurate textual representations of spoken input, the recorded audio was processed through OpenAI's Whisper transcription model (`gpt-4o-mini-transcribe`). The model transcribed each utterance prior to model inference, allowing the log to contain both user speech and corresponding text.

2.1.6 Conversation Logging and Data Management. Each conversation turn, comprising the user's transcribed text, the assistant's generated response, and a timestamp was logged. The log was organized in date-based subfolders and served as the complete record of dialogue history.

An excerpt of a sample conversation is shown below:

```
18:42:11
User: "What is the capital of Italy?"
Assistant: "The capital of Italy is Rome"
```

In addition, for empirical evaluation and reproducibility, the system was extended to record audio data for each turn. Each full interaction (the user's speech and the assistant's spoken reply concatenated) was saved as a single `.wav` file. This design choice supported the analysis of requirements coverage and usability comparison later.

²<https://openai.github.io/openai-agents-python/voice/quickstart>

2.1.7 Challenges & Observations. OpenAI's SDK offered *limited customization* options for the text-to-speech (TTS) functionality. Parameters such as speaking rate, pitch, and tone were not directly accessible to developers, making it difficult to customize the voice or adjust it for specific accessibility needs. A practical limitation of the OpenAI voice agent was the inability to handle the LLM inference and speech synthesis in-house or locally, which can cause a privacy concern, when eliciting software requirements. The debugging process within the Agents SDK was also somewhat opaque. When the voice pipeline encountered issues, such as failed audio streaming or timeouts during processing, the error messages were often generic, making them *hard to debug*.

2.2 Voice Agent 2: Using Gemma3:27b

In this section, we describe the development process of a local, speech-enabled voice assistant powered by the Gemma 3:27B.

2.2.1 Design Decisions and Rationale. We chose to *locally run* the speech-to-text (STT) and language model inference to ensure user privacy of data and low latency. During initial testing, the STT output was not heard (although generated and logged), when `pyttsx3` library is used. Therefore, macOS `say`, a *system-native TTS* command was chosen over third-party libraries like `pyttsx3` for guaranteed reliability and more natural speech. The design avoids cloud APIs or GUI-based tools, making the setup *portable and reproducible* across macOS systems. Markdown logs and timestamped audio archives provide a *traceable history* for debugging, analysis, and evaluation of the outputs. While not using full LangChain pipelines, the modular LLM wrapper is *LangChain compatible* allows easy future substitution of models (e.g., Qwen or LLaMA).

2.2.2 Speech-to-text with Whisper. The first stage records the user's speech and converts it into text. We used `sounddevice` library for microphone input and the open-source `faster_whisper`³ model for transcription. This model provides a good balance between speed and accuracy in local environments without relying on external APIs.

2.2.3 Language Model Integration. A local inference API was configured to serve the Gemma3:27B model, wrapped within a simple LangChain compatible interface. The wrapper enables sending text prompts and receiving model responses through an HTTP endpoint. Although the current implementation does not employ full LangChain chains, the design remains compatible with LangChain's `invoke()` interface for future scalability.

The prompt includes both the system instructions, behavioral constraints and the user's transcribed text, in order for the agent to remain context-aware and consistent in tone.

```
system_prompt = "You're a Requirement Engineer working in a
software company. You are required to elicit requirements from
a client about his latest product idea. Engage in a friendly
conversation with the client and extract as much as
requirements you can based on his product idea. Follow the best
practises of requirement elicitation interviews. Always respond
in English in conversation style. Do not include any links,
external sources or emojis and icons. Just text limit to 50
words.",
```

³<https://github.com/SYSTRAN/faster-whisper>

Component	OpenAI Agent	Gemma Agent
STT	Cloud (Whisper API)	Local (faster-whisper)
LLM	Cloud hosted	Local model
TTS	Cloud based	Local engine
Privacy	External processing	Fully local
Latency	Network dependent	Hardware dependent
Setup	Lightweight	Resource intensive

Table 1: Comparison of voice agent pipeline architectures

2.2.4 Text-to-speech Output with macOS System Voice. The language model’s textual response is converted back into audible speech using macOS’s built-in *say* command. To ensure consistent playback, we employed a fallback mechanism that saves the synthesized audio to a temporary file and plays it using *afplay*. This method proved to be more reliable than earlier attempts using the *pyttsx3* library, which exhibited inconsistent behavior on macOS.

2.2.5 Conversation Logging and Audio Archival. Each user utterance and model response is timestamped and appended to a markdown (.md) file. Audio recordings for both the user’s input and the assistant’s response are also saved. All logs and recordings are automatically organized in date-based subfolders.

For reproducibility and debugging, each conversation turn is also stored as a combined audio file containing both the user’s and assistant’s speech, using *soundfile* and *scipy* for basic signal concatenation.

2.2.6 Challenges & Observations. The initial implementation using the *pyttsx3* library caused silent playback on macOS due to engine driver issues. This was resolved by adopting the system-native *say* and *afplay* commands, which guarantee audible output. Early versions included a keyboard-based interruption system for halting TTS playback. This feature was later removed to simplify control flow and improve stability during speech generation. Combining user and assistant audio required consistent sampling rates between files. This was addressed using resampling functions from the *scipy.signal* package. Some delay between displaying the text and hearing the response was observed. The fallback TTS mechanism ensured that playback began only after audio generation completed, keeping speech synchronized with logs.

3 Evaluation Methodology

This evaluation was conducted as a pretest to identify faults, refine the implementation, and validate the functionality of the two voice agents before a larger-scale study. The primary objective was not to achieve statistical generalization but to observe practical behavior, usability, and conversational accuracy of the two systems in a typical requirement elicitation context.

A total of five users participated in the study, producing ten conversation sessions in total. Of the five participants, two were female, and all had a software development background and were currently working in software engineering academia. Two case studies were prepared to represent different business domains, enabling evaluation across varied contexts. The case studies are provided in Appendix A and consist of software application ideas based on day-to-day needs.

For each case study, the first author, who has prior professional experience as a business analyst, derived a finite list of requirements, referred to as *gold requirements*. These requirements were defined by considering essential functional features, non-functional constraints, and explicit constraints that an ideal requirements elicitation session would uncover. In total, eight gold requirements were specified for each case study.

Each participant completed two testing rounds, one with each agent, with each round lasting approximately four to six minutes. At the beginning of each session, the first author provided a brief overview of the study and instructions to the participant. The participant then randomly selected one case study and role-played as a company representative describing the application idea to the assigned agent. The agent used in each session was also randomly selected by the author. Participants were not informed which agent they were interacting with in order to minimize potential bias.

All conversations were conducted entirely through speech. The agent was expected to guide the interaction by asking clarifying questions and attempting to elicit as many relevant requirements as possible. A session ended when the agent no longer progressed with meaningful questioning. Throughout each session, the first author observed and documented agent behavior, response accuracy, and conversational flow using a structured evaluation form.

The evaluation form, included both quantitative and qualitative components. After the session ended, the author checked for conversation log of that session (and recorded audio, if needed) and performed manual semantic alignment against the gold requirement set. A requirement was considered captured when the agent’s responses conveyed the same functional intent or constraint as defined in the gold requirements. Finally, taking previously written gold requirements as the reference list, *Coverage* (captured requirements as a percentage of gold items) was derived. It was calculated using the below formula :

$$\text{Coverage (\%)} = \frac{\text{Captured Requirements}}{\text{Gold Requirements}} \times 100$$

At the end of each session, the participant completed the usability section of the evaluation form, rating the session’s *naturalness*, *helpfulness*, and *overall satisfaction* on a five-point scale. Simultaneously, the observing author added notes summarizing the interaction, including the agent’s responsiveness, fluency, and control of the conversation.

4 Pre-test Results

4.1 Voice Agent 1: Using GPT-4o-mini

4.1.1 Requirement Coverage. The recorded evaluation forms were examined to calculate the coverage of requirements captured by the OpenAI-based voice agent. Coverage represents the proportion of gold requirements correctly mentioned or identified by the agent during each session. The final values were calculated using the formula in previous section 3.

Table 2 presents the gold and captured requirement counts extracted from the five evaluation sheets (participants LK1, LK4, LK5, LK7, and LK10).

The OpenAI agent achieved an average coverage of approximately 77.5%, indicating that it successfully captured most of the

Participant	Gold Requirements	Captured	Coverage (%)
LK1	8	6	75.0
LK4	8	7	87.5
LK5	8	5	62.5
LK7	8	5	62.5
LK10	8	8	100.0
Average	–	–	77.5

Table 2: Requirement coverage scores for the OpenAI-based agent

relevant requirements discussed during the user sessions. The individual results show moderate variation between participants, ranging from 62.5% to full coverage (100%). This variation appears to reflect differences in participant articulation of the case study, their accent and conversational pacing rather than major inconsistencies in the agent's ability to identify relevant content.

4.1.2 Usability Ratings. The following section presents the usability ratings and behavioral observations for the OpenAI SDK-based voice agent. Five evaluation sheets (participants LK1, LK4, LK5, LK7, and LK10) were analyzed.

Table 3 summarizes the participant ratings and the calculated averages across all five sessions.

Participant	Naturalness	Helpfulness	Satisfaction	Mean Usability
LK1	3	4	4	3.7
LK4	4	3	4	3.7
LK5	4	4	4	4.0
LK7	4	4	3	3.7
LK10	4	5	5	4.7
Average	3.8	4.0	4.0	4.0

Table 3: Usability ratings for the OpenAI-based agent

The OpenAI-based agent achieved an average usability score of 4.0 out of 5, indicating consistently high user satisfaction and stable conversational performance. Ratings across participants were relatively close as well.

4.1.3 Behavioral Observations. The notes from the evaluation forms describe the OpenAI agent as generally more coherent, responsive, and natural in conversation compared to its open-source counterpart. An example dialogue starts as below:

```

12:01:07
User: "Hi, I need to develop a mobile app for LaundryMart."
Assistant: "Hi there! That sounds like a great idea. I would love to help you refine your concept for the laundromat mobile app. To start off, could you tell me what specific features you have in mind for the app? For example, are you thinking about features like scheduling wash times, tracking laundry status, or something else?"
12:02:33
User: "Everything"
Assistant: "Got it! Let's break this down a bit to ensure we capture all the details. Here are some areas we can explore:
1. User Features: .."

```

Participants frequently commented that the dialogue felt "almost human-like", that the agent maintained context effectively throughout the session (LK5 and LK7). It was observed to follow

the user's ideas logically and ask relevant follow-up questions without losing track of previous turns. Some participants mentioned minor limitations, such as the agent occasionally giving long or instruction-heavy responses (LK1) and not always recognizing conversational pauses (LK4 and LK10). Despite this, the majority of users noted that the system's questioning and clarification style was smooth. The voice delivery was considered clear and professional, though not fully human in tone. One participant (LK10) specifically noted that the agent's speech could be improved through better intonation and less monotony. There was also a suggestion to add keywords like "stop" to stop listening and recording, rather than having a 15 seconds of fixed duration of recording.

The observing author's notes highlight that the OpenAI agent displayed moderate to strong lead in elicitation, offering clarifying questions, summarizing key points, and suggesting additional ideas. Latency was not reported as an issue in any of the sessions, and the agent successfully produced brief summaries or conclusions toward the end of several conversations which were recorded in the logged transcript.

4.2 Voice Agent 2: Using Gemma3:27b

4.2.1 Requirement Coverage. The recorded evaluation sheets for the Gemma-based open-source agent were analyzed to determine how many predefined requirements were correctly captured during the test sessions. Table 4 summarizes the total and captured requirements as recorded in each evaluation form (participants LK2, LK3, LK6, LK8, and LK9).

Participant	Gold Requirements	Captured	Coverage (%)
LK2	8	2	25.0
LK3	8	3	37.5
LK6	8	3	37.5
LK8	8	3	37.5
LK9	8	3	37.5
Average	–	–	35.0

Table 4: Requirement coverage scores for the Gemma-based agent

The Gemma agent achieved an average coverage of approximately 35%, indicating that it captured roughly one-third of the total gold requirements per session. Coverage scores were relatively consistent across participants, showing limited improvement between sessions.

4.2.2 Usability Ratings. This section presents the usability ratings and observed conversational behavior of the Gemma-based voice agent as recorded in the evaluation forms. A total of five evaluation sheets were collected (participants LK2, LK3, LK6, LK8, and LK9).

Table 5 shows the individual ratings and the computed averages across the five sessions.

Across all sessions, the Gemma agent achieved an average usability score of approximately 3.3 out of 5. Naturalness and satisfaction ratings showed wide variation among participants, ranging from 1 to 5. This reflects differences in perceived quality of conversation flow and voice delivery across sessions.

Participant	Naturalness	Helpfulness	Satisfaction	Mean Usability
LK2	5	5	5	5.0
LK3	1	1	1	1.0
LK6	3	4	3	3.3
LK8	2	2	2	2.0
LK9	3	4	5	4.0
Average	2.8	3.2	3.2	3.3

Table 5: Usability ratings for the Gemma agent

4.2.3 Behavioral Observations. The evaluation notes indicate that the Gemma agent exhibited a mixed conversational performance. An unedited example dialogue starts as below:

```
11:47:53
User: "Hi, I want to develop mobile application for laundry
      cleaning clothes."
Assistant: "Hi there! That sounds interesting! So, a mobile app
           for laundry services. Great! To start, could you tell me a bit
           about *who* your target users are? Are we thinking students,
           busy professionals, or everyone?"
11:48:26
User: "Everyone from the city different cities uses"
Assistant: "Hi there! That's an interesting start! Could you
           elaborate? When you say "everyone from different cities uses,"
           what exactly are they using? What problem does this solve for
           them, and what are their main goals?"
```

In some cases, such as with participants LK2 and LK9, the dialogue was described as natural and helpful. The agent asked questions that guided participants to explain their ideas clearly. However, other participants (LK3, LK6, and LK8) reported difficulties, such as repeated or redundant questions and a loss of conversational context. These users reported that the agent sometimes forgot previous responses and asked for the same information multiple times (e.g., target user group). Several participants also mentioned the quality of the speech output. In some sessions, the voice was described as "robotic" or "mechanical", though one participant (LK9) noted that sentence endings sounded more natural and human-like. The observing author noted that the agent tended to start responses every time with simple greetings, such as "Hi there", which were not specified in the prompt and often disrupted the conversational flow.

Overall, the Gemma agent demonstrated the ability to ask clarifying questions and maintain short conversations but lacked consistent memory handling and smooth turn-taking. While one participant rated the interaction highly natural and satisfying, others perceived it as rigid and repetitive.

4.3 Threats to Validity

As a small-scale pretest, this study carries several threats to validity. Internal validity may be affected by potential learning or carryover effects since participants engaged in multiple sessions, although randomization of agent and case study order helped mitigate this. External validity is limited by the small sample size and the use of only two case studies, restricting generalization beyond the tested contexts. However, for a formative validation exercise, this level of control was sufficient to expose technical and behavioral issues in both architectures. Construct validity may also be influenced by subjective participant ratings and author's judgments during evaluation annotation. Standardized rating criteria and structured

observation forms were used to reduce this subjectivity. Despite these limitations, the pretest successfully fulfilled its purpose of validating the agents' functional reliability and identifying areas for improvement before further empirical evaluation.

5 Discussion

Traditional voice systems struggled in understanding the natural language context because they were not robust to dialogue deviations or complex queries outside a narrow script [4, 6, 21]. Generative LLM-based voice agents, by contrast, can be equipped with substantial domain knowledge and exhibit the flexibility needed for unscripted interactions.

For years, humans did not prefer machine-generated voice systems, for they are susceptible for errors leading to disillusionment or frustration [6] caused by the slow progress of underlying technologies. Interestingly, Allbert et al. made an observation that factors other than technical performance can influence the user experience in voice-based AI systems [2], opening a new hope for voice based research with emergence of accessible AI techniques.

In this experience report, we present our approach on building voice agents as "requirements elicitors" using two large language models, OpenAI's GPT-4o-mini and Gemma3:27b, in the context of requirements elicitation rounds in a typical software project. Our end goal is to use this "elicitor" agent as the first agent in an AI driven multi-agent requirement engineering workflow. We conducted a pre-testing round with five participants on the behavior of the agents. The comparative pretest results revealed clear distinctions between the two voice agents in terms of requirements coverage and user experience. The OpenAI SDK-based agent achieved an average coverage of 77.5%, while the Gemma-based open-source agent reached 35.0% of total requirements in these test rounds. These values reflect a clear difference in the systems' ability to identify and retain relevant requirements during conversation. The higher coverage of the OpenAI agent indicates that it more effectively extracted and summarized information from the participant dialogues, maintaining context across turns and producing coherent follow-up questions. In contrast, the Gemma agent often failed to capture several key requirements, which corresponds with the observations of context loss and repeated questioning recorded in the evaluation forms.

Across all sessions, the OpenAI SDK-based agent consistently achieved higher usability ratings, with an average score of 4.0 out of 5, while the Gemma-based open-source agent obtained an average score of 3.3. Participants generally observed the OpenAI agent as more coherent, natural, and responsive, showing better control of conversational flow and contextual understanding. It also asked clarifying questions and maintained topic continuity. In contrast, the Gemma agent's behavior was more variable across users, with several observations on repetition, context loss, and mechanical voice output, although some session logs showed that it could ask useful questions and sustain short, focused discussions.

Taken together, the usability and coverage findings show that the OpenAI agent provided a more complete and structured conversational performance, both in terms of perceived quality and actual requirement capture. The Gemma agent, although less consistent,

demonstrated partial success in identifying basic features and engaging in short question–answer exchanges. These differences can be attributed mainly to architectural maturity and system integration. The OpenAI agent benefits from an optimized end-to-end SDK with synchronized language, speech, and memory components, while the Gemma agent relies on modular open-source components with less stable context retention and voice synthesis.

Nevertheless, these findings confirm that both agents are functionally capable of supporting requirement elicitation, with the OpenAI system offering a more refined user experience and the Gemma implementation serving as a feasible local alternative for further experimentation and improvement.

6 Future Work

With the low test scores received for the Gemma model, we plan to employ other suitable models, such as Qwen, in future rounds of voice-based testing. As mentioned in Section 1, this voice agent will act as the first agent in a multi-agent requirements engineering workflow that aims to automate state-of-the-art requirements engineering tasks in a typical software engineering project. Furthermore, integrating multimodal inputs, such as images or text files, would extend the agent’s capabilities beyond speech, enabling richer and more context-aware interactions. We also plan to employ prompt engineering techniques, such as few-shot prompting, to mitigate model hallucinations in the next stage.

Future research will also include a wider sample of user studies to evaluate the effectiveness and user satisfaction of the two systems. Additional metrics, such as task completion time, error rate, and hallucination frequency, could provide more empirical feedback.

7 Related Work

System Architecture of Modern Voice Agents

Modern generative voice agents can be implemented using either end-to-end (E2E) systems or collaborative systems [21]. E2E systems are based on speech-text multimodal models that directly simulate speech-to-speech interactions (e.g., OmniFlatten [21], Moshi [10]). Models used in these systems include Qwen2-0.5B⁴ and Helium [10]. In contrast, collaborative systems integrate external automatic speech recognition (ASR) or TTS modules with LLM-based conversational components. For example, Qwen2-audio, a Large Audio-Language Model (LALM), processes both audio and text inputs to generate textual outputs [5]. Such pipeline-based approaches allow the use of state-of-the-art specialized models at each stage (e.g., Whisper for transcription accuracy [3]) but are susceptible to error propagation across components [15]. This limitation is particularly relevant in requirements elicitation, where misinterpretation of stakeholder statements can intensify known communication and ambiguity problems [8].

Recent work increasingly investigates end-to-end models that directly map audio input to audio output. Allbert et al. [2] compared STT–LLM–TTS stacks using data from over 300,000 AI-conducted interviews and found that combinations such as Google STT, GPT-4.1, and Cartesia TTS achieve superior quality and user satisfaction. For voice-based workflows, latency and response time are critical selection factors [18], as delays disrupt conversational flow and

hinder iterative clarification. Benchmarked models in this regard include GPT-4, Gemini 2.5, LLaMA 3.3, and models from the Qwen series [18]. Beyond pipeline performance, recent studies also examine LLMs’ reasoning and audio comprehension abilities when functioning as voice assistants [15, 16], as well as users’ perceptions of generative voice within social interactions [13].

Real World Applications

Generative voice agents have moved from laboratory settings into real-world use across multiple domains. In healthcare, they are applied to appointment scheduling, insurance queries, and medication refills to reduce staff workload [1]. Busch et al. [4] show that GPT-4o can accurately transcribe radiological terminology for CT and MRI reporting in German and English. In media and entertainment, Artioli et al. [3] present multilingual voice dubbing that preserves speaker characteristics, while Galuszka [12] reports on community-driven voice cloning. Voice agents are also widely used in customer service and personal assistant scenarios, including applications where agents act as conversational proxies for users [13].

Recent research in RE has investigated the use of LLMs to support requirements related tasks through natural language processing. For example, prior work has explored whether LLM-generated user stories can serve as creativity triggers in stakeholder workshops [17], and whether informal natural language intent can be transformed into precise, checkable specifications [11]. These studies illustrate the potential of LLMs to support the interpretation and structuring of stakeholder input; however, they focus primarily on textual artifacts rather than spoken, interactive elicitation.

Wherever natural spoken interaction adds value, voice agents are beginning to have practical impact. Nevertheless, there is limited research on their role in structured communication tasks such as software requirements elicitation, where requirements must be obtained from stakeholders prior to system development. Prior RE studies emphasize that elicitation is highly sensitive to ambiguity, communication breakdowns, and mismatches between stakeholder intent and formalized requirements [8]. While recent work shows that LLMs can assist in generating and formalizing requirements-related artifacts from natural language [11, 17], these approaches have not been investigated in free, open-ended spoken interaction settings. This work takes a step toward addressing this gap by examining the benefits, limitations, and potential enhancements of LLM-based voice agents for requirements elicitation use cases.

Code Availability

The voice agent prototypes developed for this study are part of an ongoing development effort and are not publicly released at this stage. A cleaned and documented public repository will be made available in a future stage of the project.

Declaration of Generative AI Usage

Generative AI tools were used for debugging during the development phase and editing the quality of the text in this manuscript. Afterward, the authors thoroughly reviewed the code and content to the best of their ability.

⁴<https://huggingface.co/Qwen/Qwen2-0.5B>

Acknowledgments

This work has been supported by FAST, the Finnish Software Engineering Doctoral Research Network, funded by the Ministry of Education and Culture, Finland.

References

- [1] Scott J. Adams, Julián N. Acosta, and Pranav Rajpurkar. 2025. How generative AI voice agents will transform medicine. *npi Digital Medicine* 8, 1 (june 2025), 353. doi:10.1038/s41746-025-01776-y
- [2] Rumi Allbert, Nima Yazdani, Ali Ansari, Aruj Mahajan, Amirhossein Afshar-rad, and Seyed Shahabeddin Mousavi. 2025. Evaluating Speech-to-Text x LLM x Text-to-Speech Combinations for AI Interview Systems. *arXiv preprint arXiv:2507.16835* (2025).
- [3] Emanuele Artioli, Daniele Lorenzi, Farzad Tashtarian, and Christian Timmerer. 2025. Generative AI for Realistic Voice Dubbing Across Languages. In *Proceedings of the 4th Mile-High Video Conference (MHV '25)*. Association for Computing Machinery, New York, NY, USA, 75–76. doi:10.1145/3715675.3715817
- [4] Felix Busch, Philipp Prucker, Alexander Komenda, Sebastian Ziegelmayr, Marcus R Makowski, Keno K Bressemer, and Lisa C Adams. 2025. Multilingual feasibility of GPT-4o for automated Voice-to-Text CT and MRI report transcription. *European Journal of Radiology* 182 (jan 2025), 111827. doi:10.1016/j.ejrad.2024.111827
- [5] Yunfei Chu, Jin Xu, Qian Yang, Haojie Wei, Xipin Wei, Zhifang Guo, Yichong Leng, Yuanjun Lv, Jinzheng He, Junyang Lin, Chang Zhou, and Jingren Zhou. 2024. Qwen2-Audio Technical Report. arXiv:2407.10759 [eess.AS] <https://arxiv.org/abs/2407.10759>
- [6] Philip R Cohen and Sharon L Oviatt. 1995. The role of voice input for human-machine communication. *proceedings of the National Academy of Sciences* 92, 22 (1995), 9921–9927.
- [7] Jane Coughlan and Robert D Macredie. 2002. Effective communication in requirements elicitation: a comparison of methodologies. *Requirements Engineering* 7, 2 (2002), 47–60.
- [8] B. Davey and K. Parker. 2015. Requirements elicitation problems: A literature analysis. *Issues in Informing Science and Information Technology* 12 (2015), 71–82. <http://iisit.org/Vol12/IISITv12p071082Davey1929.pdf>
- [9] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. 2018. Multi-Agent Systems: A Survey. *IEEE Access* 6 (2018), 28573–28593. doi:10.1109/ACCESS.2018.2831228
- [10] Alexandre Défossez, Laurent Mazaré, Manu Orsini, Amélie Royer, Patrick Pérez, Hervé Jégou, Edouard Grave, and Neil Zeghidour. 2024. *Moshi: A Speech-Text Foundation Model for Real-Time Dialogue*. Technical Report. Kyutai.
- [11] Madeline Endres, Sarah Fakhoury, Saikat Chakraborty, and Shuvendu K. Lahiri. 2024. Can Large Language Models Transform Natural Language Intent into Formal Method Postconditions? *Proc. ACM Softw. Eng.* 1, FSE, Article 84 (July 2024), 24 pages. doi:10.1145/3660791
- [12] Patryk Galuszka. 2025. The influence of generative AI on popular music: Fan productions and the reimagination of iconic voices. *Media, Culture & Society* 47, 3 (apr 2025), 603–612. doi:10.1177/01634437241282382
- [13] Angel Hsing-Chi Hwang, John Oliver Siy, Renee Shelby, and Alison Lentz. 2024. In Whose Voice?: Examining AI Agent Representation of People in Social Interaction through Generative Speech. In *Proceedings of the 2024 ACM Designing Interactive Systems Conference (Copenhagen, Denmark) (DIS'24)*. Association for Computing Machinery, New York, NY, USA, 224–245. doi:10.1145/3643834.3661555
- [14] Thomas K. Landauer. 1996. *Trouble with Computers: Usefulness, Usability, and Productivity*. MIT Press, Cambridge, MA, USA.
- [15] Yu-Xiang Lin, Chih-Kai Yang, Wei-Chih Chen, Chen-An Li, Chien-yu Huang, Xuanjun Chen, and Hung-yi Lee. 2025. A Preliminary Exploration with GPT-4o Voice Mode. (feb 2025). doi:10.48550/arXiv.2502.09940 arXiv:2502.09940 [cs].
- [16] Amama Mahmood, Junxiang Wang, Bingsheng Yao, Dakuo Wang, and Chien-Ming Huang. 2025. User Interaction Patterns and Breakdowns in Conversing with LLM-Powered Voice Assistants. *International Journal of Human-Computer Studies* 195 (2025), 103406. doi:10.1016/j.ijhcs.2024.103406
- [17] Agnieszka Marczak-Czajka and Jane Cleland-Huang. 2023. Using ChatGPT to Generate Human-Value User Stories as Inspirational Triggers. In *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. IEEE, Hannover, Germany, 52–61. doi:10.1109/REW57809.2023.00016
- [18] Softcery. 2025. *AI Voice Agents: Choosing the Right LLM*. <https://softcery.com/lab/ai-voice-agents-choosing-the-right-llm> Accessed: 2025-11-01.
- [19] Michael Wooldridge and Nicholas R. Jennings. 1995. Intelligent agents: theory and practice. *The Knowledge Engineering Review* 10, 2 (1995), 115–152. doi:10.1017/S0269888900008122
- [20] Dayong Ye, Minjie Zhang, and Athanasios V Vasilakos. 2016. A survey of self-organization mechanisms in multiagent systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47, 3 (2016), 441–461.
- [21] Qinglin Zhang, Luyao Cheng, Chong Deng, Qian Chen, Wen Wang, Siqi Zheng, Jiaqing Liu, Hai Yu, Chao-Hong Tan, Zhihao Du, and ShiLiang Zhang. 2025. OmniFlatten: An End-to-end GPT Model for Seamless Voice Conversation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 14570–14580. doi:10.18653/v1/2025.acl-long.709

A Appendix

A.1 Case study 1:

A growing chain of self-service laundromats, plans to develop a mobile application to improve customer convenience and optimize machine usage across its locations. Currently, customers must physically visit the laundromat to check machine availability, often leading to long waits or wasted trips during peak hours. The proposed mobile app will allow users to locate nearby branches, view real-time washer and dryer availability, and reserve machines in advance. Customers will be able to create accounts, load credit through secure payment methods, and start machines via QR code scanning, eliminating the need for cash or physical tokens. The app should also provide estimated wash and dry completion times, with push notifications when a cycle is nearly finished. Users can receive reminders to retrieve clothes and report machine issues directly through the app, with options to upload photos or comments for better issue tracking. A loyalty system will reward frequent users with discounts or free cycles. Promotional notifications and seasonal offers should be manageable via the app's admin panel. Customers should also have access to usage history and receipts for expense tracking. From an operational side, laundromat managers require a dashboard to monitor machine usage patterns, service requests, and peak traffic periods. They should be able to adjust availability settings, manage promotions, and access user feedback.

A.2 Case study 2:

A digital health startup, is developing a mobile application to help women track their menstrual cycles and gain personalized insights into their reproductive and overall health. Many existing solutions lack accuracy, personalization, or inclusivity for diverse health conditions and goals. The startup aims to address these gaps through a science-backed, user-friendly platform. The app will allow users to log menstruation dates, flow intensity, mood changes, physical symptoms, and lifestyle factors like sleep, stress, and physical activity. Based on this data, it will generate cycle predictions (menstrual, ovulation, and fertility windows), symptom patterns, and wellness tips tailored to the user's profile. Users can receive daily notifications with health insights, medication reminders, and cycle stage updates. The app will offer educational content on menstrual health, contraception, fertility, and hormonal wellness. Integration with wearables and health apps will enhance tracking by importing data such as body temperature, heart rate, and activity levels. Privacy is a central concern, so the app must provide strong data encryption, clear consent options, and the ability for users to control what data is shared or stored. A customizable interface will allow users to personalize tracking categories, themes, and notification preferences. For administrators, a dashboard will support content updates, trend analytics, and moderated community interactions. The app should also provide anonymized data insights for public health research, with explicit opt-in from users.