

System Design and Usability Evaluation of an LLM-based Voice Assistant for Controlling Digital Environments

Alexandru Gheorghiu

Ștefan cel Mare University of Suceava
Suceava, Romania
alexandru.gheorghiu@student.usv.ro

Laura-Bianca Bilius

MintViz Lab, MANSiD Center,
Ștefan cel Mare University of Suceava
Suceava, Romania
laura.bilius@usm.ro



Figure 1: Interaction workflow of the voice assistant for controlling digital environments organized into three sequential stages: (1) Listening, (2) Processing, and (3) Speaking. Note. For example, the user says “Open Firefox.” The voice assistant converts the speech into text, identifies the user’s intent, executes the corresponding action, and communicates the outcome through synthesized speech.

ABSTRACT

Recent advances in artificial intelligence have significantly improved computers’ ability to understand natural language, enabling more intuitive and natural interactions. In this context, this paper introduces a local-first, speaker-aware voice assistant for controlling digital environments that integrates speech recognition, speaker identification, large language model reasoning, automation across digital environments, and text-to-speech within a modular multi-process architecture. To support safe interaction, the voice assistant incorporates a risk-aware validation layer that classifies commands according to their potential impact and requires explicit user confirmation before executing potentially harmful actions. We evaluated the proposed voice assistant through a pilot usability study involving seven participants performing representative desktop tasks. Preliminary results indicate positive usability, a favorable user experience, and a 95.5% overall tool invocation success rate, suggesting that local-first voice assistants can provide an effective, accessible, and privacy-preserving approach to digital interaction. Based on these findings, we discuss future application opportunities in accessible computing and assistive technology.

AUTHOR KEYWORDS

Voice Assistants; Voice User Interfaces; Desktop Automation; Digital Environment Control.

INTRODUCTION

Voice assistants enable users to interact with computer systems through natural language [16], also representing an alternative input modality over conventional input devices for users with impairments [21,24]. Recent advances in speech recognition, large language models (LLMs), and speech synthesis have considerably expanded the capabilities of voice interfaces, allowing them to interpret open-ended requests and support increasingly complex interactions beyond predefined command sets [7,27]. As a result, voice interaction has moved from research prototypes into everyday life through commercial assistants such as Amazon Alexa, Apple Siri, and Google Assistant, which have popularized speech as an input modality for information retrieval and device control [9].

Interacting with digital environments often involves sequences of actions spanning applications, files, windows, web browsers, and system utilities. Designing such systems raise challenges and requires balancing technical capabilities with interaction design considerations [31]. As voice assistants increasingly perform actions on behalf of users, they need to provide clear feedback, make their actions understandable, and ensure that control ultimately remains with the user [2]. This aspect is particularly important for users with physical impairments, who may encounter additional challenges when interacting with computer systems. At the same time, privacy has become a central design consideration because many voice assistants process user speech on cloud infrastructure [18]. Moreover, voice-based interaction may also provide an alternative input modality for people with motor impairments, who may experience difficulties using

DOI: 10.37789/icusi.2026.25

conventional mouse and keyboard input [6]. Building on these considerations, we introduce in this paper an LLM-based voice assistant for controlling digital environments. Accordingly, our contributions are as follows:

- (1) A local-first voice assistant architecture for controlling digital environments that integrates speech recognition, speaker identification, LLM-based reasoning, explicit risk-aware command validation, automation across digital environments, and speech synthesis within a modular multi-process architecture.
- (2) A pilot usability study evaluating the proposed voice assistant through subjective user feedback and objective performance metrics.

RELATED WORK

This section reviews prior work relevant to the proposed voice assistant, including advances in speech and language technologies, voice-based desktop assistants for digital interaction and automation, and accessibility considerations for voice input.

Speech and Language Models

The capabilities of LLMs have expanded from text generation toward structured and multi-step problem solving [17], where the model iteratively proposes, evaluates, and refines candidate solutions instead of returning a single answer [15]. These advancements have enabled LLMs to serve not only as conversational interfaces, but also reasoning engines capable of planning actions and coordinating external tools, making them well suited for interacting with and controlling user interfaces and systems. Speech technologies complement LLMs by supporting spoken input and synthesized responses. Recent advances have enabled fast and natural voice interaction [4,12,26]. On the input side, recent advances in self-supervised and large-scale learning have significantly advanced automatic speech recognition (ASR). For example, wav2vec 2.0 learns representations from unlabeled audio [4], and Whisper demonstrates that large-scale weakly supervised training enables accurate, multilingual, and noise-robust transcription without task-specific fine-tuning [22]. Speaker recognition has advanced alongside ASR, enabling systems to identify not only what is being said but also who is speaking. Recent speaker-embedding models, from x-vectors [26] to TitaNet [12], transform short speech segments into compact embeddings that can be compared using similarity metrics for speaker verification and identification, making user-specific personalization possible in shared computing environments.

Voice Assistants

Recent work on voice assistants has increasingly explored voice as the primary interaction modality for controlling digital environments [7,10,13,27]. For example, Dobriyal *et al.* [7] introduce Sara, an AI-driven personal desktop assistant that pairs speech recognition with task automation and third-party APIs to streamline routine work. Another approach is AI-Based Desktop VIZ, which integrates an OpenAI API-based language model with PyAutoGUI automation library, enabling it to control mouse and keyboard input and interact with arbitrary desktop application windows [27]. The ZIVA desktop assistant [13] provides a Python-based voice interface for controlling Windows through spoken commands to operate their machine, replacing manual input for routine tasks. Similarly, Leo [10] is a hands-free assistant that uses voice

commands for common desktop tasks, including messaging, media playback, and screenshots.

Accessibility of Voice Input for Users with Impairments

Voice interaction has also been explored as an alternative to conventional touch, mouse and keyboard input, particularly in accessibility contexts [3,21,23,24]. For example, Pradhan *et al.* [21] found that voice assistants can support greater independence for users with disabilities in everyday activities, while speech and hearing impairments can present accessibility barriers. Voice-based programming has been investigated for users with motor impairments, *e.g.*, Rosenblatt *et al.* [24] developed VocalIDE for speech-based code navigation and editing. Beyond programming, Aziz *et al.* [3] explored speech-based manipulation of graphical objects, while Reicherts *et al.* [23] found that voice-based interaction in collaborative data analysis led to more system interactions, conversational turns, and questions.

To the best of our knowledge, existing voice assistants for digital environment control do not jointly support speaker-aware personalization and explicit multimodal confirmation for high-risk actions. In the following sections, we introduce a voice assistant for controlling digital environments that adopts a local-first architecture with speaker-aware personalization and explicit risk-aware validation, ensuring that potentially harmful actions require user approval before execution.

SYSTEM DESIGN AND ARCHITECTURE

This section presents the design principles and architectural decisions that guided the implementation of the proposed voice assistant. The proposed voice assistant is a modular, multi-process system comprising four components such as audio capture, speech understanding, reasoning and action, and speech synthesis. These components form a processing pipeline synchronized through a shared interaction state and coordinated by the main process, which hosts the graphical user interface and centralizes application logging. The architecture is local-first, with all speech processing performed on-device, while the language model may run locally or through a remote API. The LLM communicates with the operating system through a structured tool interface (*e.g.*, launching applications, managing windows, running shell commands, browsing and reading or writing files). All tool invocations are validated, and high-risk operations require explicit user confirmation before execution. Within a single interaction, the system identifies the speaker, interprets the request, executes the corresponding action, communicates the outcome, and returns to the listening state; see Figure 1. The supporting architecture is shown in Figure 2. The following sections describe in detail the main components of our system.

Audio Capture

The audio process captures the microphone at 16 kHz in 512-sample (32 ms) frames and applies a streaming Silero voice-activity detector¹ to decide, frame by frame, whether the user is speaking; see Figure 2. A short pre-buffer of 0.4 s is added to each detected speech segment to ensure that the beginning of a word is preserved. Speech is considered complete after 500 ms of trailing silence. To prevent the voice interface from transcribing its own output, a fundamental turn-taking

¹<https://github.com/snakers4/silero-vad>

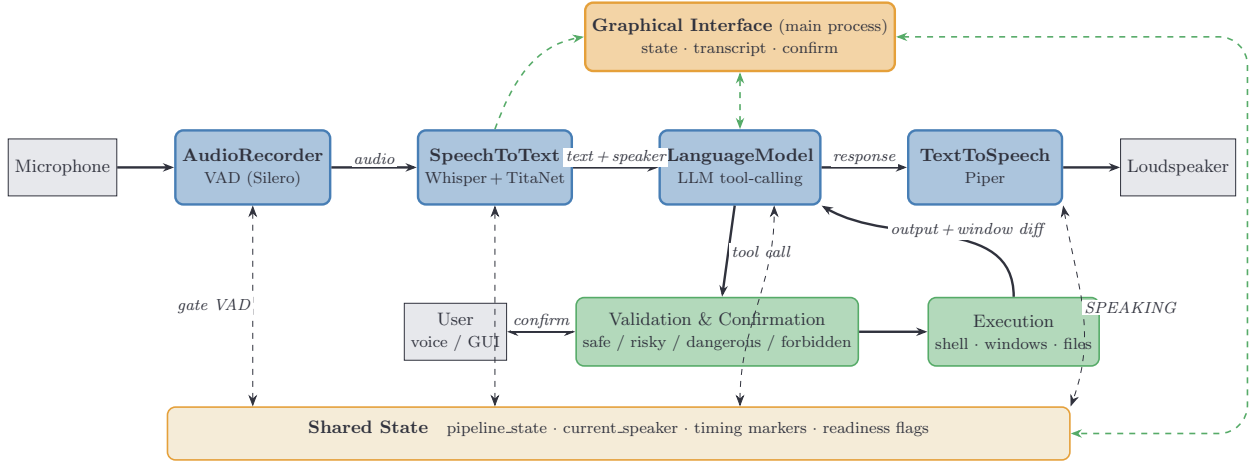


Figure 2: The architecture of the proposed voice assistant consisting of four concurrent stages connected by message queues and a shared interaction state, with a validate-gate-confirm-log safety layer around command execution.

problem [20], the pipeline maintains an explicit shared interaction state with three values: `LISTENING`, `PROCESSING` and `SPEAKING`. Specifically, immediately before audio playback begins, the synthesis stage sets the interaction state to `SPEAKING`. The audio capture process detects this change and temporarily suspends speech detection, preventing the voice assistant from transcribing its own output. Once playback is complete, a short post-speech delay of 0.35 s allows any residual audio from the loudspeaker to dissipate. The detector is then reset to clear its internal state, and the interaction state returns to `LISTENING`. The values of the pre-buffer duration, silence threshold, and post-speech delay were selected empirically based on iterative experimentation.

Speech Recognition

Each detected speech segment is transcribed by Faster-Whisper² (large-v3-turbo) [22] using GPU-accelerated quantized inference; see Figure 2. The language of each speech segment is detected independently and processed only when the confidence score exceeds a predefined threshold. Segments shorter than a minimum duration are discarded to avoid transcribing incidental noise.³ In parallel, speaker identification is performed using a TitaNet speaker-embedding model [12]; see Figure 2. Each speaker is enrolled offline by recording a short voice sample. A dedicated registration tool extracts and stores a compact embedding that represents the speaker’s voice. During operation, an embedding is computed for each detected speech segment and compared with the stored profiles using cosine similarity, the scoring method used by TitaNet model. If the highest similarity score exceeds a predefined threshold, the corresponding speaker is identified, otherwise, the speech segment is treated as originating from an unknown user.

Language Understanding and Execution

The transcribed request includes the speaker’s identity and a snapshot of the currently active application windows, providing contextual information about both the user issuing the request and the current state of the digital environment. This stage encapsulates an LLM behind a provider-agnostic interface and implements an iterative tool-calling loop based on reasoning-and-acting agent frameworks [25]. At each step, the model either responds to the user or invokes one or more tools. Following each tool invocation, the resulting output is returned to the model for evaluation, enabling it to determine whether additional actions are necessary or the requested task has been completed. The assistant provides tools that mirror common digital interactions. It can execute shell commands, open, close, focus, or resize application windows, open web pages, send keystrokes, query system information, and read or write files. Shell commands can be executed in three modes, such as a background mode that returns the output only to the assistant, a launch mode for starting graphical applications without waiting for completion, and a visible mode that runs commands in a terminal so the user can monitor their progress; see Figure 2. Beyond maintaining awareness of the currently open windows, the assistant can also perceive on-screen content when required. It achieves this by either extracting visible text using Tesseract OCR⁴ or by providing a screenshot to a vision language model. This capability enables the assistant to ground requests that depend on the displayed content, such as reading an error message or locating a visible interface element. Because screen perception is significantly more computationally expensive than interacting with the desktop directly, it is invoked only when the required information cannot be obtained through other means.

Safety Layer

Because the assistant can perform real actions, some of which are irreversible, trust and user control are treated as fundamental aspects of the interaction rather than implementation details. Before execution, each action is classified as safe, risky, dangerous, or forbidden using lists of known safe and dangerous commands, patterns for irreversible operations

²<https://github.com/SYSTRAN/faster-whisper>

³On standard benchmarks, the Whisper large-v3 (<https://github.com/openai/whisper>) model achieved a low word error rate (WER) for English, approximately 4.1% on FLEURS and 9.3% on Common Voice 15. Faster-Whisper reproduces the same transcription accuracy as the reference implementation while providing substantially faster and more memory-efficient inference.

⁴<https://github.com/tesseract-ocr/tesseract>

(*e.g.*, recursive deletion or disk formatting), and heuristics that detect attempts to bypass these checks through command chaining or piping. The use of four ordered levels follows the granularity of established risk-severity scales in safety and security engineering [29]. Commands classified as forbidden are rejected and never executed. In the example from Figure 1, launching Firefox is a safe action and proceeds without interrupting the user, whereas a request such as deleting a file is treated as dangerous. Whether a risky or dangerous action is executed depends on the speaker's permission level. When confirmation is required, the assistant pauses and requests explicit approval through either voice or the graphical interface, ensuring that the final decision remains with the user rather than the language model. Each executed command is logged with its risk level, exit status, output size, and execution time, providing an accountable record of the assistant's actions. This layered design, validate, gate, confirm, and log, follows the principle that autonomous systems should keep users in control; see Figure 2.

Speech Synthesis, Interface, and Instrumentation

System responses are synthesized locally using the Piper neural text-to-speech model⁵ and streamed to the loudspeaker; see Figure 2. During playback, the interaction state is set to `SPEAKING`, temporarily suspending microphone input to prevent the assistant from interpreting its own synthesized speech as a new command. In our example (see Figure 1), the assistant synthesizes the confirmation "Opened Firefox." Once playback is complete, the interaction state returns to `LISTENING`, and control of the conversation is handed back to the user. The current interaction state (`LISTENING`, `PROCESSING`, and `SPEAKING`) is displayed as a banner combining textual label and background color. The same state is mirrored as text in the window status bar, while a secondary status line reports the microphone condition and the most recent transcription. The interface further presents live speech transcriptions, assistant responses, and controls for muting the microphone, entering a command as text, and confirming pending actions, all accessible through standard keyboard. During execution, each process and component records structured logs and performance metrics, which are collected and stored for subsequent analysis.

Error Handling and Recovery

Because the pipeline is probabilistic, uncertain input is rejected before execution, *i.e.* speech shorter than 0.5 s are dropped as noise or with low language-identification confidence is discarded rather than forwarded to the language model. Also, speaker identification returns *unknown* when it cannot detect the speaker. In both cases the voice assistant stays silent and returns to `LISTENING`, so a rejected spoken input simply requires the user to repeat the request and does not produce an unintended action. When execution itself fails, the resulting error is returned to the language model as an explicit failure observation, *e.g.*, "Text 'Sign in' not found on screen" when a screen element cannot be located. Commands classified as risky require explicit confirmation, which can be provided by voice ("yes", "cancel", "stop") or by pressing a confirmation button on the interface. An unrecognized response triggers one additional prompt, and no answer within

30 s cancels the command by default, so the pipeline never blocks. Retained dialogue history allows users to correct misinterpreted requests conversationally, *e.g.*, "No, open it in VS Code instead." The assistant can also undo actions within the active application using synthesized keyboard input. Canceling, repeating, and correcting a request do not require mouse or keyboard input, but reverting a completed file operation requires a separate explicit instruction.

Overall, in developing the proposed voice assistant, we adopted Silero VAD, TitaNet, and Piper TTS because they are open-source, support local execution, and provide real-time performance. Silero VAD detects speech with minimal delay, TitaNet identifies speakers efficiently, and Piper TTS generates natural-sounding speech on the user's device.

PILOT USER STUDY

To evaluate the proposed voice assistant for controlling digital environments, we conducted a pilot user study focusing on desktop environments as a specific type of digital environment. In the following, we describe the study methodology, including experimental tasks and evaluation measures.

Pilot User Study Description

Seven participants (five male and two female), aged between 23 and 41 years ($M=28.7$, $SD=6.4$), were recruited through mailing lists and convenience sampling. All participants reported frequent use of both PC/laptop and smartphone. Overall, participants reported relatively infrequent use of voice assistants, with a mean usage frequency of 2.0 on a 5-point Likert scale; see Figure 3a. Similarly, their comfort with cloud-based speech processing was moderate to low ($M=2.6$), suggesting some concerns about cloud-based speech processing; see Figure 3b. Participants also revealed a preference for privacy in speech processing, therefore, more than half (57.1%) preferred all speech processing to be performed locally on the device, while 28.6% favored a hybrid approach in which cloud processing is used only when necessary. None of the participants preferred speech processing to occur entirely or primarily in the cloud, and 14.3% reported having no preference; see Figure 3 for additional details. The study was approved by the university ethics review committee.

Task

After completing the demographic questionnaire, participants performed three practice voice commands to familiarize themselves with the proposed voice assistant. Then they completed the experimental tasks T_1 - T_{15} listed in Table 1. The tasks were designed to cover desktop activity categories identified by Hazem *et al.* [1], including searching, browsing, reading, writing, interpreting, debugging, playing, and watching, while also assessing the proposed voice assistant's capabilities in file management, application control, and desktop automation. Finally, participants completed the SUS and UEQ+ questionnaires to assess the system's usability and user experience. All sessions were conducted on a laptop with an Intel Core i9-14900HX CPU, 64 GB of RAM, and an NVIDIA GeForce RTX 4070 Laptop GPU with 8 GB of VRAM, running Ubuntu 24.04 LTS with GNOME desktop under X11 session.

Design and Measures

We evaluated the proposed voice assistant for controlling digital environments from two complementary perspectives.

⁵<https://github.com/OHF-Voice/piper1-gpl>

Table 1: List of experimental tasks used to evaluate the proposed voice assistant for controlling digital environments.

No.	Voice command
T ₁	What is the current weather in <X>?
T ₂	Create a folder on the Desktop named "Notes <today's date>" and show me the folder.
T ₃	Open Firefox and search for paper <i>Attention Is All You Need</i> .
T ₄	Open the second result and read the title.
T ₅	Write the abstract of this paper to a .txt file and save it in the previously created folder.
T ₆	Open the .txt file.
T ₇	Create a Python file with Rock, Paper, Scissors game.
T ₈	Debug the generated Python code in Visual Studio.
T ₉	Play a game. Press the S key. Close the game.
T ₁₀	Search for image1.png on the Desktop and copy it to the previously created folder.
T ₁₁	Show me the folder.
T ₁₂	Search for a video about Rock, Paper, Scissors and open the first result.
T ₁₃	Close Firefox.
T ₁₄	Select the files from previously created folder and delete them.
T ₁₅	Delete all folders from the Desktop.

First, we assessed users' subjective perceptions of usability and user experience using the following subjective measures:

- M₁ *User Experience Questionnaire Plus* (UEQ+)⁶, using the Comprehensibility, Response Quality, Response Behavior, Intuitive Use, and Stimulation scales. We selected these scales based on prior work evaluating Alexa [11].
- M₂ *System Usability Scale* (SUS), 10-item questionnaire that captures users' perception of a system's usability [5].

Voice assistant performance was assessed using the following objective measures collected during the experimental tasks:

- M₃ *Execution Time*, measured as the completion time between the invocation of a tool and the completion of its execution, reported as the mean execution time per tool.
- M₄ *Success Rate*, defined as the proportion of successful tool invocations out of all tool invocations across all experimental tasks.

RESULTS

The results revealed a positive overall user experience across all selected UEQ+ dimensions (Figure 3f). Response Behaviour received the highest rating (M=4.4, SD=0.7), followed by Comprehensibility (M=4.3, SD=0.8), while Intuitive Use and Response Quality were also rated highly (M=4.2, SD=0.7 and M=4.2, SD=0.8, respectively). Stimulation received the lowest, but still positive, rating (M=3.9, SD=0.9). The proposed system achieved a SUS⁷ score of 78.9. Overall, these results indicate that participants perceived the proposed voice assistant as usable, intuitive, and engaging.

Table 2 summarizes performance metrics for 216 tool invocations recorded across the 15 experimental tasks (see Table 1). The number of tool invocations varied across participants because some commands were executed successfully on the first attempt, while others required one or more repetitions before successful completion. Execution times ranged from 0.2 ms (*write_file*) to 1747.5 ms (*click_text*), reflecting differences in operation type across three performance categories. The fastest operations are lightweight

Table 2: Execution performance of the desktop control tools invoked in our user study. For each tool, the table reports the number of invocations, average execution time, 95th-percentile latency, and success rate across all sessions.

Tool	No.	Avg. (ms)	p95 (ms)	Rate of success
<i>click_text</i>	14	1747.5	2099.0	92.9%
<i>close_application</i>	9	6.0	6.6	77.8%
<i>focus_window</i>	8	6.3	7.7	100%
<i>get_system_info</i>	7	56.0	68.2	100%
<i>open_application</i>	6	1100.1	1524.1	100%
<i>open_url</i>	19	573.7	1291.4	89.5%
<i>read_screen</i>	37	873.3	1313.3	100%
<i>run_shell</i>	87	241.1	904.3	93.1%
<i>set_window_state</i>	4	5.9	6.4	100%
<i>show_desktop</i>	2	2.6	2.7	100%
<i>type_text</i>	13	112.0	152.1	92.3%
<i>write_file</i>	10	0.2	0.3	100%
Overall	216	393.7	614.7	95.5%

local tasks, such as *write_file*, *show_desktop*, *close_application*, *focus_window*, and *set_window_state*, which execute synchronously and complete in under 7 ms without the overhead of launching external applications or interpreting on-screen content. The second category includes task-dependent operations. For example, the execution time of *type_text* increases with the amount of text (112 ms on average, 152.1 ms p95), whereas *run_shell* varies considerably depending on the executed command (241.1 ms, 904.3 ms p95). The slowest tools depend on external programs or screen perception. Opening an application or a web page requires waiting for the target application or browser to start and become available (*open_application*, 1100.1 ms or *open_url*, 573.7 ms). Similarly, *read_screen* (873.3 ms) must capture and interpret the display and *click_text* (1747.5 ms), the most expensive tool, must first locate the target text on the screen before acting; please see Table 2 for more details. Tool usage was concentrated on a few general-purpose tools, with *run_shell* accounting for 40.3% of all calls, followed by *read_screen* (17.1%) and *open_url* (8.8%).

Across all experimental sessions, 95.5% of tool invocations were successful. The failures that occurred were mostly associated with tools that must locate a specific on-screen element, either by matching an application title or by searching for visible text, e.g., *close_application* (77.8%), *open_url* (89.5%), *type_text* (92.3%) and *click_text* (92.9%). By contrast, several tools achieved a 100% success rate, such as *show_desktop*, *read_screen*, *write_file*, *get_system_info*, *set_window_state*, *focus_window*. Overall, the results indicate that the assistant is more likely to fail when determining where to act than when executing the action itself.

The qualitative feedback collected from our participants was generally positive and aligned with the quantitative results. Participants highlighted the voice assistant's ease of use (P₃), fast response time (P₄), and the versatility of the supported voice commands (P₅). They also described the system as providing a novel interaction experience and appreciated its ability to accelerate common desktop tasks through voice control (P₂, P₇). One participant, P₁ specifically appreciated the explicit confirmation mechanism for potentially harmful actions, noting that it increased their confidence in using the

⁶<https://ueqplus.ueq-research.org/>

⁷According to established SUS benchmarks [14], scores around 80 are generally interpreted as indicating good usability and strong user acceptance.

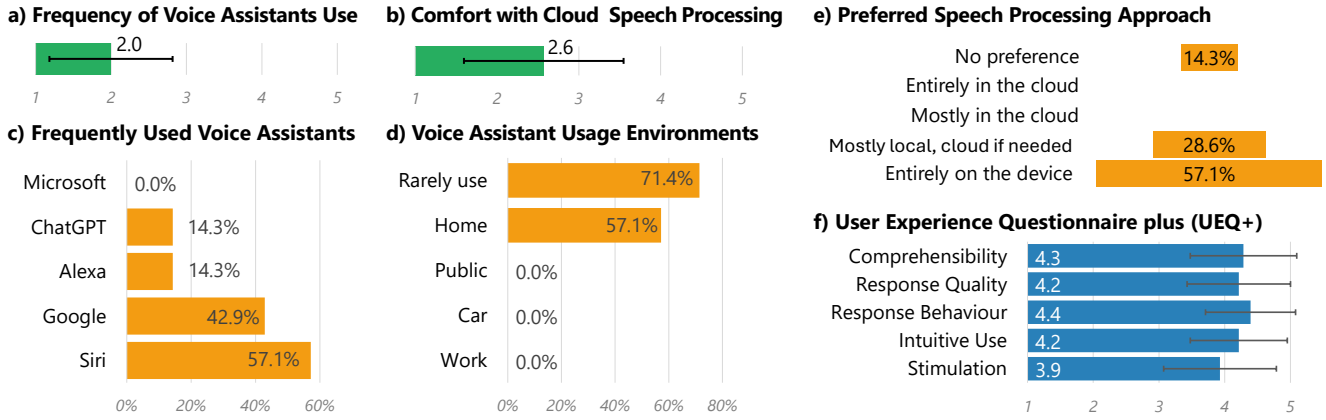


Figure 3: Participants' voice assistant experience, usage habits, privacy preferences (a-e), and UEQ+ ratings (f).

voice assistant because it prevented unintended or irreversible operations from being executed without their approval. Overall, their feedback suggests that users perceived the proposed voice assistant as both practical and intuitive.

DISCUSSION

This study has several limitations that provide directions for future work. First, our evaluation covered a limited range of speech patterns. Although none of the participants reported speech or other physical impairments, we observed recurring cases in which spoken input was discarded before reaching the language model, most often because non-native English speech was misidentified as another language. The assistant may encounter difficulties when users change their intent mid-command, reformulate requests, or use regional dialects. Moreover, discarded spoken inputs currently receive no response, leaving users without clear feedback about the cause of the failure. Future work will therefore evaluate a broader range of speech patterns, while improving feedback for unrecognized input, speech recognition, and language understanding to better handle such variations.

The relatively small number of participants limits the generalizability of the findings. For example, our study did not include users with motor, speech, or visual impairments nor users of assistive technologies. Consequently, the system's accessibility was not evaluated directly for these user populations, and the empirical evidence we gathered is limited to indirect assessments of overall system usability through UEQ+ and SUS. However, users with speech impairments may experience additional recognition barriers, while users with motor or visual impairments may have different interaction and feedback needs that should be captured in future evaluations. Accessibility-oriented interaction design has been investigated for other input modalities in the scientific literature. Gesture-A11Y [28], for example, provides a large-scale repository of touch, motion, on-wheelchair, and on-body gesture records from users with visual and motor impairments to inform the design and evaluation of accessible gesture input. In this context, it could be extended to voice interaction by including speech data from users with various disabilities, supporting the design of more accessible voice interfaces. Moreover, Guriță and Vatavu [8] found that including accessibility requirements can improve the accessibility of LLM-generated user interfaces. However, they also mentioned that meeting

technical requirements is not enough on its own and accessible systems should also reflect real user needs, diverse data, and the context in which people interact with them. In this context, future work will include a larger and more diverse group of users with disabilities, including motor, speech, and visual impairments, while considering differences in speech abilities and speaking patterns when evaluating accessibility. Future work will extend the voice assistant to other digital environments and devices, such as smartphones and tablets, and compare voice-based interaction with conventional input methods across different usage scenarios, including touch-screen gesture-based input [19] and its examination in ability-oriented approaches [30] for more accessible interactions. Lastly, our evaluation focused on desktop interaction within a Linux-based GNOME environment and did not compare speech with conventional keyboard-and-mouse input, which we leave for future work as well.

CONCLUSION

This paper introduced a local-first, speaker-aware voice assistant for controlling digital environments that integrates LLM-based reasoning with explicit risk-aware command validation. The preliminary evaluation showed positive user perceptions, while the high success rate of tool invocations reflects the effectiveness of the evaluated desktop-control operations. These characteristics also highlight the potential of voice-based interaction as an alternative modality for users with motor or visual impairments, who may face additional challenges in using conventional interaction techniques and devices.

ACKNOWLEDGMENTS

This work was supported by a grant of the Ministry of Education and Research, CCCDI-UEFISCDI, project no. PN-IV-P7-7.1-PTE-2024-0434, within PNCDI IV.

References

- [1] H. Al-Najjar, N. Al-Rousan, H. F. Assous, and D. AL-Najjar. 2025. Machine learning models to predict desktop activity recognition based on low-point gaze features. *Array* 28 (2025). <https://doi.org/10.1016/j.array.2025.100525>
- [2] S. Amershi, D. Weld, M. Vorvoreanu, A. Fourney, B. Nushi, P. Collisson, J. Suh, S. Iqbal, P. N. Bennett, K. Inkpen, J. Teevan, R. Kikin-Gil, and E. Horvitz. 2019. Guidelines for Human-AI Interaction. In *Proc. of CHI Conf. on Human Factors in Computing Systems*. 13 pages. <https://doi.org/10.1145/3290605.3300233>
- [3] F. Aziz, C. Creed, S. Sarcar, M. Frutos-Pascual, and I. Williams. 2022. Voice Snapping: Inclusive Speech Interaction Techniques for Creative Object Manipulation. In *Proc. of the 2022 ACM Designing Interactive Systems Conference (DIS '22)*. ACM, USA, 1486–1496. <https://doi.org/10.1145/3532106.3533452>
- [4] A. Baevski, H. Zhou, A. Mohamed, and M. Auli. 2020. wav2vec 2.0: a framework for self-supervised learning of speech representations. In *Proc. of the 34th Int.*

- Conf. on Neural Information Processing Systems (NIPS '20)*. Curran Associates Inc., USA, Article 1044. <https://dl.acm.org/doi/abs/10.5555/3495724.3496768>
- [5] Aaron Bangor, Philip T. Kortum, and James T. Miller. 2008. An Empirical Evaluation of the System Usability Scale. *International Journal of Human-Computer Interaction* 24, 6 (2008), 574–594. <https://doi.org/10.1080/10447310802205776>
- [6] Rodolfo Cossovich, Steve Hodges, Jin Kang, and Audrey Girouard. 2023. Co-designing new keyboard and mouse solutions with people living with motor impairments. In *Proc. of the 25th Int. Conf. on Computers and Accessibility (ASSETS '23)*. ACM, Article 98. <https://doi.org/10.1145/3597638.3614549>
- [7] A. Dobriyal, Daksh Rawat, S. Vats, and V. Sharma. 2024. AI-Driven Personal Desktop Assistant. In *Proc. 2nd Int. Conf. on Innovations in High Speed Comm. and Signal Proc.* 1–5. <https://doi.org/10.1109/IHCSP63227.2024.10959800>
- [8] Alexandra-Elena Gurita and Radu-Daniel Vatavu. 2025. When LLM-Generated Code Perpetuates User Interface Accessibility Barriers, How Can We Break the Cycle?. In *Proc. of the 22nd Int. Web for All Conference (W4A '25)*. ACM, New York, NY, USA, 124–134. <https://doi.org/10.1145/3744257.3744266>
- [9] Matthew Hoy. 2018. Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants. *Medical Reference Services Quarterly* 37 (01 2018), 81–88. <https://doi.org/10.1080/02763869.2018.1404391>
- [10] S. Juneja, P. Bakshi, G. Kaur, R. Chandra, and R. K. Yadav. 2025. Leo: Personal Desktop Assistant Using Python. In *2025 3rd Int. Conf. on Disruptive Technologies (ICDT)*. 1508–1512. <https://doi.org/10.1109/ICDT63985.2025.10986691>
- [11] A. M. Klein, J. Kollmorgen, A. Hinderks, M. Schrepp, M. Rauschenberger, and M.-J. Escalona. 2025. Validation of the UEQ+ Scales for Voice Quality. *Computer Standards & Interfaces* 93 (2025). <https://doi.org/10.1016/j.csi.2024.103971>
- [12] Nithin Rao Koluguri, Taejin Park, and Boris Ginsburg. 2022. TitaNet: Neural Model for Speaker Representation with 1D Depth-Wise Separable Convolutions and Global Context. In *2022 IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*. <https://doi.org/10.1109/ICASSP43922.2022.9746806>
- [13] Sunil Kumar, Shubham Patel, Sonam, and Vaishnav Srivastav. 2024. Voice-Based Virtual Assistant for Windows (Ziva - AI Companion). In *2024 IEEE International Conference on Computing, Power and Communication Technologies (IC2PCT)*, Vol. 5. 960–965. <https://doi.org/10.1109/IC2PCT60090.2024.10486800>
- [14] James R. Lewis and Jeff Sauro. 2018. Item benchmarks for the system usability scale. *J. Usability Studies* 13, 3 (May 2018), 158–167. <https://dl.acm.org/doi/abs/10.5555/3294033.3294037>
- [15] Fei Liu, Yiming Yao, Ping Guo, Zhiyuan Yang, Xi Lin, Zhe Zhao, Xialiang Tong, Kun Mao, Zhichao Lu, Zhenkun Wang, Mingxuan Yuan, and Qingfu Zhang. 2026. A Systematic Survey on Large Language Models for Algorithm Design. *ACM Comput. Surv.* 58, 8 (2026), 32 pages. <https://doi.org/10.1145/3787585>
- [16] C. Murad, H. Candello, and C. Munteanu. 2023. What's The Talk on VUI Guidelines? A Meta-Analysis of Guidelines for Voice User Interface Design. In *Proc. of the 5th Int. Conf. on Conversational User Interfaces (CUI '23)*. ACM, USA, Article 19, 16 pages. <https://doi.org/10.1145/3571884.3597129>
- [17] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, P. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. 2022. Training language models to follow instructions with human feedback. In *Proc. of the 36th Int. Conf. on Neural Information Processing Systems (NIPS '22)*. Curran Associates Inc., USA, 15 pg. <https://dl.acm.org/doi/10.5555/3600270.3602281>
- [18] Kentrell Owens, Johanna Gunawan, David Choffnes, Pardis Emami-Naeini, Tadayoshi Kohno, and Franziska Roesner. 2022. Exploring Deceptive Design Patterns in Voice Interfaces. In *Proc. of the 2022 European Symposium on Usable Security (EuroUSEC '22)*. ACM, 64–78. <https://doi.org/10.1145/3549015.3554213>
- [19] Irina Petrariu, Laura-Bianca Bilius, Liviu Mitrofan, Sebastian Crintea, Robert Croitor, and Tudor Horomnea. 2026. A Comparative Study of Regression Techniques for Touch-based Input and Adaptive User Interfaces. In *Proceedings of the 18th International Conference on Development Application Systems (DAS '26)*. IEEE, 278–283. <https://doi.org/10.1109/DAS69882.2026.11553333>
- [20] M. Porcheron, J. E. Fischer, S. Reeves, and S. Sharples. 2018. Voice Interfaces in Everyday Life. In *Proc. of the 2018 CHI Conf. on Human Factors in Computing Systems (CHI '18)*. ACM, 1–12. <https://doi.org/10.1145/3173574.3174214>
- [21] Alisha Pradhan, Kanika Mehta, and Leah Findlater. 2018. "Accessibility Came by Accident": Use of Voice-Controlled Intelligent Personal Assistants by People with Disabilities. In *Proc. of the 2018 CHI Conf. on Human Factors in Computing Systems (CHI '18)*. ACM, USA, 1–13. <https://doi.org/10.1145/3173574.3174033>
- [22] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2023. Robust speech recognition via large-scale weak supervision. In *Proc. of the 40th Int. Conf. on Machine Learning (ICML'23)*. JMLR.org, Article 1182, 27 pages. <https://dl.acm.org/doi/10.5555/3618408.3619590>
- [23] Leon Reicherts, Yvonne Rogers, Licia Capra, Ethan Wood, Tu Dinh Duong, and Neil Sebire. 2022. It's Good to Talk: A Comparison of Using Voice Versus Screen-Based Interactions for Agent-Assisted Tasks. *ACM Trans. Comput.-Hum. Interact.* 29, 3, Article 25 (Jan. 2022), 41 pages. <https://doi.org/10.1145/3484221>
- [24] Lucas Rosenblatt, Patrick Carrington, Kotaro Hara, and Jeffrey P. Bigham. 2018. Vocal Programming for People with Upper-Body Motor Impairments. In *Proc. of the 15th Int. Web for All Conference (Lyon, France) (W4A '18)*. ACM, USA, Article 30, 10 pages. <https://doi.org/10.1145/3192714.3192821>
- [25] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Sjalom. 2023. Toolformer: language models can teach themselves to use tools. In *Proc. of the 37th Int. Conf. on Neural Information Processing Systems (NIPS '23)*. Curran Associates Inc., Article 2997. <https://dl.acm.org/doi/10.5555/3666122.3669119>
- [26] David Snyder, Daniel Garcia-Romero, Gregory Sell, Daniel Povey, and Sanjeev Khudanpur. 2018. X-Vectors: Robust DNN Embeddings for Speaker Recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 5329–5333. <https://doi.org/10.1109/ICASSP.2018.8461375>
- [27] Martin Subi, M. Rajeswari, J. Jerlin Rajan, and S Sri Harshini. 2024. AI-Based Desktop VIZ: A Voice-Activated Personal Assistant-Futuristic and Sustainable Technology. In *2024 10th Int. Conf. on Communication and Signal Processing (ICCCSP)*. 1095–1100. <https://doi.org/10.1109/ICCCSP60870.2024.10543665>
- [28] Mihail Terenti, Laura-Bianca Bilius, Ovidiu-Ciprian Ungurean, and Radu-Daniel Vatavu. 2025. Gesture-A11Y: A Large-Scale Hub for Accessible Gesture Input. In *Proc. of the 22nd Int. Web for All Conference (W4A '25)*. ACM, New York, NY, USA, 48–50. <https://doi.org/10.1145/3744257.3744280>
- [29] Vu N. Tran, Long V. Tran, and Viet N. Tran. 2021. Functional Hazard Analysis for Engineering Safe Software Requirements. In *Proc. Int. Conf. on Information and Computer Technologies*. <https://doi.org/10.1109/ICICT52872.2021.00031>
- [30] Radu-Daniel Vatavu, Irina Petrariu, Tudor Horomnea, and Ovidiu-Ciprian Ungurean. 2026. Modeling Touch Input for Users with Motor Impairments: Empirical Insights into Training Size Requirements. In *Proc. of the Ext. Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '26)*. ACM, New York, NY, USA, Article 473, 6 pages. <https://doi.org/10.1145/3772363.3798487>
- [31] Mikael Wiberg and Erik Bergqvist. 2023. Automation of interaction—Interaction design at the crossroads of user experience (UX) and artificial intelligence (AI). *Pers. Ubiq. Comput.* 27 (2023). <https://doi.org/10.1007/s00779-023-01779-0>