Screen Reading Enabled by Large Language Models

Anujay Ghosh*
Department of Computer Science,
Stony Brook University
United States
asghosh@cs.stonybrook.edu

Monalika Padma Reddy*
Department of Computer Science,
Stony Brook University
United States
mpadmareddy@cs.stonybrook.edu

Satwik Ram Kodandaram
Department of Computer Science,
Stony Brook University
United States
skodandaram@cs.stonybrook.edu

Utku Uckun†
Department of Computer Science,
Stony Brook University
United States
uuckun@cs.stonybrook.edu

Vikas Ashok
Department of Computer Science, Old
Dominion University
United States
vganjigu@odu.edu

Xiaojun Bi
Department of Computer Science,
Stony Brook University
United States
xiaojun@cs.stonybrook.edu

IV Ramakrishnan
Department of Computer Science,
Stony Brook University
United States
ram@cs.stonybrook.edu

Abstract

Large language models (LLMs), such as the pioneering GPT technology by OpenAI, have undeniably become one of the most significant innovations in recent history. They have achieved phenomenal success across a broad spectrum of applications in numerous industries, transforming how we interact with the digital world. Notwithstanding these remarkable successes, applying LLMs within the realm of accessibility has largely been unexplored. We introduce Savant, as a demonstration of the potential of LLMs for accessibility. Specifically, Savant leverages the impressive text comprehension abilities of LLMs to provide uniform interaction for screen reader users across various applications, mitigating the significant interaction burden imposed by the heterogeneity in user interfaces for blind screen reader users. Savant automates screen reader actions on control elements like buttons, text fields, and drop-down menus via spoken natural language commands (NLCs). Interpreting the NLC, identifying the correct control element, and formulating the action sequence are facilitated by LLMs. Few-shot prompts supply context and guidance for the LLMs to produce appropriate responses, specifically converting the NLC into a correct series of actions on the user interface elements, which are then performed automatically. The demonstration will exhibit Savant's capability across a variety of exemplar applications, emphasizing its versatility.

*Both authors contributed equally to this research.
†This research was done while the author was at Stony Brook University.

ASSETS '24, October 27–30, 2024, St. John's, NL, Canada

ACM ISBN 979-8-4007-0677-6/24/10
https://doi.org/10.1145/3663548.3688491

CCS Concepts

• **Human-centered computing** → **Accessibility technologies**; **Empirical studies in accessibility**.

Keywords

Blind users, Computer Interaction, Accessibility, Assistive technology, Uniform interaction, Large language models (LLMs)

ACM Reference Format:
Anujay Ghosh, Monalika Padma Reddy, Satwik Ram Kodandaram, Utku Uckun, Vikas Ashok, Xiaojun Bi, and IV Ramakrishnan. 2024. Screen Reading Enabled by Large Language Models. In *The 26th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '24), October 27–30, 2024, St. John's, NL, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3663548.3688491

1 Introduction

People who are blind predominantly rely on assistive technologies, such as screen readers(e.g., *JAWS* [1], *VoiceOver* [2], and *NVDA* [3]), to interact with computer applications. The screen reader vocalizes the text and describes the interface elements in a sequential, one-dimensional manner (e.g., pressing "P" will shift focus to the next paragraph), which allows blind users to navigate and interact with various controls and content sections of an application through keyboard shortcuts. However, the graphical user interfaces (GUIs) of these computer applications are primarily designed for sighted two-dimensional "point-and-click" interactions using a mouse. This design focus often makes it cumbersome for blind users to navigate and interact with these interfaces using keyboard shortcuts [5].

The heterogeneity of the applications' GUIs further increases the interaction burden on blind users. Each application features a *distinct* interface layout, necessitating that users memorize unique, application-specific shortcuts and navigation strategies to access

[1]https://www.freedomscientific.com/products/software/jaws/
[2]https://www.apple.com/accessibility/mac/vision/
[3]https://www.nvaccess.org/

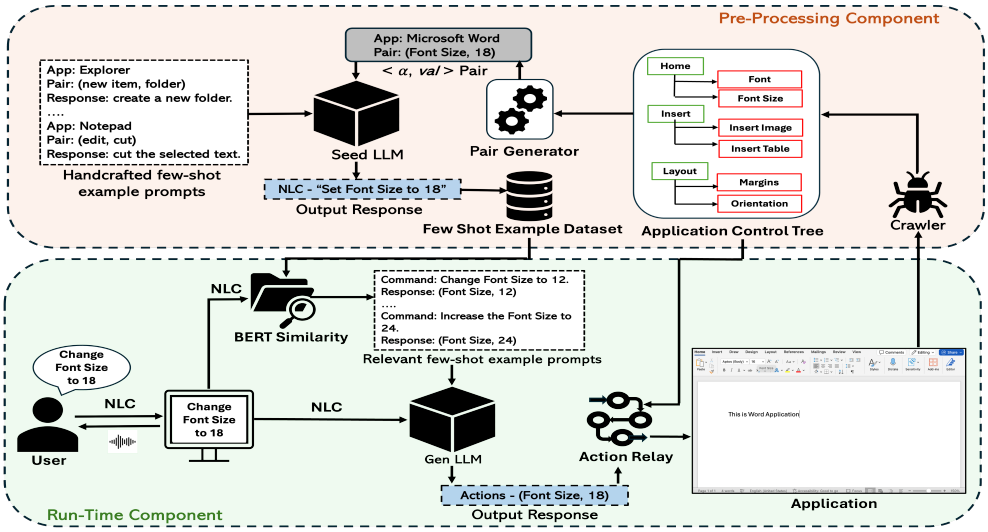

Figure 1: Architectural schematic of Savant. α stands for Control Element. *val* stands for the value of the Control Element. NLC stands for Natural Language Command.

content efficiently. This need for specialized knowledge is compounded when accessing the same application on different platforms, as each requires a different set of commands. For instance, to open the font dialog box in *Microsoft Word*, a user must press "Ctrl+Shift+F" on *Windows*, while on a *Mac*, it is "Command+D".

This *variability* across applications and platforms significantly deteriorates the user experience for blind users. In practical computer tasks, research indicates that blind users can be up to *ten times* slower than their sighted counterparts in the most challenging scenarios [3, 6, 11]. Furthermore, the *cognitive load* required to remember and execute the correct shortcuts often leads to overwhelming feelings and frequent mistakes, such as unintended key presses. For example, a blind user attempting to adjust text alignment in a document might mistakenly alter the text style if the keyboard shortcuts are confusingly similar or forgotten. This inconsistency can decrease efficiency and hinder digital accessibility.

Blind participants in a previous study highlighted significant limitations in the available assistive technologies, emphasizing the critical need for consistent and standardized access across various platforms, screen readers, and applications [2]. Despite their strong desire for *uniformity*, the methods to achieve such consistency still need to be developed. However, the recent emergence of Large Language Models (LLMs) (e.g., *GPT-4* [1], *Gemini* [9], and *Llama-2* [10]) presents a very promising avenue to tackle the challenging heterogeneity problem effectively.

As a demonstration of how LLMs can help improve accessibility, we present Savant. To address the problems caused by the varying user interfaces, Savant uses sophisticated text comprehension skills of LLMs to offer screen reader users a consistent interaction experience across several applications. By automating screen reader actions on control elements like buttons, text fields, and drop-down menus through spoken Natural Language Commands (NLCs), Savant significantly reduces the interaction burden.

Interpreting the NLC, identifying the correct control element, and formulating the action sequence are facilitated by LLMs. Few-shot prompts [7] supply context and guidance for the LLMs to produce appropriate responses, specifically converting the NLC into a correct series of actions on the user interface elements, which are then performed automatically. The demonstration will exhibit Savant's capability across a variety of exemplar applications, emphasizing its versatility.

2 Technical Design of Savant

Savant can be activated by the user using the keyboard shortcut "Alt+Insert". It consists of two major components: (1) Pre-Processing component and (2) Run-Time component. Figure 1 illustrates the architectural schematic of Savant.

The Pre-Processing component is primarily responsible for (i) constructing a non-visual hierarchical representation of the application GUI, termed as "Application Control Tree" (ACT), which encapsulates all the information required to generate a sequence of UI steps for triggering actions on control elements; and (ii) creating a dataset of few-shot examples for the application to serve as prompts (referred to as "Few-Shot Example Dataset" in the figure). The Crawler module, Pair Generator, and Seed LLM are the core elements of the Pre-Processing component.

The Crawler module constructs the ACT consisting of all the control elements of the application. It leverages the *Microsoft UI Automation Tool* [8] to extract both visible (control elements that are visible on screen) and invisible (control elements that are hidden behind other control elements, e.g., sub-menus) control elements in the application, along with metadata (e.g., control element name, type, actions on the control elements etc.). It extracts the invisible control elements by traversing visible elements in a breadth-first approach and simulating user interactions, such as clicking dropdowns or tabs recursively. It records the actions that reveal new controls and associates these actions with the corresponding control elements. This generates a hierarchical tree structure that accurately represents the UI hierarchy of an application.

The *Pair-Generator* module uses the metadata of the ACT to generate <control element (α), value (*val*)> pairs for each control element α in the application and saves it. We associate each action with a pair <α, *val*>, meaning that execution of the action will

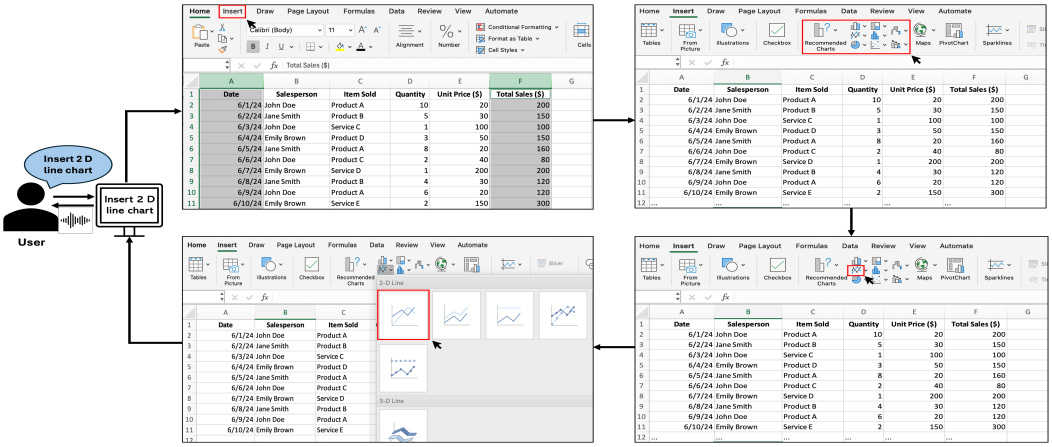

Figure 2: Demonstration on Excel to insert a Line Chart using the selected columns.

result in a sequence of screen reader steps that will result in the assignment of the *val* to α. If control elements (e.g., Bold button in Word) have no corresponding values, *val* is set to "None".

The Seed LLM generates a NLC for the given <α, *val*> pair, which is then stored as a <*NLC*, α, *val*> triplet in the Few-shot Example Dataset (FED). The <α, *val*> pairs from the Pair-Generator, along with a set of hand-crafted few-shot examples (each paired with a corresponding NLC as a response), serve as prompts to guide the Seed LLM in generating the NLCs accurately. The hand-crafted prompts are generic and are created only once. They are used to generate the FED dataset for any application.

The main objective of the Runtime component is to interpret the user's NLC, generate corresponding <α, *val*> pair, map an appropriate action to this pair, and execute the corresponding sequence of screen reader steps to perform the action. It includes the Gen LLM and the Action Relay modules.

The Gen LLM generates a <α, *val*> pair that accurately reflects the interpretation of the user's NLC. The prompt provided to the Gen LLM includes the user's NLC along with a set of few-shot examples, selected for their relevance through BERT similarity [4] from the FED. The Action Relay module generates a sequence of screen reader steps to execute the action on the <α, *val*> pair. These are guided by information retrieved from the ACT.

The Pre-processing for each application is done only once. During runtime, the Application Control Tree (ACT) and Few-Shot Examples Dataset (FED) are used repeatedly to generate the sequence of screen reader steps needed to execute the user's NLC as shown in Figure 1. We employed GPT-4 [1] with few-shot learning, as both the Seed LLM and Gen LLM.

3 Demonstration Scenarios

Savant enhances accessibility and streamlines navigation across various applications through NLCs, as demonstrated in the scenarios below. It eliminates the need to memorize application-specific shortcuts and control names, thus enabling consistent and efficient user interactions across diverse applications. Savant's ability to understand multiple expressions of the same command allows for more natural and flexible application interactions. Additionally, it significantly reduces cognitive load by simplifying task execution, enabling users to complete tasks without sequentially navigating every on-screen element.

3.1 Scenario 1

Bob, a blind marketing professional, is preparing a monthly sales report for his manager on MS Excel. He wants to insert a line graph illustrating the sales in the previous month.

If Bob solely relied on screen readers to perform this task, he would first select the columns "Date" and "Total Sales". Then, he would activate the Menu ribbon and select the *Insert* Tab using the keyboard shortcut *Alt+N*. Subsequently, he would use the arrow keys to navigate to the *Line or Area chart* button, choose the *2 D Line chart option* from the dropdown menu, and hit *Enter*.

With Savant, Bob simply needs to select the columns labeled "Date" and "Total Sales" and say "Insert a 2 D Line chart", as illustrated in Figure 2. In response, Savant interprets the NLC, generates the control value pair <*insert line or area chart*, *line*>, identifies and sequentially executes the following sequence of screen reader steps: *Select Menu ribbon, Click Insert tab, Click Line or Area chart button, Select 2 D Line chart.* Upon executing the command, Savant provides auditory confirmation indicating the task completion.

Whether Bob says "Add a 2D line chart" or "Create a 2D line chart", Savant accurately interprets and executes the action. Bob can then continue to refine his work. Whether he wants to adjust the font size by first selecting the text and saying "Set the font size to 18" or modify the font color by saying "Change the font color to Red", Savant will accurately interpret his command and execute the corresponding steps. It substantially reduces Bob's cognitive load by streamlining task execution, enabling him to perform such simple tasks, without the need to sequentially navigate every on-screen element. This makes iterative editing significantly easier.

3.2 Scenario 2

Claire, who is blind, is sitting in her home office listening to music on Spotify, practicing for an upcoming audition. As the first song nears its end, she decides to replay it and focus on the high notes.

If Claire were to use a screen reader to replay the song, she would have to navigate to the "Enable repeat button" using the *Tab* key and hit *Enter*. Instead, she simply commands Savant to "Repeat this song"

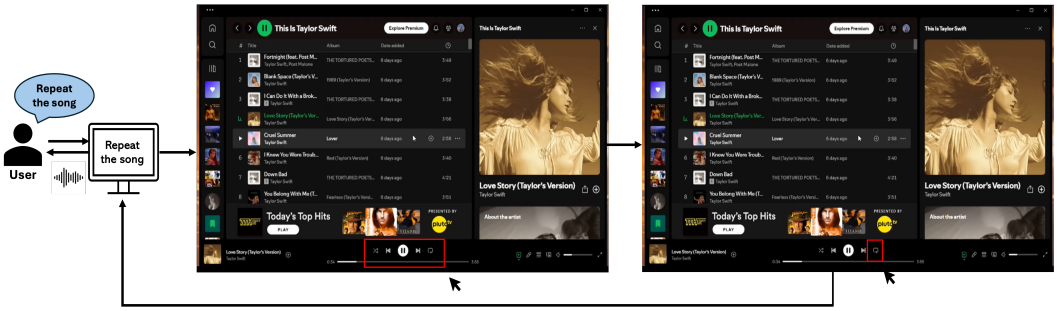

Figure 3: Demonstration on Spotify to repeat the current song.

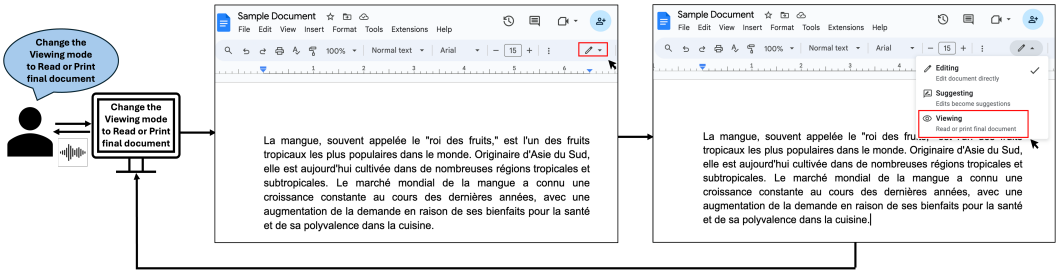

Figure 4: Demonstration on Google Docs to change the view mode to Read or Print final document.

as shown in Figure 3. Savant identifies the intent of her command, which is to "Repeat the song that is currently playing", and converts it into an action on the control element using the control-value pair <*enable repeat*,*none*>. Then, it simulates a click on the "Enable repeat button". Upon executing the command, Savant issues voice feedback to indicate the task has been successfully completed.

3.3 Scenario 3

Liam, a blind writer, has finalized his article on Google Docs and wants to proofread it in "Read or Print final document" view mode.

If Liam decides to rely on his screen reader to change the viewing mode to "read or print final document", he would first navigate to the View menu using the keyboard shortcut *Alt + Shift + V*. Next, he would use the *Down* arrow key to navigate to the "Mode" sub-menu, the *Right arrow* key to expand the submenu and continue using the *Down arrow* key to reach the "Viewing - Read or Print final document" mode. Finally, he would press *Enter*.

Instead, Liam completes the same task by simply saying, "Change the viewing mode to read or print final document" as seen in Figure 4. In response, Savant interprets Liam's NLC and generates <*Editing mode*, *Viewing read or print final document*> as control value pair. It identifies and automates the series of screen reader steps: *Click Editing Drop down, Click Viewing Read or print final document*. Savant confirms the completion of the task with voice feedback after executing the command.

4 Evaluation

We conducted an IRB-approved study with 11 blind users using a within-subject design to assess our system's effectiveness across six applications: *Amazon* (opened in *Firefox*), *Excel*, *File Explorer*, *Google Docs* (opened in *Chrome*), *Outlook*, and *Spotify*. Participants completed the assigned tasks under three conditions: using *JAWS* alone, with the state-of-the-art *AccessBolt* [11], and with our *Savant* system to evaluate each setup's generalizability across different use cases like e-commerce, productivity, and text editing.

We assessed performance across all conditions using key metrics such as task completion rates, key press statistics, Single Ease Question (SEQ) scores, and NASA-TLX questionnaire responses.

The task completion rates were 27.3% (6 out of 22 tasks) for *JAWS* alone, 63.6% (14 out of 22 tasks) for *JAWS with AccessBolt* and 81.8% (18 out of 22 tasks) for *JAWS with Savant*. The key press analysis showed that basic navigation keys made up 91% of key presses in the *JAWS* condition, 70% with *AccessBolt*, and 30% with *Savant*.

The Single Ease Question (SEQ) scores indicated better usability for Savant (6.57/7, median=7, SD=0.80) and AccessBolt (5.6/7, median=6, SD=1.61) compared to JAWS (2/7, median=2, SD=1.19). NASA-TLX scores reflected lower task load for Savant (mean=16.16, median=15, SD=8.57) and AccessBolt (mean=28.93, median=29.16, SD=5.09) versus JAWS (mean=49.9, median=45.8, SD=20.9). Participants particularly valued the system's user-friendliness, fewer keystrokes, and effective natural language command interface.

5 Conclusion

In conclusion, Savant represents a significant advancement in accessibility technologies for blind users, leveraging the capabilities of large language models to enhance interaction with digital interfaces. By automating complex screen reader actions through natural language, Savant reduces the cognitive load and interaction burden, significantly improving usability and efficiency. This demonstration underscores the transformative potential of LLMs

in making digital environments more accessible and user-friendly for visually impaired individuals.

Acknowledgments

We thank the anonymous reviewers for their insightful feedback. This work was supported by the Google Inclusion Research Award, NSF Awards: 2153056, 2125147, 2113485, and the following NIH Awards: R01EY030085, R01HD097188, and R01EY035688.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Syed Masum Billah, Vikas Ashok, Donald E Porter, and IV Ramakrishnan. 2017. Ubiquitous accessibility for people with visual impairments: Are we there yet?. In *Proceedings of the 2017 chi conference on human factors in computing systems*. 5862–5868.

[3] Maria Claudia Buzzi, Marina Buzzi, Barbara Leporini, and Giulio Mori. 2012. Designing e-learning collaborative tools for blind people. *E-Learning-Long-Distance and Lifelong Perspectives* (2012), 125–144.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[5] Hae-Na Lee, Vikas Ashok, and IV Ramakrishnan. 2020. Repurposing visual input modalities for blind users: a case study of word processors. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2714–2721.

[6] H. N. Lee, V. Ashok, and I. V. Ramakrishnan. 2020. Repurposing Visual Input Modalities for Blind Users: A Case Study of Word Processors. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2714–2721. https://doi.org/10.1109/SMC42975.2020.9283015

[7] Ben Mann, N Ryder, M Subbiah, J Kaplan, P Dhariwal, A Neelakantan, P Shyam, G Sastry, A Askell, S Agarwal, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* 1 (2020).

[8] Microsoft. 2021. *Microsoft UI Automation.* https://docs.microsoft.com/en-us/windows/win32/winauto/entry-uiauto-win32 Accessed: 2024-07-02.

[9] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).

[10] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[11] Utku Uckun, Rohan Tumkur Suresh, Md Javedul Ferdous, Xiaojun Bi, IV Ramakrishnan, and Vikas Ashok. 2022. Taming User-Interface Heterogeneity with Uniform Overlays for Blind Users. In *Proceedings of the 30th ACM Conference on User Modeling, Adaptation and Personalization*. 212–222.