Speech-Driven Task Automation in Windows Applications Using Large Language Models for Accessibility

A Thesis presented

by

Anujay Ghosh

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Master of Science

in

Computer Science

Stony Brook University

December 2024

Stony Brook University

The Graduate School

Anujay Ghosh

We, the thesis committee for the above candidate for the

Master of Science degree, hereby recommend

acceptance of this thesis.

Dr. IV Ramakrishnan – Thesis Advisor
Professor and Associate Dean for Research, Department of Computer Science

Dr. Paul Fodor – Chairperson of Defense
Associate Professor of Practice, Department of Computer Science

Dr. Xiaojun Bi
Associate Professor, Department of Computer Science

This thesis is accepted by the Graduate School

Celia Marshik

Dean of the Graduate School

Abstract of the Thesis

Speech-Driven Task Automation in Windows Applications Using Large Language Models for Accessibility

by

Anujay Ghosh

Master of Science

in

Computer Science

Stony Brook University

2024

Accessibility and task automation are crucial for blind users to effectively interact with various software applications. Studies show that blind users often face significant challenges when navigating complex user interfaces that deal with point-and-click operations using the mouse pointer that are time consuming when trying to perform using just the keyboard and screen reader software, often resulting in frustration and reduced productivity. Automating tasks using voice-driven interfaces can help blind users have a better experience when navigating complex series of actions required to perform specific tasks in numerous software applications.

We present Savant, a cutting-edge assistive technology that automates screen-reader steps to perform a sequence of actions to perform tasks in various software on personal computers. This interface software tackles some of the major challenges caused by complicated and varied desktop application graphical user interfaces (GUIs), which frequently cause blind users to have an unpleasant and time-consuming experience.

Savant is an application-agnostic solution that uses a push-based methodology to simplify and standardize non-visual computer interaction. Utilizing the advanced technologies of Large Language Models (LLMs), Savant successfully comprehends the intent of the user, taking in speech input from them given in natural language, and converting them to a sequence of actions to perform to complete the task. It then executes the actions in the correct order on the software, providing a voice confirmation when the task is complete. Users no longer need to familiarize themselves with distinct interface designs of each program, trying to remember which control is under which menu and which tab, thanks to the streamlined process of Savant that eliminates the need for memorization of such controls.

"People with impairments are disabled by their environments; or, to put it differently, impairments aren't disabling, social and architectural barriers are."

— Alison Kafer

Contents

List of Figures

List of Tables

List of Abbreviations

SBU	**S**tony **B**rook **U**niversity
Ph.D.	**D**octor of **Ph**ilosophy
M.S.	**M**aster of **S**cience
ACT	**A**pplication **C**ontrol **T**ree
AI	**A**rtificial **I**ntelligence
FED	**F**ew-Shot **E**xamples **D**ataset
GPT	**G**enerative **P**re-trained **T**ransformers by OpenAI
GUI	**G**raphical **U**ser **I**nterface
JAWS	**J**ob **A**ccess **W**ith **S**peech
LLM	**L**arge **L**anguage **M**odel
NLC	**N**atural **L**anguage **C**ommand
NVDA	**N**on**V**isual **D**esktop **A**ccess
OCR	**O**ptical **C**haracter **R**ecognition
WOZ	**W**izard of **Oz** Study

Acknowledgements

First and foremost, I want to sincerely thank my advisor, Dr. IV Ramakrishnan, for his invaluable mentorship and unwavering support throughout my thesis journey. His extensive knowledge and insight have played a crucial role in guiding my research and helping me overcome the challenges of graduate school. I am truly thankful for his mentorship and encouragement.

As members of my thesis committee, Dr. Paul Fodor and Dr. Xiaojun Bi have made significant contributions, for which I'm truly grateful. Their insightful feedback, expert guidance, and stimulating discussions have greatly improved the quality of my work and broadened my research perspective.

My Ph.D. mentor, Satwik Ram Kodandaram, has my sincere gratitude for his dedicated commitment and insightful guidance has been essential to my growth as a researcher. Our regular brainstorming sessions, rich with new ideas and innovative techniques, have not only enhanced my work but also fueled my passion for research.

My deepest appreciation goes to my family and friends, whose unwavering support has been my anchor during this challenging yet rewarding academic journey. Their unconditional love, empathy, and constant encouragement have strengthened my resolve, providing the emotional foundation necessary to navigate the complexities of this scholarly pursuit.

Finally, I want to thank and recognize the support and resources from Stony Brook University that have made enabled me to do this research.

I want to express my profound gratitude to everyone above as well as to everyone who has contributed to my academic journey in various ways. Thank you for being a vital part of this important milestone in my life.

To my parents and sister, whose unwavering support, love, and encouragement have been the bedrock of my journey.

Chapter 1

Introduction

Navigating the digital landscape presents unique challenges for blind individuals. Screen readers, such as JAWS [16], VoiceOver [4] and NVDA [36], serve as crucial bridges, translating visual graphical user interfaces (GUIs) into auditory experiences. These tools enable blind users to interact with applications through keyboard commands, vocalizing on-screen content and interface elements.

Yet, the evolution of computer applications has inadvertently widened the accessibility gap. Modern software often consists of intricate GUIs with densely packed visual elements, optimized for mouse-driven point-and-click or drag-and-drop interactions. This design, while visually appealing and intuitive for users with good vision, often creates barriers for those who rely on keyboard navigation and screen readers.

The resulting disconnect between interface design and accessibility needs manifests itself in everyday challenges. Blind users frequently struggle with tasks that sighted users might find trivial, such as locating and activating ribbon commands or navigating complex menu structures. This disparity stresses a pressing need for more inclusive design practices in software development, the ones that consider the diverse interaction methods required by all users, regardless of visual ability. As technology continues to advance, addressing these accessibility challenges becomes increasingly critical. Ensuring that digital tools are truly accessible to all users not only enhances usability for blind users but also promotes a more inclusive digital ecosystem.

1.1 Screen Readers vs. Savant

Savant, an assistive interface running on top of other software applications, tackles these challenges by leveraging LLMs to enable blind users to streamline interaction with different applications through voice commands. Receiving a speech command provided by the user in natural language, Savant uses LLMs to understand the intent of the command and execute a sequence of one or more steps to complete the task.
To understand the workings of Savant, let's consider the following demonstration scenario: Alice, a legally blind individual is typing up a Word document at work and needs to insert a table with 4 rows and 5 columns to enter quarterly sales data. Using just a screen reader, the task would be executed using the following steps:

1. Navigate to the desired location in the document where Alice wants to insert the table using the cursor.
2. Press Alt+N, T, I to open the Insert Table dialog box.
3. Press Alt+C to move to the Number of columns field, then type "5"
4. Press Alt+R to move to the Number of rows field, then type "4".
5. Press the Tab key until she hears "OK" button.
6. Press Enter.

Alternatively, using Savant, the process would simplify as illustrated in Fig. 1.1:

1. Navigate to the desired location in the document where Alice wants to insert the table using the cursor.
2. Press the push button hotkey Alt+Insert and say "Insert a table with 4 rows and 5 columns."
3. Press Enter on the Savant overlay.

Savant's capabilities extend far beyond simple command execution. Without this tool, Alice would need to manually perform each step of the process, which can be time-consuming, not to mention remembering a long list of shortcut combinations increasing her cognitive load. What truly sets Savant apart is its flexibility in interpreting

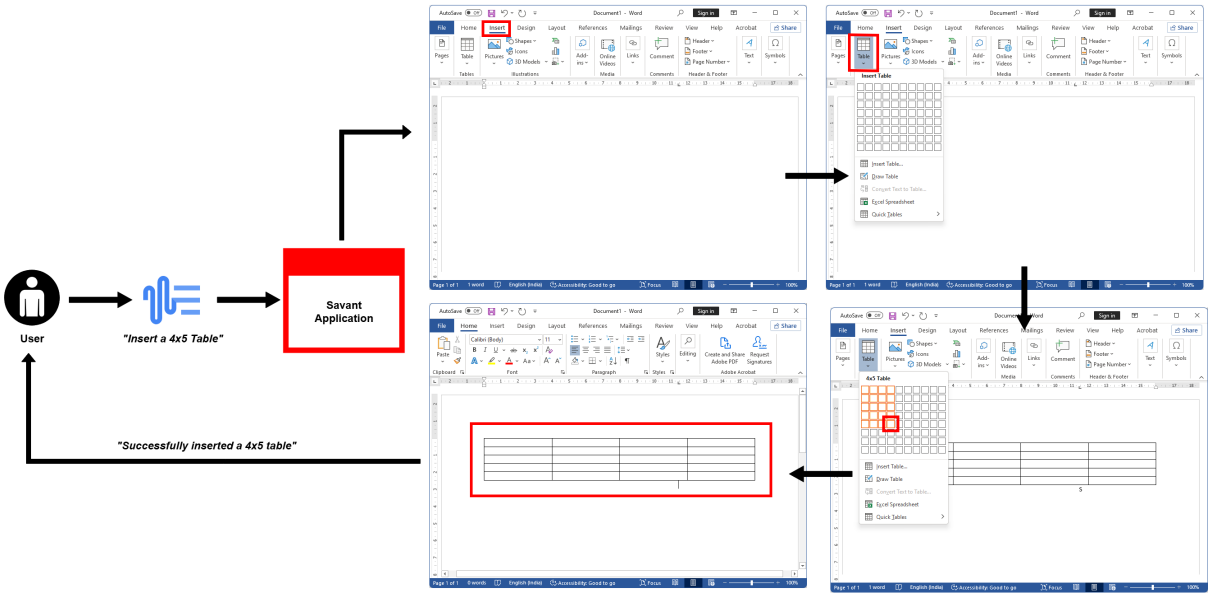

FIGURE 1.1: Inserting a table on Word using Savant

user intent. Whether Alice phrases her request as "Add a table with 5 columns and 4 rows", "Create a 4x5 table" or "I want to insert a 4x5 table", Savant's underlying LLMs accurately decipher the intent. This robust natural language understanding allows Savant to consistently translate diverse phrasings into the same series of screen reader actions on the active application.

This adaptability showcases the true power of Savant. It doesn't just automate tasks; it bridges the gap between natural human communication and complex software interfaces. By leveraging advanced AI capabilities, Savant provides a more intuitive and accessible computing experience for visually impaired users, reducing the cognitive load associated with remembering specific commands or navigating complex menu structures and presents a standardized natural language interface to interact with controls across a variety of desktop applications by harnessing the power of LLMs.

Chapter 2

Literature Review

2.1 Screen Readers

2.1.1 Usage and Functionality

Screen readers are essential assistive technologies for the blind or low-vision demographic of users, facilitating them to access digital content and work on various software through auditory feedback. These tools convert on-screen text into speech, allowing users to navigate websites, documents, and software applications using keyboard combinations. Popular screen readers include JAWS [16], NVDA [36], VoiceOver [4], and ZoomText [18], each offering varying functionality and accessibility features.

Screen readers have been instrumental in promoting independence, inclusive education, and employment opportunities for low-vision individuals since they are more accessible and cheaper than alternative solutions such as braille displays [21, 54, 53]. They facilitate online shopping, banking, and accessing educational materials by converting two-dimensional GUIs into one-dimensional solutions. However, learning to use screen readers effectively requires time and practice due to the cognitive load of memorizing keystroke commands and interpreting audio feedback provided back.

2.1.2 Drawbacks and Challenges

Despite screen readers being very useful, they present significant challenges for users [12, 44, 48, 51]. Studies highlight difficulties such as cognitive overload from long-drawn sequential navigation, issues accessing multimedia content, and the inability to interact with non-standard or inaccessible controls [25, 39]. For example, higher education students using screen readers reported difficulties in accessing printed classroom materials and digital content, as well as time lost due to technical issues [25]. Another study revealed that blind users often face high cognitive loads when navigating complex interfaces like digital artboards or electronic health record (EHR) systems [34]. These challenges arise from inaccessible design elements and the need for extensive navigation to locate desired information [39]. Additionally, inconsistencies in screen reader support across applications further increase usability issues and create unnecessary hurdles for low-vision users [28].

2.2 Alternatives to Screen Readers

While screen readers remain the most popular "go-to" assistive technology for blind users, alternative methods have been explored. Voice assistants like Amazon Alexa [2], Apple's Siri [3] and Google Assistant [20] offer a certain degree of automation and interactions through natural language commands. These tools have been shown to enhance accessibility by providing intuitive ways to perform tasks such as searching for information or controlling smart devices such as smartphones, smart speakers and smart TVs among others [46]. While users can perform basic tasks such as launching applications, these assistants fall short in executing more specific commands within those applications. For example, on MacOS, Siri can open Microsoft Word but lacks the capability to adjust document settings like margins or enable tracking changes. This limitation extends even to native MacOS applications; in Finder, Siri is unable to

perform simple file management tasks such as renaming files. These constraints highlight the gap between the current capabilities of voice assistants and the more intricate needs of users, particularly those who rely on non-visual interaction methods. Other alternatives include image captioning and OCR technologies that use computer vision to describe visual content to blind users. For instance, JAWS' Picture Smart AI and OCR features have been utilized to develop image captioning models that help low-vision individuals understand their surroundings better with their 2024 updates [17]. However, while these alternatives provide additional functionality, they often lack the comprehensive navigation capabilities of traditional screen readers. Another limitation of these solutions is that they are typically confined to operate within specific applications or ecosystems, restricting their versatility. Furthermore, the proprietary nature of their underlying technologies makes it challenging, if not unfeasible, to adapt or extend these assistants for universal use across diverse applications.

2.3 Computer Usability for Screen Reader Users

While extensive research has focused on enhancing the accessibility of computer applications [13, 24, 27, 41, 42, 56], along with studies addressing general web safety and improvements [10, 29], complemented by comprehensive guidelines and recommendations [22, 35], the domain of usability—encompassing the ease, efficiency, and satisfaction with which blind users perform computer-based tasks—remains relatively underexplored. Only a limited number of studies have delved into the usability of specific computer applications [6, 30, 51]. The inherent divergence between modern two-dimensional graphical user interfaces, designed primarily for visual interaction, and the linear navigation paradigm employed by screen readers exacerbates usability challenges, as evidenced by prior investigations [7, 32, 45, 50, 51]. For example, Shinohara et al. [40], in an early and seminal work, illuminated the profound navigational challenges encountered by blind users within application interfaces. Their

findings underscored the destabilizing impact of even a single erroneous key press, which could significantly disorient users and necessitate considerable cognitive effort and time to regain contextual bearings within the application.

In another investigation, Billah et al. [11] identified that inconsistencies in shortcut mappings across different screen readers and applications introduced significant confusion and interaction difficulties for blind users relying on screen readers. A more recent study by Islam et al. [24] highlighted the multifaceted nature of how screen reader users perceive the usability and accessibility of desktop applications. Their findings revealed that these perceptions extend beyond conventional definitions, encompassing elements such as the user's sense of independence, the ease with which an application can be learned, the ability to articulate its use to others, the reliability and predictability of keyboard shortcuts, and the transferability of knowledge to support others in using the application.

Research on enhancing usability for visually impaired users includes various assistive technologies, primarily focused on web [5, 14, 15, 43] and mobile [9, 26, 31] applications. Notable studies have introduced natural language assistants and web automation techniques to facilitate web interaction for blind users, while solutions for desktop applications are less frequently addressed.

On the other hand, Uckun et al. [45] introduced *AccessBolt*, a system-wide solution for Windows OS that utilized Microsoft's UI Automation tree [33] to present a centralized pop-up interface for sequentially accessing application controls, eliminating the need to navigate across the GUI. While AccessBolt demonstrated notable usability improvements for blind users, it had significant limitations: it only supported visible controls (excluding those in dropdowns or inactive ribbons) and required extensive time and shortcut inputs for sequential navigation, particularly in complex applications with numerous controls. In contrast, Savant streamlines interaction by bypassing low-level screen-reader operations and GUI navigation, offering a more seamless experience through flexible natural language commands.

2.4 Natural Language Assistance for Low-vision Users

Recent advancements in assistive technologies have significantly improved accessibility for blind persons [13, 24, 41]. LLMs and AI devices are at the forefront of this innovation, offering new possibilities for interaction with software applications. Modern applications predominantly feature two-dimensional GUIs optimized for sighted users, which inherently become a barrier with the linear navigation methods of screen readers [23, 38]. This gap between the two methods has resulted in many usability issues, as evidenced by several recent studies.

For instance, Billah et al. [11] highlighted how inconsistencies in shortcut mappings across different screen readers and applications lead to confusion and interaction difficulties for blind users. Building on this, Reuschel et al.'s research [38] on online job applications revealed that only 55.6% of application attempts by screen reader users were successful, underscoring the persistent barriers in web accessibility.

As technology advances with access to complex AI systems to an average user, new challenges also emerge with them. Adnin et al.'s study [1] on blind users' interactions with generative AI tools like ChatGPT [37] and Be My AI [8] exposed how users navigate accessibility issues and develop sometimes flawed mental models of these complex systems. This research emphasizes the need for more intuitive and accessible interfaces for emerging technologies.

Furthermore, a comprehensive user experience study advocated for tailored design approaches for the blind population and explored the potential of AI integration to enhance screen reader functionality. These findings collectively point to a critical gap in assistive technology: the need for a more uniform, intuitive, and efficient interface that can adapt to various applications and emerging technologies.

It is within this context that Savant emerges as a pioneering solution. By integrating the power of LLMs, Savant addresses the limitations identified in these studies. It offers a natural language-based interface that provides consistent interaction across

diverse applications, effectively bridging the gap between complex GUIs and the linear navigation of traditional screen readers [28]. Savant's approach not only simplifies navigation by automating actions but also adapts to new technologies by crawling application controls and updating its database, potentially resolving the challenges highlighted in Adnin et al.'s study [1] of AI tool interactions.

Savant represents a significant step forward in assistive technology, directly addressing the usability issues and inconsistencies identified in recent research. By providing a uniform interface that can interpret natural language commands across a variety of applications due to its application-agnostic processing, Savant helps improve the digital accessibility landscape for blind users [19].

Chapter 3

Architecture

Building on the AccessBolt framework [45], Savant's architecture consists of two main components: a Pre-Processing component and a Runtime component. The Pre-Processing component is leveraged for creating a one-dimensional non-visual representation of the application's GUI called the Application Control Tree (ACT) and developing a dataset of few-shot examples for the application, referred to as the Few-Shot Examples Dataset (FED). The Runtime component interprets the user's natural language commands (NLCs), generates the appropriate control element and value pairs based on this interpretation, associates actions with these pairs, and executes the necessary screen reader steps to perform the actions.

3.1 Pre-Processing Component

The Pre-Processing component serves two primary functions:

1. Creating a non-visual representation of the application's graphical user interface (GUI). This representation, termed the Application Control Tree (ACT), contains information used to generate the sequence of screen reader steps necessary for invoking actions on control elements.
2. Developing a dataset of few-shot samples for the application, referred to as the Few-Shot Examples Dataset (FED), which augments the prompts for the system.

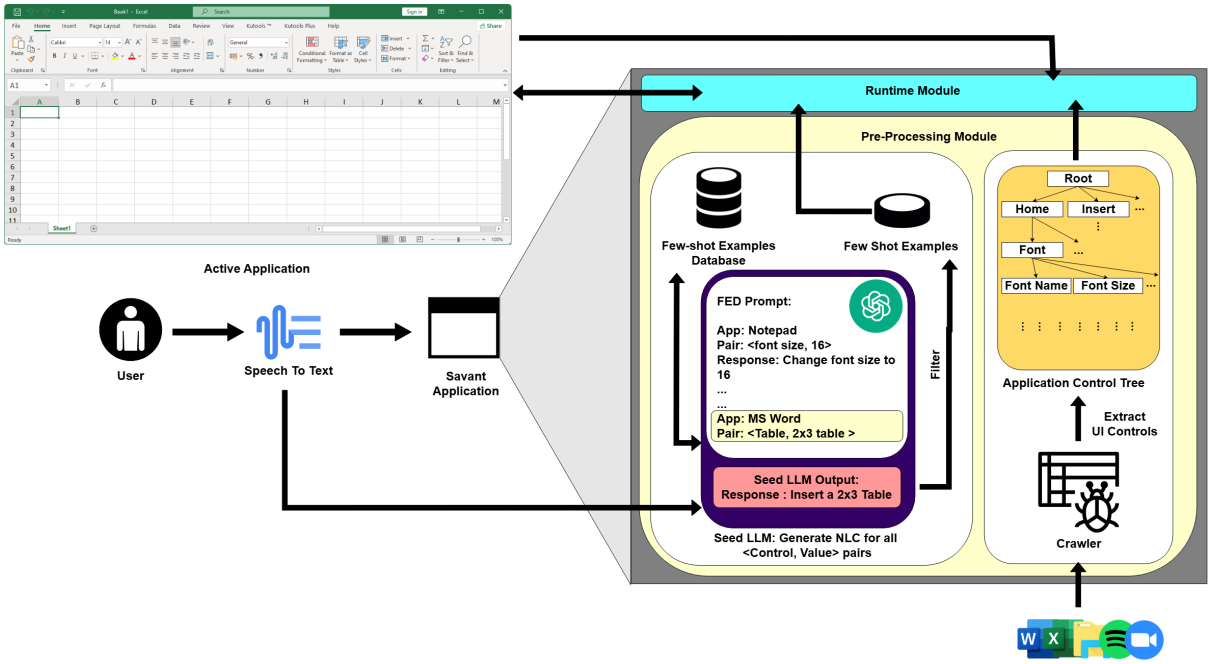

FIGURE 3.1: Pre-Processing Component Architecture of Savant

Fig. 3.1 illustrates the architecture of the Pre-Processing component in Savant. It's important to note that pre-processing occurs only during the initial setup or when there is an update to the application's control structure. The resulting ACT and FED are then utilized repeatedly at runtime to generate the sequence of screen reader steps required to execute the user's natural language command (NLC).

3.1.1 Crawler

The Crawler module constructs the ACT by leveraging the UI Automation tool API(UIA) by Microsoft [33] to extract all control elements of the application, including relevant metadata such as element names, types, hierarchy, and possible actions.

It employs a sophisticated approach to capture both visible and hidden controls within application interfaces. It begins by identifying visible controls directly present in the application window, such as control names and types. However, the challenge lies in accessing hidden controls, like sub-menus and dropdowns, which are not immediately crawled in the UI Automation tree. To address this, the Crawler implements a

breadth-first exploration strategy, systematically simulating user interactions on visible elements. This process includes actions such as expanding drop-down menus and selecting tabs, which can reveal previously hidden controls. For instance, in Word, clicking on the "font" drop-down exposes all available font options, while selecting the "Insert" tab displays its associated controls related to inserting tables, graphs etc.

3.1.2 Pair Generator

The Pair-Generator module creates and stores <*Control Element, Value*> pairs for every control element in the application, using the ACT's metadata as a reference. For example, in Microsoft Word, a pair might be <*Table,* $4x5$ *table*>, with additional pairs created for some of the other possible values like "$3x5$ *Table*", "$7x7$ *Table*", or "$20x2$ *Table*" based on user's command. This control would insert the table of the dimensions spoken by the user into the Word page wherever the cursor is located.
In the case of certain control elements, there might not be associated values to select from. For example, the Italics feature in Word operates as a toggle rather than offering multiple options. In such instances, the value for the Italics control element is designated as 'none' to indicate its binary nature. This approach allows the system to consistently represent all control elements, regardless of whether they have selectable values or function as simple on/off toggles.

3.1.3 Seed LLM

The Seed LLM module generates the few-shot examples dataset for that particular application, storing it in the FED. Each example in the FED is a triple <*NLC, Control Element, Value*>, providing an interpretation for the NLC by linking the action of assigning a value to a specific control element. The process of generating the Few-Shot Example Dataset (FED) involves using a Seed Large Language Model (LLM) with carefully designed prompts. These prompts include a selection of hand-crafted examples,

some derived from a Wizard of Oz (WOZ) study [28], along with corresponding Natural Language Commands (NLCs) as responses. Table A.1 in the Appendix contains an example of such a prompt used and it's corresponding response generated.
The Seed LLM is tasked with creating NLCs for each Control Element and Value pair provided by the Pair-Generator module. The output of this process is stored in the FED, forming a comprehensive dataset of example commands for the application.
To ensure broad applicability, the few-shot examples used in the prompts cover a diverse range of applications, including file management, text editing, video conferencing, music streaming, and email. This variety helps the model generalize across different types of computer applications and functionalities. The examples also encompass a wide spectrum of command types, such as:

- Creating new items (e.g. folders)
- Text formatting (e.g. applying strike-through)
- Meeting management (e.g. starting a video conference meeting)
- Adjusting formatting settings (e.g. changing font size)
- Media playback controls(e.g. play next song, repeat current song)
- Interface display modifications (e.g. maximize window)

This breadth of command types ensures that the Seed LLM can generate accurate and varied NLCs for different scenarios. Importantly, these hand-crafted prompts are designed to be generic, with various types of applications crawled(see Table A.3 in the Appendix) allowing the same set of few-shot examples to be used in generating FED datasets for any application. This approach enhances the system's flexibility and adaptability across different software environments.

3.2 Runtime Component

The Runtime component of Savant is designed to process and execute user commands given in natural language as illustrated in Fig. 3.2. This component interprets the

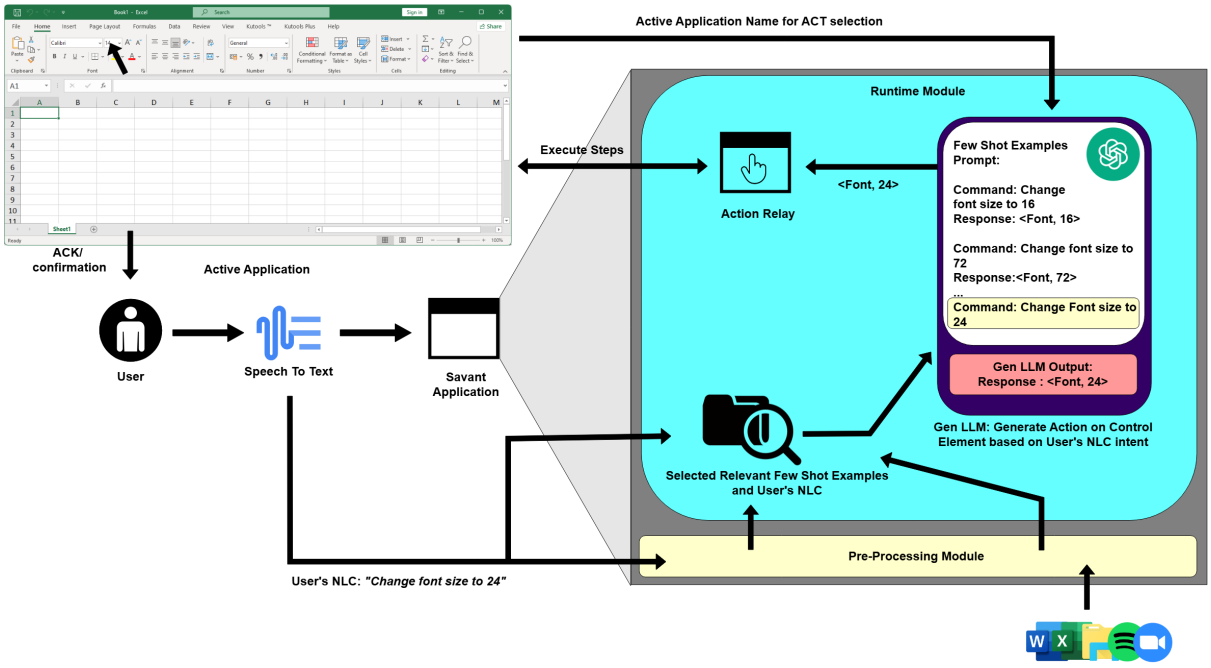

FIGURE 3.2: Runtime Component Architecture of Savant

user's input, generates an appropriate <*Control Element, Value*> pair, associates an action with this pair, and executes the necessary steps to perform the action.

3.2.1 Generative LLM

At the heart of this process is the Generative LLM module(Gen LLM), which utilizes GPT-4 or Gemini's capabilities (based on user preference) with few-shot learning to interpret the user's NLC. The module's prompt includes the user's input along with relevant few-shot examples. These examples are selected based on their semantic similarity to the user's command, calculated using BERT word embeddings. This approach ensures that the generated <*Control element, Value*> accurately reflects the user's intent as illustrated by an example in Table A.2 of the Appendix.

Once the Gen LLM module produces the control element and value pair, the Action Relay module takes over. This module generates and executes a sequence of screen reader steps corresponding to the interpreted action. It refers to the Application Control Tree (ACT) of the respective application to determine the precise steps needed to

carry out the action within the application's interface.

The system is designed to handle potential ambiguities in user commands. In cases where a command could have multiple interpretations, the Gen LLM module can present multiple options as an overlay which can be read out out with the screen reader to the user for clarification. This feature helps to ensure that the system accurately interprets the natural language command and executes the user's intended actions in the proper sequence.

For evaluating the performance of the Gen LLM module, the dataset from a Wizard of Oz study was employed consisting of 11 blind persons. [28] The module achieved an accuracy of 78% in correctly interpreting user commands, whereas further analysis of incorrect outputs revealed challenges with multi-step commands, where users combined multiple actions into a single instruction. This limitation highlights areas for future improvement in the system's natural language processing capabilities.

3.2.2 Action Relay

The Action Relay module plays a crucial role in translating the interpreted command into actual screen reader actions. For example, when processing a command to change margins to wide, the module executes a series of steps such as selecting the appropriate tab(Layout Tab), menu item(Margins Menu), dropdown(Margins dropdown), and margin option(Narrow, Normal, Wide, etc). The system provides confirmation messages to the user upon successful completion of tasks and prompts to speak the command again if an action fails. This runtime component represents a significant advancement in making computer applications more accessible to blind users. By using advanced natural language processing and machine learning techniques, it bridges the gap between natural language commands and the complex interface navigation required by screen readers.

Chapter 4

Limitations and Future Work

The results of the user study by Kodandaram et al. [28] indicated a preference for activating the system with a wake-word rather than a push-to-talk button. Implementing this feature would involve addressing challenges related to continuous microphone activation, accurate wake-word detection, privacy concerns that comes with continuous microphone recording, and ensuring multi-platform compatibility.

Many practical computing tasks involve multiple applications. Enhancing Savant to handle such scenarios would require developing capabilities to reason across different application contexts, identify application-specific intents, and seamlessly execute operations across multiple interfaces. For example "Copy this text from Word document and paste it in Excel spreadsheet" is currently not possible using Savant as it detects only the application in focus and is bound in it.

Participants of the study also highlighted the need for Savant to manage tasks involving pop-ups and sub-windows. This presents challenges due to their dynamic nature, the complexity of relationships between various parent and child windows, and the context-dependent behavior of controls within these elements. Future work will focus on developing strategies to interact with these dynamic components effectively.

While Savant currently handles simple single-action commands, user feedback suggests a need for processing more complex instructions. For example, a command like "Insert a 3x5 table and make all the text bold" requires multiple actions. The user currently needs to execute "Insert a 3x5 table" after which they can execute "Make the text

bold". Implementing a task decomposition model could allow Savant to break down complex commands into a series of simpler, executable actions. Employing chain-of-thought prompting in LLMs [49] to break down complex commands into multiple simpler actions for this purpose presents an interesting avenue for future research. To increase the accuracy of Savant's actions with respect to the natural language commands provided by the user, techniques like chain-of-verifications proposed by Wiegreffe et al. [52] can also be explored, which would double-check the interpreted actions before execution. Another approach to explore would be self-consistency [47] on top of Chain of Thought prompting, which generates multiple reasoning paths and then aggregates them to produce more reliable responses from the LLM that would further improve Savant's understanding of reasoning and intent.

Given the proficiency of LLMs in understanding and translating non-English languages as evidenced by Zhu et al. [55], integrating translation functionalities into Savant could significantly broaden its accessibility for non-English speaking users. This expansion represents an exciting direction for future development.

Chapter 5

Conclusion

This thesis addresses the accessibility challenges encountered by low vision individuals when using desktop applications with screen readers. Previous studies have highlighted various difficulties faced by blind users, including the complexity of remembering numerous keyboard shortcuts and the struggle to interact with non-visible interface elements. To tackle these issues, we developed Savant, an interactive interface that harnesses the power of large language models to automate actions and execution of application controls on behalf of the user. This approach enables a consistent interaction method across diverse desktop applications, significantly simplifying the user experience for blind individuals.

The development process was iterative and user-centric, incorporating feedback from blind users at various stages. The resulting system demonstrates marked improvements in both, the efficiency of computer interactions and the overall usability of desktop applications for blind users. Savant represents a step forward in creating a more inclusive digital environment of the future. By reducing the cognitive load associated with navigating complex interfaces and providing a more intuitive interaction method, Savant has the potential to enhance the computing experience for blind users as well as the older demographic of users unfamiliar with popular applications on the desktop. This research contributes to the ongoing efforts to make technology more accessible and user-friendly for individuals with low vision, paving the way for further advancements in assistive technologies.

Bibliography

[1] Rudaiba Adnin and Maitraye Das. “"I look at it as the king of knowledge": How Blind People Use and Understand Generative AI Tools”. In: *The 26th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '24)*. 2024, pp. 1–14. DOI: 10.1145/3663548.3675631.

[2] Amazon. *Amazon Alexa*. Accessed: 2024-03-15. 2024. URL: https://www.amazon.com/alexa.

[3] Apple Inc. *Siri*. Accessed: 2024-03-15. 2024. URL: https://www.apple.com/siri/.

[4] Apple Inc. *Vision Accessibility - Mac*. Accessed: 2024. 2024. URL: https://www.apple.com/accessibility/mac/vision/.

[5] Vikas Ashok, Mohan Sunkara, and Satwik Ram. “Assistive Technologies for People with Visual Impairments”. In: (2023).

[6] Vikas Ganjigunte Ashok. “Non-Visual Web Browsing: From Accessibility with Screen Readers to Usability with Assistants”. PhD thesis. State University of New York at Stony Brook, 2018.

[7] Mark S Baldwin et al. “The tangible desktop: a multimodal approach to nonvisual computing”. In: *ACM Transactions on Accessible Computing (TACCESS)* 10.3 (2017), pp. 1–28. DOI: 10.1145/3075222.

[8] Be My Eyes. *Be My AI: AI-Powered Visual Assistant*. Accessed: 2024-03-15. 2023. URL: https://www.bemyeyes.com/blog/be-my-ai-announcement.

[9] Syed Masum Billah et al. “Accessible gesture typing for non-visual text entry on smartphones”. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 2019, pp. 1–12. DOI: 10.1145/3290605.3300606.

[10] Syed Masum Billah et al. “SteeringWheel: a locality-preserving magnification interface for low vision web browsing”. In: *Proceedings of the 2018 CHI conference on human factors in computing systems*. 2018, pp. 1–13.

[11] Syed Masum Billah et al. “Ubiquitous accessibility for people with visual impairments: Are we there yet?” In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 2017, pp. 5862–5868. DOI: 10.1145/3025453.3025731.

[12] Michael F Chiang et al. “Computer and world wide web accessibility by visually disabled patients: Problems and solutions”. In: *Survey of ophthalmology* 50.4 (2005), pp. 394–405. DOI: 10.1016/j.survophthal.2005.04.003.

[13] Iyad Abu Doush and Enrico Pontelli. “Non-visual navigation of spreadsheets: Enhancing accessibility of Microsoft Excel™”. In: *Universal Access in the Information Society* 12.2 (2013), pp. 143–159. DOI: 10.1007/s10209-012-0272-1.

[14] Javedul Ferdous et al. “Enabling Efficient Web Data-Record Interaction for People with Visual Impairments via Proxy Interfaces”. In: *ACM Transactions on Interactive Intelligent Systems* 13.3 (2023), pp. 1–27. DOI: 10.1145/3563824.

[15] Javedul Ferdous et al. “InSupport: Proxy interface for enabling efficient non-visual interaction with web data records”. In: *27th International Conference on Intelligent User Interfaces*. 2022, pp. 49–62. DOI: 10.1145/3490099.3511159.

[16] Freedom Scientific. *JAWS: Job Access With Speech*. Accessed: 2024. 2024. URL: https://www.freedomscientific.com/products/software/jaws/.

[17] Freedom Scientific. *Picture Smart AI: How We Got Here*. Accessed: 2024-03-15. 2024. URL: https://blog.freedomscientific.com/picture-smart-ai-how-we-got-here/.

[18] Freedom Scientific. *ZoomText Magnifier/Reader*. Accessed: 2024. 2024. URL: https://www.freedomscientific.com/products/software/zoomtext/.

[19] Anujay Ghosh et al. “Screen Reading Enabled by Large Language Models”. In: *Proceedings of the 26th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '24)*. 2024, pp. 1–5. DOI: 10.1145/3663548.3688491.

[20] Google. *Google Assistant*. Accessed: 2024-03-15. 2024. URL: https://assistant.google.com/.

[21] Yoichi Haga et al. “Dynamic braille display using sma coil actuator and magnetic latch”. In: *Sensors and Actuators A: Physical* 119.2 (2005), pp. 316–322.

[22] Simon Harper and Alex Q Chen. “Web accessibility guidelines: A lesson from the evolving Web”. In: *World Wide Web* 15.1 (2012), pp. 61–88. DOI: 10.1007/s11280-011-0130-8.

[23] Monica Hegde. “User Experience Study of Screen Readers for Visually Challenged Users”. Purdue University Graduate School, 2023. DOI: 10.25394/PGS.23750877.v1.

[24] Md Touhidul Islam, Donald E Porter, and Syed Masum Billah. “A Probabilistic Model and Metrics for Estimating Perceived Accessibility of Desktop Applications in Keystroke-Based Non-Visual Interactions”. In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 2023, pp. 1–20. DOI: 10.1145/3544548.3581400.

[25] S. Kırboyun Tipi. "How Screen Readers Impact the Academic Work of College and Graduate Students with Visual Impairments". In: *Sakarya University Journal of Education* 13.3 (2023), pp. 416–434. DOI: 10.19126/suje.1201482.

[26] Yu-Jung Ko et al. "Modeling Gliding-based Target Selection for Blind Touchscreen Users". In: *Proceedings of the 23rd International Conference on Mobile Human-Computer Interaction*. 2021, pp. 1–14. DOI: 10.1145/3447526.3472022.

[27] Satwik Ram Kodandaram et al. "Detecting Deceptive Dark-Pattern Web Advertisements for Blind ScreenReader Users". In: *Journal of Imaging* 9.11 (2023), p. 239. DOI: 10.3390/jimaging9110239.

[28] Satwik Ram Kodandaram et al. "Enabling Uniform Computer Interaction Experience for Blind Users through Large Language Models". In: *The 26th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '24)*. 2024, pp. 1–14. DOI: 10.1145/3663548.3675605.

[29] Satwik Ram Kodandaram et al. "Unveiling Coyote Ads: Detecting Human Smuggling Advertisements on Social Media". In: *Proceedings of the 35th ACM Conference on Hypertext and Social Media*. 2024, pp. 259–272.

[30] Barbara Leporini, Maria Claudia Buzzi, and Marina Buzzi. "Interacting with mobile devices via VoiceOver: usability and accessibility issues". In: *Proceedings of the 24th Australian Computer-Human Interaction Conference*. 2012, pp. 339–348. DOI: 10.1145/2414536.2414591.

[31] Zhi Li et al. "Select or Suggest? Reinforcement Learning-based Method for High-Accuracy Target Selection on Touchscreens". In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 2022, pp. 1–15. DOI: 10.1145/3491102.3517472.

[32] Mei Miao et al. "Contrasting usability evaluation methods with blind users". In: *Universal Access in the Information Society* 15 (2016), pp. 63–76. DOI: 10.1007/s10209-014-0378-8.

[33] Microsoft. *Microsoft UI Automation*. Accessed: 2024-03-15. 2024. URL: https://docs.microsoft.com/en-us/windows/win32/winauto/entry-uiauto-win32.

[34] MM Moncy et al. "Evaluation of accessibility of open-source EHRs for visually impaired users". In: *AMIA Annual Symposium Proceedings* 2023 (2024). PMID: 38222344; PMCID: PMC10785889, pp. 1165–1174.

[35] Lourdes Morales, Sonia M Arteaga, and Sri Kurniawan. "Design guidelines of a tool to help blind authors independently format their word documents". In: *CHI'13 Extended Abstracts on Human Factors in Computing Systems*. 2013, pp. 31–36. DOI: 10.1145/2468356.2468363.

[36] NV Access. *NVDA: NonVisual Desktop Access*. Accessed: 2024. 2024. URL: https://www.nvaccess.org/.

[37] OpenAI. *ChatGPT: Optimizing Language Models for Dialogue*. Accessed: 2024-03-15. 2022. URL: https://openai.com/blog/chatgpt/.

[38] W Reuschel, M McDonnall, and D Burton. "The Accessibility and Usability of Online Job Applications for Screen Reader Users". In: *Journal of Visual Impairment and Blindness* 117.6 (2023), pp. 479–490. DOI: 10.1177/0145482x231216757.

[39] Anastasia Schaadhardt, Alexis Hiniker, and Jacob O. Wobbrock. "Understanding Blind Screen-Reader Users' Experiences of Digital Artboards". In: *CHI Conference on Human Factors in Computing Systems (CHI '21)*. New York, NY, USA: ACM, 2021, pp. 1–19. DOI: 10.1145/3411764.3445242.

[40] Kristen Shinohara and Josh Tenenberg. "Observing Sara: a case study of a blind person's interactions with technology". In: *Proceedings of the 9th international ACM SIGACCESS conference on Computers & accessibility*. 2007, pp. 171–178. DOI: 10.1145/1296843.1296873.

[41] Reeta Singh. "Blind handicapped vs. technology: How do blind people use computers". In: *International Journal of Scientific & Engineering Research* 3.4 (2012), pp. 1–7.

[42] Mohan Sunkara et al. "Assessing the Accessibility and Usability of Web Archives for Blind Users". In: *International Conference on Theory and Practice of Digital Libraries*. Springer. 2024, pp. 203–221.

[43] Mohan Sunkara et al. "Enabling Customization of Discussion Forums for Blind Users". In: *Proceedings of the ACM on Human-Computer Interaction* 7.EICS (2023), pp. 1–20. DOI: 10.1145/3610410.

[44] J Thatcher. "Screen reader/2: Access to os/2 and the graphical user interface". In: *Proceedings of the First Annual ACM Conference on Assistive Technologies*. 1994, pp. 39–46. DOI: 10.1145/191028.191037.

[45] Utku Uckun et al. "Taming User-Interface Heterogeneity with Uniform Overlays for Blind Users". In: *Proceedings of the 30th ACM Conference on User Modeling, Adaptation and Personalization (UMAP '22)*. Association for Computing Machinery. New York, NY, USA, 2022, pp. 212–222. DOI: 10.1145/3503252.3531317.

[46] Alessandro Diogo Vieira, Higor Leite, and Ana Vitória Lachowski Volochtchuk. "The impact of voice assistant home devices on people with disabilities: A longitudinal study". In: *Technological Forecasting and Social Change* 184 (2022), p. 121961. ISSN: 0040-1625. DOI: https://doi.org/10.1016/j.techfore.2022.121961. URL: https://www.sciencedirect.com/science/article/pii/S0040162522004826.

[47] Xuezhi Wang et al. “Self-Consistency Improves Chain of Thought Reasoning in Language Models”. In: *arXiv preprint arXiv:2203.11171* (2022). DOI: 10.48550/arXiv.2203.11171.

[48] Tetsuya Watanabe, Shinichi Okada, and Tohru Ifukube. “Development of a gui screen reader for blind persons”. In: *Systems and Computers in Japan* 29.13 (1998), pp. 18–27. DOI: 10.1002/(SICI)1520-684X(19981130)29:13<18::AID-SCJ3>3.0.CO;2-R.

[49] Jason Wei et al. “Chain-of-thought prompting elicits reasoning in large language models”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 24824–24837. DOI: 10.48550/arXiv.2201.11903.

[50] Brian Wentz, Harry Hochheiser, and Jonathan Lazar. “A survey of blind users on the usability of email applications”. In: *Universal Access in the Information Society* 12 (2013), pp. 327–336. DOI: 10.1007/s10209-012-0285-9.

[51] Brian Wentz and Jonathan Lazar. “Usability evaluation of email applications by blind users”. In: *Journal of Usability Studies* 6.2 (2011), pp. 75–89.

[52] Sarah Wiegreffe et al. “Reframing Human-AI Collaboration for Generating Free-Text Explanations”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 2023, pp. 14720–14743. DOI: 10.18653/v1/2022.naacl-main.47.

[53] Cheng Xu et al. “Tactile display for the visually impaired using teslatouch”. In: *CHI'11 Extended Abstracts on Human Factors in Computing Systems*. 2011, pp. 317–322. DOI: 10.1145/1979742.1979705.

[54] Levent Yobas et al. “A novel integrable microvalve for refreshable braille display system”. In: *Journal of microelectromechanical systems* 12.3 (2003), pp. 252–263. DOI: 10.1109/JMEMS.2003.811754.

[55] Wenhao Zhu et al. “Multilingual Machine Translation with Large Language Models: Empirical Results and Analysis”. In: *Findings of the Association for Computational Linguistics: NAACL 2024*. Mexico City, Mexico: Association for Computational Linguistics, 2024. DOI: 10.18653/v1/2024.findings-naacl.176.

[56] Hong Zou and Jutta Treviranus. “ChartMaster: A tool for interacting with stock market charts using a screen reader”. In: *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility*. 2015, pp. 107–116. DOI: 10.1145/2700648.2809862.

Appendix A

Appendix

A.1 Seed LLM Example

The prompt given to Seed LLM: "Given a computer application and a control-value pair, generate a natural language command that a user might say to perform the action represented by the control-value pair. The command should be concise, clear, and in the imperative form. Examples are as follows:"

TABLE A.1: Prompt and Response for the Seed LLM

Application	**Control-Value pair**	**Response**
Spotify	(enable shuffle, none)	Shuffle the current queue
Wordpad	(strikethrough, none)	Strike the selected text
Excel	(align, align right)	Align the selected text to the right
Notepad	(edit, cut)	Cut the selected text
Word	(font size, 20)	*Change the font size to 20*

Note: The last row (highlighted) represents the output generated by LLM (italicized).

A.2 Gen LLM Example

The prompt given to Gen LLM: "Given a computer application and a natural language command, generate a control-value pair in the form of "(control element, value)" Examples are as follows:"

TABLE A.2: Prompt and Response for the Gen LLM

Application	Natural Language Command	Response
Word	Insert a 4x5 table	(table, 4x5 table)
Word	Insert a table with 4 columns and 5 rows	(table, 4x5 table)
Word	Add a 7x20 table	(table, 7x20 table)
Word	Add a table with 5 rows and columns	(table, 5x5 table)
Word	Create a table with 10 rows and 2 columns	*(table, 2x10 table)*

Note: The last row (highlighted) represents the output generated by LLM (italicized).

A.3 Applications crawled for FED

TABLE A.3: Applications crawled to create the FED dataset

Application
Amazon (Opened in Firefox)
Excel
Explorer
Gmail (Opened in Chrome)
Google Docs (Opened in Chrome)
Notepad
Outlook
Spotify
Word
Wordpad
Zoom