Enabling Natural Language Voice-to-Code Dictation in Talon Voice with LLMs

Jarad Forristal*
jforristal@ucsd.edu
UC San Diego
La Jolla, California, USA

Ilana Shapiro*
ilshapiro@ucsd.edu
UC San Diego
La Jolla, California, USA

Abstract

A frequently overlooked subpopulation of computer science students is those who face physical disabilities that interfere with their ability physically interface with a keyboard and mouse and thus write code. Such difficulties add to the already significant cognitive burden both these students and their physically-abled peers when learning new content and apply it to course projects. These kinds of disabilities can also have a profound effect on their ability to succeed in the software engineering field. Voice-to-code technologies have been developed to alleviate this burden, such as Talon Voice, an open-source, multi-platform software that enables the user to vocally dictate commands and code. However, to our knowledge, neither Talon nor any other existing voice-to-code software supports natural language commands; rather, users must memorize a specific list of English commands and cannot deviate from them. This further adds to the cognitive load of the user, particularly those whom English is not their first language. To that end, we propose and implement an extension to Talon Voice that enables natural language dictation via an LLM-based interpreter. We conclude by outlining studies (both user-oriented and automated) that we plan to execute to evaluate our tool.

CCS Concepts

• **Human-centered computing → Accessibility systems and tools**; **Accessibility technologies**; • **Social and professional topics → Computing literacy**.

Keywords

Accessibility, Human-Centered AI, Human Factors in Computing, Software Engineering, Computer Science Education

ACM Reference Format:
Jarad Forristal and Ilana Shapiro. 2018. Enabling Natural Language Voice-to-Code Dictation in Talon Voice with LLMs. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

*Both authors contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference acronym 'XX, Woodstock, NY
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Computers are the primary interface through which many people today complete academic, personal, and professional tasks. From filling out forms to sending emails, browsing course websites, writing code: effective computer use is now a functional requirement for participation in many areas of modern life. Yet, standard graphical user interfaces are still built with the consumption of direct manipulation through a keyboard and mouse. For many users, especially those with disabilities and limited motor ability, this interaction model is inefficient, fatiguing, or impossible. Assistive technologies such as speech input, screen readers, and keyboard-centric navigation tools partially address this gap, but do not alleviate it.

Recent progress in large language models suggests a possible different route: rather than directly requiring users to execute every interaction, an agent may be able to interpret high level intent and carry out actions on behalf of the user. A growing body of literature is exploring this idea within the context of graphical user interfaces, in digital environments more broadly. Prior work has demonstrated that agents can perceive application state, perform advanced tasks, and interact with desktop environments, browsers, and software [1, 2, 5]. At the same time, existing research has highlighted substantial limitations [3, 11, 12]. High latency, poor reliability, brittleness and visual grounding, and long execution traces remain just some of many critiques. These issues become extremely important within the setting of accessibility, where tools must not only function but also serve reliably enough to reduce user burden.

This problem is especially important in the context of education. College students, especially those in introductory computing courses, are expected to perform a huge range of computing tasks within varying applications and environments. This requires both general purpose desktop interaction and specialized navigation of technical tools like VSCode. If a student also has limited keyboard and mouse accessibility, the barriers compound. Ordinary "easy" tasks become herculean, everything becomes slower and more cumbersome, and the ability to not only complete the tasks but achieving the course learning outcomes that all students are expected to meet becomes nearly impossible.

In this paper, we present a computer use agent designed to support users who struggle with standard input devices. Our system is designed to complement, rather than replace, existing tools that are popular in this community, such as Talon and Vimium. The goal is to allow users to offload portions of interactions directly to computer use agents rather than having to directly accomplish them themselves. Talon provides a sturdy foundation to build upon. Talon provides voice-mediated accessible access to the desktop

via powerful commands and user/community-written application-specific integrations. This is where our agent sits: integrated into and as a piece of Talon, rather than as a standalone computer use agent. Our goal is to both reduce cognitive burden to Talon users (especially computing students), and increase the breadth of tasks users can accomplish with Talon.

2 Related Work

2.1 Prior Work on Voice-Mediated Coding Tools

Coding by voice, as a concept, has existed for decades. One of the earliest contributions is a design for a system from the year 2000 that generates environments enabling people to program by voice. Arnold et al designed VocalGenerator, a tool that takes a programming language's grammar and produces a voice-controlled, syntax-directed programming environment. Via the grammar, their system supports code navigation and automatic completion of code constructs. They argue that VocalGenerator naturally provides support for entering data and writing XML documents by voice [1].

Begel and Graham built upon this foundation in a 2005 study of how well programmers can write/edit code using speech instead of typing in the context of repetitive-strain injuries or motor impairments. They conducted a preliminary investigation about how programmers verbally describe their coding intentions, they found "spoken code" introduces ambiguities, both syntactic and semantic. To address this, they created Spoken Java, which includes optional punctuation and natural language words. Then, they built SPEED, an Eclipse plugin that supports spoken commands, spoken Java code, and program-analysis-based disambiguation of speech input. Via a user study, they found that users of SPEED could quickly learn spoken commands to successfully write code, but speech recognition errors made the system slower and more frustrating than actual typing. They found that voice works better for high-level actions than low-level syntax. However, speech-based programming could still be valuable for accessibility settings, and has clear room for improvement with better speech recognition engines and interface design [2].

More recently, researchers have investigated how professional developers with physical disabilities use voice coding. Creed and Sayan found that these developers rely on highly customized and multimodal setups using tools like Talon Voice, gaze tracking, etc, which leads to challenges such as speech-recognition errors, having to learn new commands, and difficulty with tool navigation and editing. To that end, they developed a novel voice-coding prototype based on CodeMirror, an open-source code editor, and conducted a first study with 7 disabled developers. Their new prototype used fixed grammar approaches with natural language commands, which coders found slow and overly verbose – coders preferred customizable commands, command chaining, context awareness, and multimodal support. The takeaway is that voice coding is essential for accessibility but must be flexible, customizable, and integrable with other support tools [3].

In a similar vein, Eknefelt and Oskar explore coding by voice in the reactive, as opposed to the overarching imperative, coding paradigm. They developed and tested a vocal programming environment that recognizes RxJava (a framework for Java that enables reactive programming) operators and commands. They wrote a custom RxJava-adapter for a voice-to-code software called Serenade. Like Talon Voice, Serenade has no dedicated support for RxJava. However, it does provide the user with an API for creating custom commands. Eknefelt and Oskar created a library of commands for RxJava in both Serenade and Talon Voice to improve programming-by-voice in the reactive paradigm. They find that the Serenade commands reduce the number of words required to produce the example code by 19%, reduce the number of words that only produce a single character by 42%, and require 45% fewer words than using Talon Voice to produce the same example code. Talon also required more than six times as many words that only produce a single character, compared to the default version of Serenade. Overall, this study shows that Serenade is more suited to the reactive programming paradigm than Talon Voice is [5].

These findings may be due to fundamental differences in the imperative vs reactive coding paradigms; namely, that reactive coding often requires more written code than imperative does. Lagergren and Soneryd investigated this phenomenon and revisit Talon Voice's efficacy in the reactive and imperative domains. They conducted a study of programmers with physical disabilities that affect hand and arm motion, such as repetitive strain injury or carpel tunnel syndrome. They identified common issues with writing code by voice and survey several popular voice-to-text tools and their attempts to face current challenges within the field. Using Talon Voice, they compared voice scripts for code in the imperative and reactive paradigms (Java and RxJava) and collected data to measure vocal and cognitive load. They found that the results were similar for the two paradigms, but the reactive code required roughly 5% more characters of code and had more necessary pauses [9].

Coding by voice has even reached the robotics domain, opening up new avenues for robot-based accessibility scenarios. Ionesco and Schlund developed a new voice-based programming approach and software framework for collaborative robotic arms (cobots), which are frequently used to automate repetitive manual tasks. Their approach is based on the Web Speech API (WSAPI), a W3C specification published and maintained by the Speech API Community Group supported by modern web browsers. Their framework targets human programmable interfaces (HPIs), which can be used by novice coders via a voice-directed meta-programming approach. Upon a voice instruction (such as move, pick, release, etc.), the tool automates the manual tasks required to manipulate the vendor-provided HPI. Ionesco and Schlund felt that the main advantage of their framework is simplified, guided programming (which only requires the knowledge of 5–10 voice instructions and increased cobot programming speed by up to 46% compared to manual coding), and the possibility of sharing programs as videos [7].

2.2 Experiences of Physically Disabled Developers

Significant research has also been conducted to evaluate the experiences and needs of physically disabled developers and computing students. Desai et al examined disability culture through interviews of disabled innovators that created popular accessibility technologies. They argue that disability culture (which extends beyond traditional narratives of barriers faced by disabled people to also

recognize disabled people's many acts of resistance, creativity, and connection in everyday life) has the potential to reshape how accessibility research and design are approached. They identify examples of access technology that are grounded in disabled experiences and values, and unencumbered by ableist biases. To that end, Desai et al presented accounts of seven innovators of accessibility technologies, and contextualized their design choices through the lens of disability culture. They found that the seven innovators exemplify a liberatory approach to access by embodying goals of increasing autonomy, sharing power, and affecting structural change. They propose that accessibility technologies designed by disabled people should be considered as artifacts of disability culture, in how they both embody cultural values and facilitate cultural processes [4].

Such theories regarding disability culture find a practical application in the lived experiences of computing students. Ladner et al examined the experiences of CS students and recent grads with disabilities and explored the difficulties they face in CS education and careers, as well as the types of support needed to help them succeed. They found that students report challenges such as inaccessible programming tools, difficulties in getting accommodation, limited mentorship opportunities, and negative attitudes from peers and instructors, despite legal support dictated by disability–rights legislation [8].

These challenges persist as students embark on professional career journeys. Nowrin et al investigated the preferences of seven programmers who depend on voice programming interfaces for their careers. They explored the programmers' thoughts on current voice programming systems, the difficulties they encountered while using voice to enter programs, and what they expect from a future voice programming system. Their interviews showed that despite the existence of several voice programming systems, motor-impaired programmers hoped for a more natural and purpose-built solution. The interviews also disabled programmers' frustrations with the accuracy of current systems, as well as the challenges in dictating code. They believe a practical and naturally spoken programming system is a natural next step given recent advances in speech and natural language processing [12].

Norwin continued this work by investigating how programmers "speak code" (i.e. without teaching them a specific grammar) in order to help inform the design of more effective and robust voice-based systems. They conducted the first comparison of two possible approaches in collecting spoken code: speaking a missing line of code, and speaking a highlighted line of code. They found that programmers tended to speak faster using natural language, thus motivating the need for a naturally spoken programming system, but that standard speech recognizers had a high error rate due to the mismatch between the written English they were trained on and how people speak code [11].

The body of prior work converges on one critical message: flexible voice coding tools supporting natural language dictation are greatly needed by the community of physically disabled computer scientists, yet there is a lack of such technology in production. We aim to address this gap in our work.

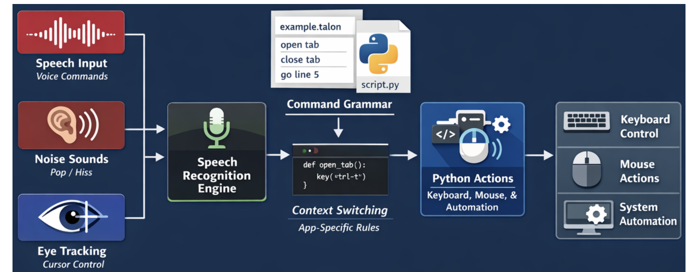

Figure 1: Overview of Talon Architecture

3 Talon Voice

Talon Voice is a system enabling hands-free interaction with applications and the operating system. It allows users to control their computer entirely through voice commands and/or eye tracking. Talon is flexible across platforms: it works across macOS, Windows, and Linux. It was originally designed to help people with physical disabilities such as repetitive strain injuries, motor impairments, or neurological conditions that make traditional keyboard and mouse use difficult. For our project, Talon is especially relevant because it improves accessibility in computing, allowing physically disabled students to write code and interact with development tools using voice instead of manual input [13].

Talon is relatively unique among other voice-coding tools due to its large open source community. Though Talon Voice itself is not open source, its support for user scripts has created an active open source community featuring a robust selection of extensions. These range in size and scope, from a simple interface for a single application, all the way through to complex projects like metaprogramming designed for programming languages while speaking languages, and even to advanced OCR-and-gaze based text editing. Talon is therefore well-suited for our contribution, both due to the ease of contributing and the direct access to the community that engages with it.

3.1 Talon Architecture

Talon works as a pipeline that converts user input into executable computer actions. It has five layers, as detailed in Figure 1. First, First, it captures multimodal input such as spoken commands, short noise signals like pops or hisses, and optionally eye-tracking data in real-time. That audio is then processed by a speech-to-text engine, which converts the speech into text tokens representing what the user said. Talon then performs context-aware grammar resolution, meaning the recognized text is matched against predefined `.talon grammar` rules that are filtered contextually based on the active application. When a rule matches, Talon invokes the associated Python action in its runtime. Finally, those actions are translated into low-level automation events such as keyboard presses, mouse movements, or operating system commands [14].

3.2 Talon Limitations

The Talon Voice engine has a number of limitations, some of which are purposeful. One of these is a lack of on-screen dependent abilities. This is purposeful: Talon does not handle things at that layer.

Figure 2: Our Solution in Talon's Architecture

It is designed to be API/keyboard focused, not to try handling everything. Another limitation is the extremely strict grammar. The user is expected to memorize every command in rigid syntax. This is a feature, not an inherent issue; one exact command string maps to exactly one action. However, creates a huge learning curve for the user, and no room for error in spoken commands, which places an increased burden on a novice computing student to both absorb new course content and recall the precise Talon commands that allow them to apply their course content in practice. In particular, non-native English speakers may face an increased struggle. We address these limitations by adding an LLM-based interpreter to Talon Voice that enables natural language dictation. In theory, the flexibility of our tool's LLM should support multiple languages (such as Spanish or Chinese), but, as sole English speakers, we were unable to test this in the scope of this project. [1]

4 Our Solution

Our agent architecture is relatively simple. We added an LLM interpreter layer between Talon's speech regocnition engine layer and the command grammar layer (Figure 2). We provide an LLM with a robust system prompt detailing that it should not act as a conversational assistant but instead as a deterministic controller of an automation system. The prompt also specifies the available tools it can call, the required JSON output format, and its behavioral constraints. Our system supports both local models (via OpenAI-compatible servers such as Ollama or vLLM) as well as more robust remote models if the user has access to a paid OpenAI API subscription. The model is instructed to produce a structured automation plan consisting of steps like key presses, text insertion, delays, or calls to registered Talon actions. These plans are validated against a strict JSON schema before execution to ensure safety (i.e. so the user cannot accidentally delete their entire documents folder or share secure information with a remote source).

The more complex part of our project was the search function (i.e. our Talon action discovery mechanism). Talon exposes an extensive library of user actions, spoken commands, and editor integrations. If we allow the LLM to call arbitrary actions, we must first determine the action that corresponds to the user's command. This requires that our system is able to efficient search a set of very large candidates for the desired action. However, our application is extremely constrained by latency, as large pauses between user prompt an action significantly decrease usability. Therefore, we cannot give a large prompt, nor can we have a bad search function that takes many iterations.

Consequently, two constraints arise: (1) the prompt sent to the LLM must remain concise and (2) the search function must return only a small number of highly likely candidate actions. There is thus a fundamental tradeoff in our system between recall and latency. If the search function returns many candidate actions, the correct action is likely included (high recall), but the resulting prompt becomes too large, and slows everything down. Conversely, if the search is aggressively pruned to produce a very small number of candidates to optimize for performance and reduce latency, the correct action may not appear in the candidate list, preventing the agent from completing the task.

To accomplish this, we tried a variety of methods (simple sub string search, fuzzy matching, keyword search, TF-IDF/Term Frequency-Inverse Document Frequency), all with relatively poor performance. Substring and keyword matching failed when the user phrasing differed from the action name. TF-IDF improved recall, but frequently returned large candidate sets that had many irrelevant results.

A hybrid method using Okapi BM25 and an embedding reranking model turned out to be the best option [10]. BM25 is a probabilistic information retrieval model in the BM (Best Matching) family of algorithms. It is an improved version of the traditional TF-IDF approach, and is widely used in modern search engines and databases. In our implementation, we tokenize action names, documentation strings, and function signatures to build a BM25 index over all available Talon actions. At query time, the spoken command is tokenized and scored against every indexed action using the BM25 scoring function.

BM25 has extremely high recall, but can still return many candidates. To reduce the candidate set while preserving semantic relevance, we proceed to a second step using embedding-based reranking. First, we select the top BM25 candidates (typically around 40). We then compute vector embeddings for the query and each candidate description using a lightweight embedding model. Cosine similarity is used to measure semantic closeness between the query and each candidate. The final ranking score is computed as a weighted combination of normalized BM25 scores and embedding similarity scores. This allowed us to present only a few options to the model, optimizing for latency, while simultaneously achieving extremely high recall. Please see Figure 3 for a detailed visualization of our implementation's pipeline in relation to Talon's speech recognition and command layers.

5 Proposed Tool Evaluation

5.1 User Study

To evaluate our tool, we intend to conduct a user study targeting college-aged students in introductory computer science courses. The study will have two stages, one focused on general application usage, and another on programming specifically. There will be two groups, one that is able to interact with computers with a keyboard and mouse, and those that have limited access to the keyboard/mouse (only when necessary) combined with access to Talon, Vimium (a voice-based browser navigator), a brief tutorial

[1] Our GitHub repository with our tool's code is available at https://github.com/jaradfor/talon-LLM-integration

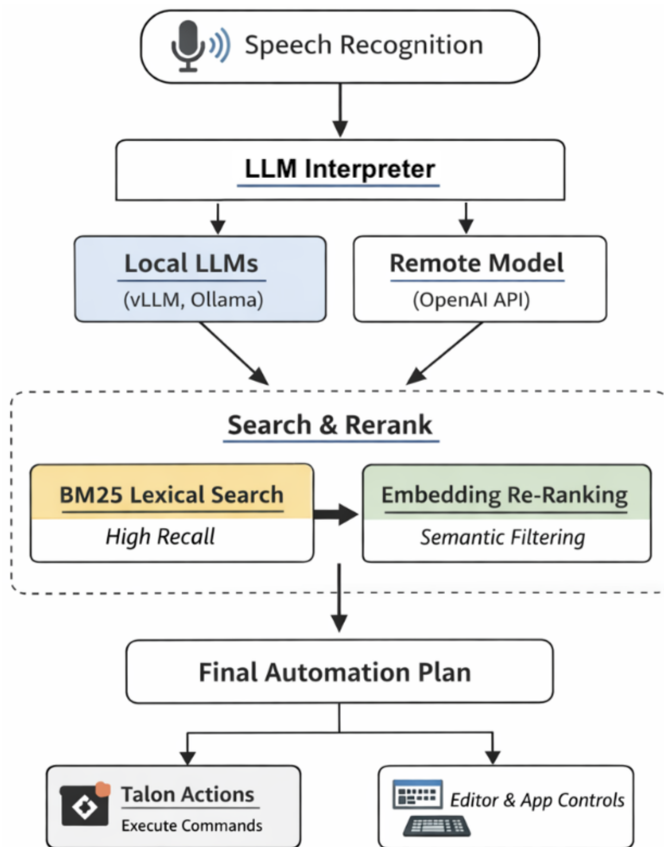

Figure 3: Details of Our Implementation

on all relevant tooling, and access to our agent. Participants in both groups will be instructed to verbally express their thoughts as they occur (with regards to the tasks), any issues they encounter, as well as what they find easy/trivial. Participants will be randomly assigned into groups, with prior experience being balanced via stratified assignment.

In the first stage of the evaluation, we will instruct the participants to do a number of "standard" computer tasks: create a spreadsheet, edit a word document, send a mock email, complete a canvas quiz, and fill out a form. We will collect the following metrics for each participant: the task type, stage, completion status, completion time, number of interventions, number of high-level actions, number of failures and recoveries, and any subjective ratings. We will also ask participants to think aloud, focusing on any issues encountered along the way, and will record all transcripts.

In the second stage, participants will be presented with a prompt detailing how to perform several "easy" to "medium" difficulty programming tasks in a language they know. There will be a prepared VSCode environment in a virtual machine setup for them. Participants will be asked to think aloud. Like before, the mentioned key metrics alongside a detailed transcription of everything spoken will be recorded.

Our primary quantitative comparison is time to completion, with a secondary qualitative analysis. As such, this is a mixed-methods study. Our strongest outcomes will be completion rate, time-on-task, assistance burden, efficacy, and self-efficacy. With respect to the gathered think-aloud transcripts, we will qualitatively code the emissions and perform thematic analysis. We will use categories such as planning, confusion, error diagnosis, recovery behavior, agent trust, and perceived ease/frustration.

5.2 Automated Evaluation

Separate from the user study, we also present a plan to evaluate our agent on standard computer-use benchmarks. We will filter for benchmarks that only focus on applications with the Talon user-community support, and further filter for tests that have a corresponding implementation inside of the Talon user-community.

AssistGUI[6] introduces a benchmark for evaluating general-purpose LLMs on their ability to complete Windows-based tasks. It leverages multi-agent collaboration and task decomposition. The key novelties are a suite of 100 tasks over nine different popular Windows applications, a multi agent, collaboration framework, as well as a thorough bench marking of models within this framework. The best models tested only reported success rates of around 46%[6].

OSWorld[15] introduces an execution-based benchmark/ environment for open-ended computer tasks across multiple operating systems, with hundreds of tasks covering diverse applications and workflows. A core novelty is its emphasis on reproducibility through controlled initial states and automated evaluation procedures, enabling direct measurement of end-to-end completion rather than only intermediate correctness. Results show a large gap between current agents and humans, particularly on tasks requiring robust GUI grounding and visual understanding. The authors explicitly note that understanding of graphical user interfaces is a key bottleneck.

We will evaluate completion rates, latency, and token usage and compare it to other state of the art computer usage models. We hope that our integration with existing Talon user-community scripts not only improves reliability and the rates of completion, but leads to a more private agent that does not require access to the screen. We will use both AssistGUI and OSWorld as a source of tasks.

6 Limitations of Our Approach

Currently, applications of our tool must have Talon user-community implementations that already exist. For instance, if no Python action has been written in the user-community to tell the OS to open a new browser, then no command we give our tool will be able to execute this action. To overcome this, we are experimenting with allowing the agent to also write these implementations, though this will require careful tuning of safety constraints so as not to allow the agent unrestricted control over the user's OS.

With no access to the screen comes privacy benefits, however some applications are very mouse-centric and require precision clicking. This itself is an accessibility issue: blind users of applications cannot use mice, and as such, we hope that over-time all application become possible to use completely without a mouse, and, in turn, by Talon actions.

7 Conclusions

We have successfully enabled natural language dictation in Talon Voice by adding an LLM interpreter layer to Talon's architecture. The action search function underlying our interpreter (using the BM25 algorithm and semantic reranking via embeddings) gives a balanced tradeoff between accuracy and latency. By not requiring screen-capture, we have also created a more private type of computer assistant, and by using Talon as opposed to other voice-coding agents, we gain the strengths of the huge user-community repository. We are now able to offer the voice coding community a relatively private, low-latency computer-use agent that requires substantially less tokens than existing methods.

In the future, we hope to conduct a user study to demonstrate that this new paradigm of computer-use agents, ones that extend the flexibility of existing APIs with off-the-shelf LLM-based interpreters, can be more reliable and faster than ones that require unbounded access to the user's input devices and high-reasoning models. We also intend to increase the flexibility of our model by carefully enabling the agent to write its own OS commands, albeit within strict safety constraints. Our hope this that these efforts will greatly lower the burden and cognitive overload to novice computing students who face physical disabilities that impair their ability to code.

References

[1] A. Arnold et al. 2000. Programming by Voice: VocalProgramming. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. doi:10.1145/354324.354362

[2] Andrew Begel and Susan L. Graham. 2005. An Assessment of a Speech-Based Programming Environment. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. doi:10.1109/VLHCC.2005.32

[3] Chris Creed and Sayan Sarcar. 2023. Voice Coding Experiences for Developers with Physical Impairments. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. https://www.open-access.bcu.ac.uk/15942/1/Creed_Voice_Coding_Experiences_Disabled_Developers%20%28Author%20Accepted%20Manuscript%29.pdf

[4] Anushka Desai et al. 2024. Exploring Disability Culture Through Accounts of Disabled Innovators of Accessibility Technology. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. doi:10.1145/3663547.3746330

[5] Anton Eknefelt and Oskar Nilsson. 2022. *Vocal Reactive Programming: Enabling RxJava*. Master's thesis. KTH Royal Institute of Technology. https://www.diva-portal.org/smash/get/diva2:1691627/FULLTEXT01.pdf

[6] Difei Gao, Lei Ji, Zechen Bai, Mingyu Ouyang, Peiran Li, Dongxing Mao, Qinchen Wu, Weichen Zhang, Peiyi Wang, Xiangwu Guo, Hengxu Wang, Luowei Zhou, and Mike Zheng Shou. 2024. AssistGUI: Task-Oriented PC Graphical User Interface Automation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[7] Andrei Ionesco and Sebastian Schlund. 2020. Programming Cobots by Voice: A Human-Centered, Web-Based Approach. *Procedia CIRP* 97 (2020), 460–465. doi:10.1016/j.procir.2020.05.255

[8] Richard E. Ladner et al. 2021. Experiences of Computing Students with Disabilities. In *Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE)*. doi:10.1145/3408877.3432574

[9] Anton Lagergren and Filip Soneryd. 2021. *Programming by Voice: Efficiency in the Reactive and Imperative Paradigm*. Master's thesis. KTH Royal Institute of Technology. https://www.diva-portal.org/smash/get/diva2:1617714/FULLTEXT01.pdf

[10] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2009. Okapi BM25: A Non-binary Model. In *Introduction to Information Retrieval*. Cambridge University Press, Chapter 11. https://nlp.stanford.edu/IR-book/html/htmledition/okapi-bm25-a-non-binary-model-1.html Accessed: 2026-03-08.

[11] Sadia Nowrin. 2023. Programming by Voice: Exploring User Preferences and Speaking Styles. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. doi:10.1145/3571884.3597130

[12] Sadia Nowrin et al. 2022. Exploring Motor-Impaired Programmers' Use of Speech Recognition. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. doi:10.1145/3517428.3550392

[13] Talon Community. 2026. Talon Community Wiki. https://talon.wiki/. Accessed: 2026-03-09.

[14] Talon Voice. 2026. *Talon Documentation*. https://talonvoice.com/docs/#overview

[15] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. *arXiv preprint arXiv:2404.07972* (2024).

Received 8 March 2026; revised 12 March 2026; accepted 5 June 2026