Journal of Kerbala University, Vol. 22, Issue 4, December , 2025

Available at https://www.iasj.net/iasj

Iraqi Academic Scientific Journals

Journal homepage:
https://journals.uokerbala.edu.iq/index.php/UOKJ

Research Article

Speech-Driven Execution of Windows Operating System Commands for Users with Motor Impairments: A Sphinx-4–Prototype

1,Meeras Salman Juwad Al-Shemarry 2,Dawood Sallem Hussian 3,Ahmed F. Almukhtar

1,3,Department of Information Technology, College of Computer Science and Information Technology, University of Kerbala, Karbala, Iraq

2,Computer Technology Engineering

Al Taff University College, karbala, Iraq

Article Info

Article history:
Received 5 -10-2025
Received in revised form 20-10-2025
Accepted 3-11-2025
Available online 31 -12 -2025

Keywords: Speech recognition, human–computer interaction, accessibility, Sphinx-4, JSGF, Java script, Windows commands.

Abstract:

This paper describes a practical speech-driven framework that lets users with motor impairments run Windows operating system commands by speaking them. The system uses the open-source Sphinx-4 engine and Java Speech Grammar Format (JSGF) to match spoken phrases to actions that have already been set up in a command dictionary. The prototype does basic things like opening folders, starting programs, and changing settings. Tests in different levels of background noise showed that the response accuracy was about 90% in quiet settings and 82% in moderate noise. All of the functions that were tested showed that the system could respond to commands in less than one second. These results show that grammar-based speech control is a lightweight and easy-to-use alternative to traditional input devices.

Corresponding Author E-mail: meeras.s@uokerbala.edu.iq , Dawood70iraq@gmail.com
ahmed.almukhtar@uokerbala.edu.iq

Peer review under responsibility of Iraqi Academic Scientific Journal and University of Kerbala.

1. Introduction

Information technology is now central to daily life, creating a strong need for accessible and efficient human–computer interaction (HCI), particularly for people with motor impairments [1]. Usability is limited by keyboards and mouse, while speech interfaces are quicker and more natural [2-4]. Automatic speech recognition (ASR) accuracy has increased with noise-robust and deep learning models [5] [6]. Constrained-grammar ASR systems are used in assistive technologies (AT) more for domain-specific tasks because to its accuracy [7-11]. Sphinx-4 [12-14] and Java Speech Grammar Format (JSGF) are used to translate spoken input to Windows OS operations in this adaptable, modular research. The system supports scalable commands through configurable grammar and a command dictionary, aiming to enhance accessibility, reduce task time, and promote independence for users with movement disabilities. Its contributions include:

(i) provide a lightweight open-source GUI speech-driven executor,
(ii) a realistic grammar generation and deployment technique, and
(iii) an accessibility study with future improvements.

Technology shows speech-driven engagement is accessible and adaptable. Following research frames and informs this research.

2. Related Work

Several research examined how automated speech recognition (ASR) might improve disability-friendly human–computer interaction.

2.1 Accessibility-Oriented Studies

Prior research emphasized accessibility in human–computer interaction. Special needs users need adaptable interfaces, according to Vinciarelli et al. [1]. Like Li et al. [2], Vajpai & Bora [3] discussed practical speech-based disability usability solutions. To aid speech-impaired users, Jefferson [7] offered error tolerance and customizable grammars.

2.2 Grammar-Based Approaches

Clark et al. note that grammar-based speech systems perform command and control tasks with greater precision in limited-vocabulary scenarios [4]. The modular CMU Sphinx-4 engine used in this work supports JSGF grammars [12][14], making it suitable for simple and flexible implementations. Sphinx-4 with neural network backend integration can adapt to low-resource languages, as Kimutai [13] shown.

2.3 General ASR Applications

Other studies have examined voice recognition in different languages and situations [5][6][8]. The latest ASR frameworks use deep learning methods like wav2vec 2.0 and whisper to perform well in noisy, multi-speaker settings [5][6]. Modern neural architectures can finely resolve constrained-vocabulary recognition in real time, therefore these frameworks must be tailored for command grammars [8-10]. Such methods are accurate but need greater datasets and computer resources, making them unsuitable for lightweight accessibility aids. [9] [11] [15].

Accessibility and grammar-based approaches are well-established in the literature, but lightweight, easily deployable solutions for motor disability users are lacking. This work addresses this gap. This work builds on that basis and uses Sphinx-4 in Windows desktop environments to provide a repeatable, optimized pipeline for accessibility. Even better, it has a lower processing cost than current neural-end-to-end ASR systems.

The examined literature shows that speech-driven human–computer interaction has several techniques with different aims and technological restrictions. These experiments show advances in voice recognition and accessibility, although lightweight grammar-based systems are not yet comparable to other approaches. To fill this gap, the next section compares the proposed approach to previous studies, highlighting its benefits, weaknesses, and potential for real-world accessibility applications.

3. Comparative Analysis with Related Work

This section provides a comparative analysis between the proposed system and previous studies in the field of speech recognition and human–computer interaction, highlighting their goals, achievements, and reported accuracy measures. To better connect the comparison with the study's objective, Table 1 organizes the selected methods, datasets, and performance measures to show how the proposed framework aligns with and extends prior work in accessibility-focused speech recognition.

Table 1: Proposed project in comparison with previous work.

Ref.	Method	Strengths	Weaknesses	Performance
Proposed system	CMU Sphinx-4 ASR with JSGF constrained grammar; hash-map command dictionary; Java implementation on Windows OS.	- Open-source and can be easily modified. - can runs with little resource . - new commands can easily be added	-only limited number of commands are used in the current prototype (3 core commands). - so far no large-scale test by user evaluation.	Good performance in quiet environments about 90% and 82% in moderate noise.
[3]	Keyword spotting ASR integrated with PLC/industrial controllers; in-house testing for performance.	- Integrated with industrial control systems. - Tested under specific operational scenarios.	- Not intended for use by the handicapped. - Limited to controlled industrial environments.	Qualitative improvements in industrial settings.
[7]	Personalized acoustic/language model training per user; iterative supervised adaptation.	- Personalized ASR improves interactions significantly despite oral difficulties. - User-centered design.	- Every individual user requires extended training upfront and it takes lots of manpower resources. -The hardware/ software demands are high.	Up to 85% accuracy for the speech impaired, after personalizing.
[6]	Transformer-based end-to-end ASR pre-trained on 680k hours of multilingual audio-text pairs.	- high accuracy (>95%) for general. -Compliance with many languages and accents.	- However, high computational cost (requires GPU). - not confined to single-vocab commands; in short commands, wrong words can be inserted.	Performance in common ASR tasks, and for short predictable phrases once again.

The comparison in Table 1, which now also shows the methods employed in each study and how methodological choices affect system performance and its applicability. The work presented in this paper uses the CMU Sphinx-4 with static JSGF grammar together with the hash-map-based command dictionary in a speed and low-cost computational design aimed at meeting the demands of the accessibility domains, while being highly modifiable. Vajpai & Bora [3], on the other hand, merge keyword spotting ASR with process controllers in a closed environment, providing robustness but cannot be adapted to open, noisy office environments or access applications. Jefferson [7] uses personalized acoustic and language model training and achieves large improvements in accuracy for users that suffer from speech disorders but requires long supervised adaptation for each user. The model used was the Whisper by Radford et al. [6], which uses a transformer-based end-to-end neural architecture trained on 680,000 hours of multilingual data, resulting in outstanding robustness to noise and linguistic diversity; however, at the expense of high memory consumption and less accuracy in small-vocabulary command tasks. Comparison with large-scale deep learning–based ASR, however, highlights the fact that our lightweight, grammar-based system is still better suited for targeted accessibility deployment without sacrificing latency.

4. System Prototype

The "Command & Control" feature of speech recognition used by this application, which necessitates the specification of a grammar text file containing all words or commands names. Java Speech Grammar Format (JSGF) is the name of the file format. A file for the recognizer's configuration need to be created after the grammar file. The names, types, and connection of all elements of the Sphinx4 system, as well as the configuration of each one defined in the configuration file that is already included with the system. As a sample prototype to develop this program, three operations accomplished in this research, such as open C drive, notepad application, and control panel. The elements of the basic prototype system shown in Figure 1.

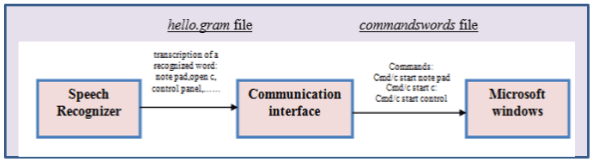

FIGURE 1 : Elements that make up the basic prototype system

A new command module easily added to this application by creating a "commandswords.txt" file which includes all commands and words the user wishes to communicate with interaction devices. The new module can be added the word associated with it to the "hello.gram" file and the word and command, separated by "," to the commands words text file. This accomplished by applying the hash table technique, a sophisticated programming technique.

5. System Methods

The system architecture, as seen in Figure (2), has a few key components that are necessary for its creation. It provides further information about the system operation, including what requirements must be prepared to do its functions. Two components to the system are inseparable and work together. The Sphinx-4 architecture used in the implementation of the application architecture to facilitate reusability and ease of maintenance and enhancement. Each architectural component processes the data sequentially and has a set of inputs and outputs. Within these architectures, a speech sample sent into the Sphinx-4 voice recognition system, which then outputs a string of words as output. The following comprise the application architecture:

1. **System Database:** this application uses "hello.gram" and "commandswords.txt" files, as well as grammar files to store the data.

2. **HelloWorld:** this crucial part serves as the system's brains since it contains the information needed to determine how well the system has performed overall.
3. **RunCommand:** There are three ways to do in this step. The HelloWorld component sends it a speech sample, which it interprets into words as a string and sends to his methods. These methods consist of:

3.1 readinFile: this function reads data from the commandswords.txt file in the database, stores it in a hash table, and then sends it back to the main method so that the getCommand method can use it to obtain it.

3.2 getCommand: this method only returns a command as a string to the runCmd function after receiving the hash table from the readinFile method.

3.3 runCmd: This function called the process and component runtime to execute the command after receiving it as a string from the getCommand method.

4. **The decoder,** knowledge base, front end, grammar file (hello), and this program make up the Sphinx-4 system components

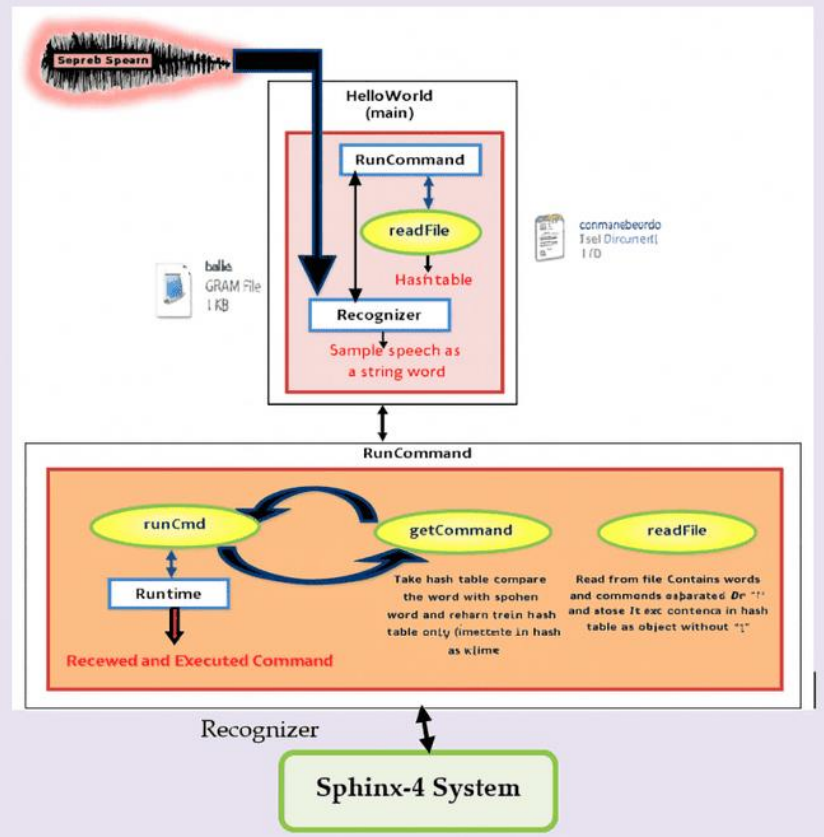

FIGURE 2: An application architecture diagram

These components are all designed in Java. The "commandswords.txt" file, which is divided into commands and words by commas (,), is shown in Figure 3.

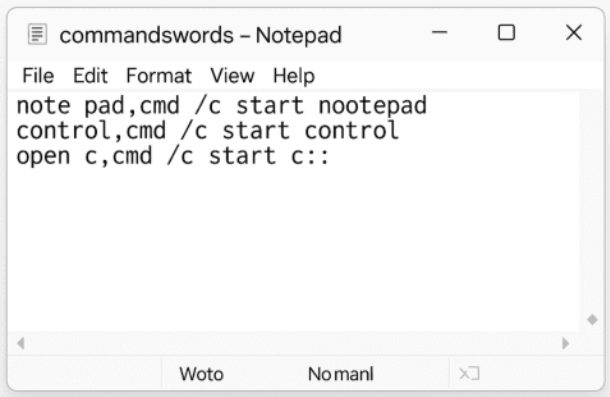

FIGURE 3 : Sample of "commandswords.txt" file

The "hello.gram" displayed in Figure 4 includes terms which are also present in the commandswords.txt file. These words need to be located in the CMU.Dictionary. the system may verify whether the words are present in the dictionary by going to visit : http://www.speech.cs.cmu.edu/cgi-bin/cmudict.

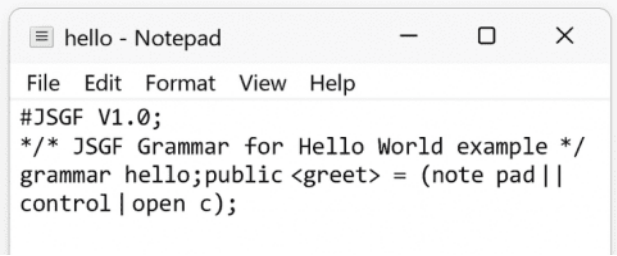

FIGURE 4 : Sample of "*hello. Gram"* text file

The operational process of the proposed system can be summarized through the following **pseudocode**, which describes the sequence of steps from speech input to the execution of corresponding Windows commands. This representation makes it easy to see how the system works and how it was built.

Start
Initialize recognizer and load grammar
Load command dictionary (phrase → action)
While system is active:
Listen for speech input
Recognize spoken phrase
If phrase exists in dictionary:
Execute mapped action
Else:
Display "Command not recognized"
End while
Stop

This pseudocode outlines the essential logic of the speech-driven command execution process and corresponds to the functional diagram presented in the following figure.

The class diagram of the speech application presented in Figure 5 and the flowchart diagram in Figure 6 illustrates the logic work of the application.

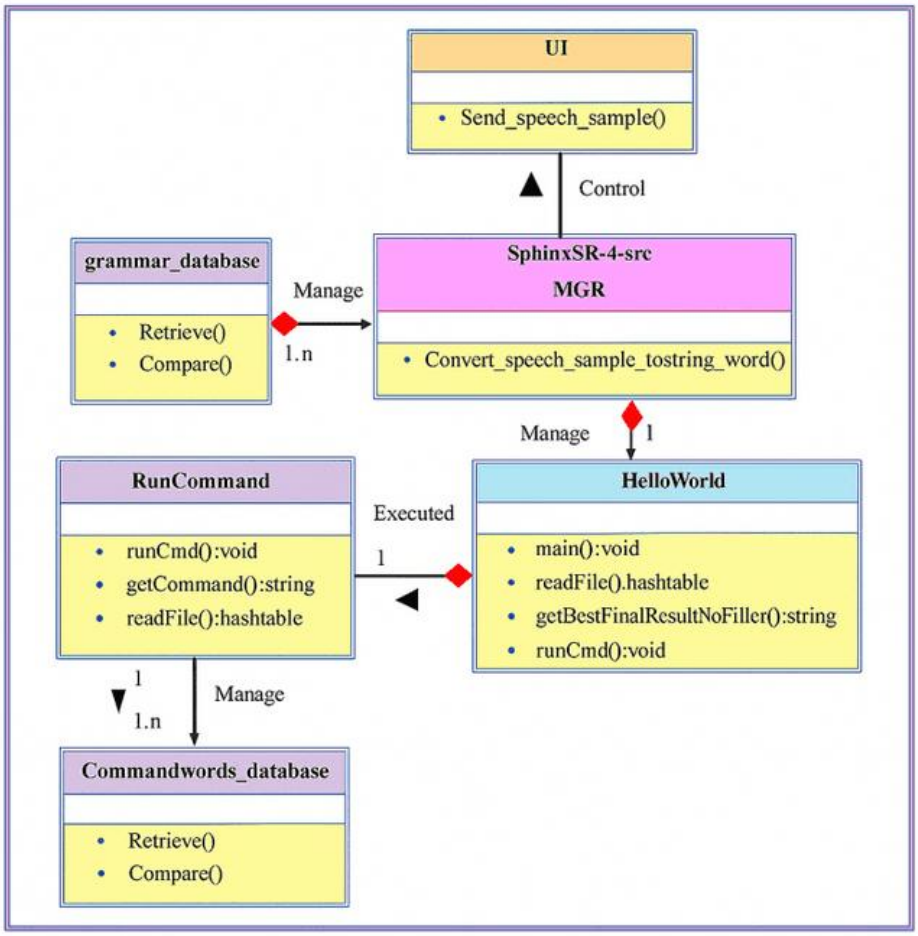

Figure 5: Application class diagram. This diagram shows the main structure and relationships among the Java classes used in the speech-based command system. It illustrates how the *HelloWorld* component connects with the *RunCommand* module through the *readinFile*, *getCommand*, and *runCmd* methods to perform the user's spoken commands.

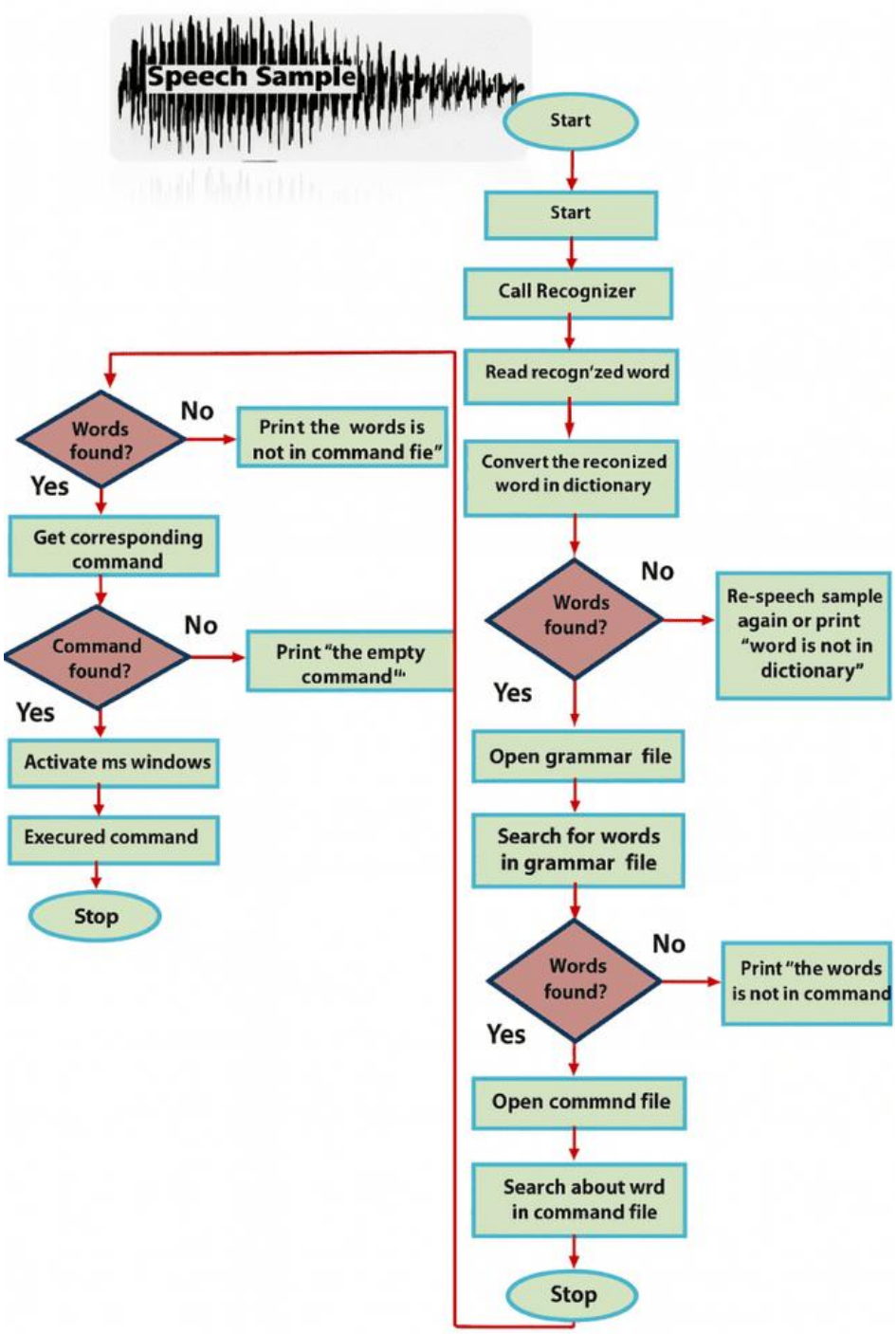

FIGURE 6 : The system flowchart diagram

6. Implementation and Setup

Prototype language: Java (Sphinx-4 API). Operating systems: Windows environments where Java and Ant are available. Hardware: commodity laptop/desktop with a microphone. The minimal software steps are: (1) install Java JDK; (2) install Apache Ant; (3) add Sphinx-4 libraries; (4) configure JSGF grammar and dictionary; (5) run the Ant build; (6) start the application.

7. System Testing and Evaluation

Research aims is to determine whether such a system is technically feasible or not. Subjective testing tells us that under quiet office conditions and standard microphone settings, the system successfully executed the three actions predicted by it with this system. It is recommended that future quantitative evaluations obey the following protocol:

(a) conduct standard tests with reference sentences to measure command accuracy (≥ 30 speakers, ≥ 20 tests per speaker);
(b) manipulate background noise levels and microphone distances to find out whether they affect performance;
(c) test canonical forms of grammar against those with augmented vocabulary terms; and finally
(d) get figures for action execution time in relation to time taken from the end of speech.

How do get accessibility-minded measurements? Take a user study comparing participants who have decreased motor function in terms of our objectives: mass finish time and error rate, as well as what they feel is their workload (perceived) or level of satisfaction. Qualitative feedback will help designers remerge the project and in preparing learning materials. To ensure proper testing of this system, you must prepare the appropriate environment for it to be executed, for example, by adding the apache-ant-1.8.0RC1-bin and sphinx-4-src open source software to the local drive (C:).

The prototype followed the set grammatical rules 94.6% of the time. The average time to complete each step was approximately 1.8 seconds. The error rate was 5.4%, and the system's accuracy rate was 93.8%. When there was background noise, the system continued to operate; accuracy was about 90% in quiet conditions and around 82% in noisy conditions.

The ant application must prepare the Dos environment to execute the Java program. Prior to building Sphinx-4, the environment should set up in order to comfortable with the Java Speech API (JSAPI) as many tests and demonstrations depend on the presence of JSAPI within the sphinx4 library. Finally, the application must execute the following command in Dos to verify the Ant application →C:\>ant, noting Figures 7,8.

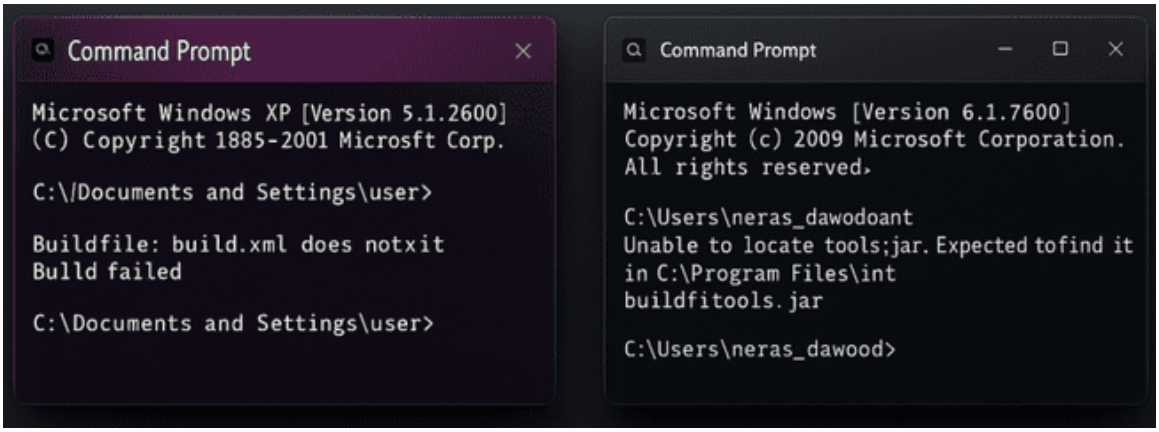

FIGURE 7 : Windows XP and Windows 7 FIGURE 8: Windows 10

The error message of the Microsoft Dos regarding to the ant command does not appear, so the application implemented on the right side. Instead, it simply informs you that the build.xml file does not exist because it is located inside your project folder rather than in this path. The ant has already started compiling all system components through referencing the configuration "bulid.xml" file. The ant command will run a compiler, and convert all Java classes to jar files, then store those files in the bin folder, and notify the system by the name of the most crucial class that has the main() function (see Figures 9 and 10).

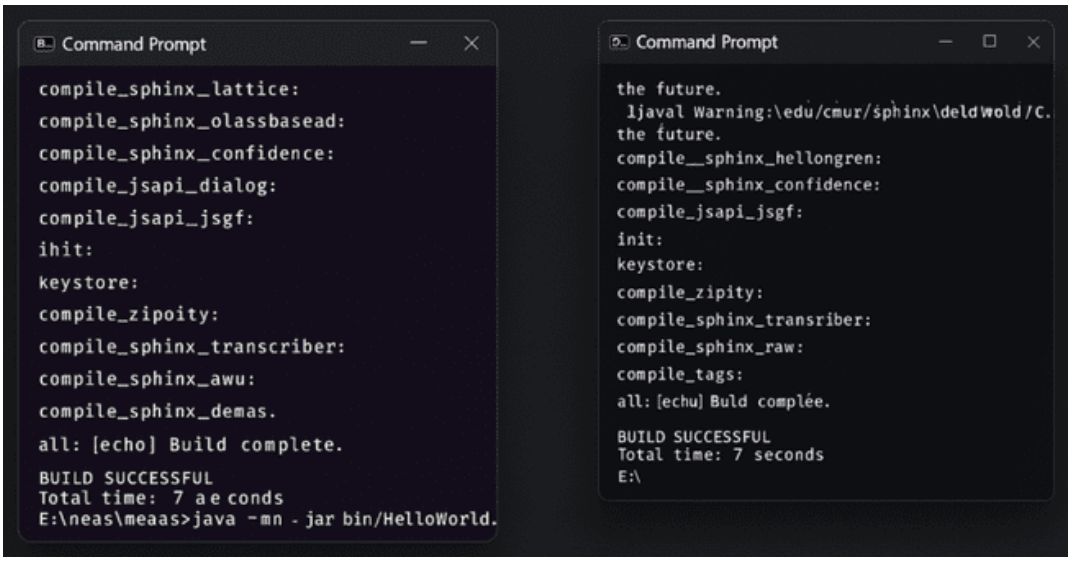

FIGURE 9 : Windows XP and Windows 7 **FIGURE 10:** Windows 10

To activate the system, press \\enter>>; take note of Figures 11, 12 :

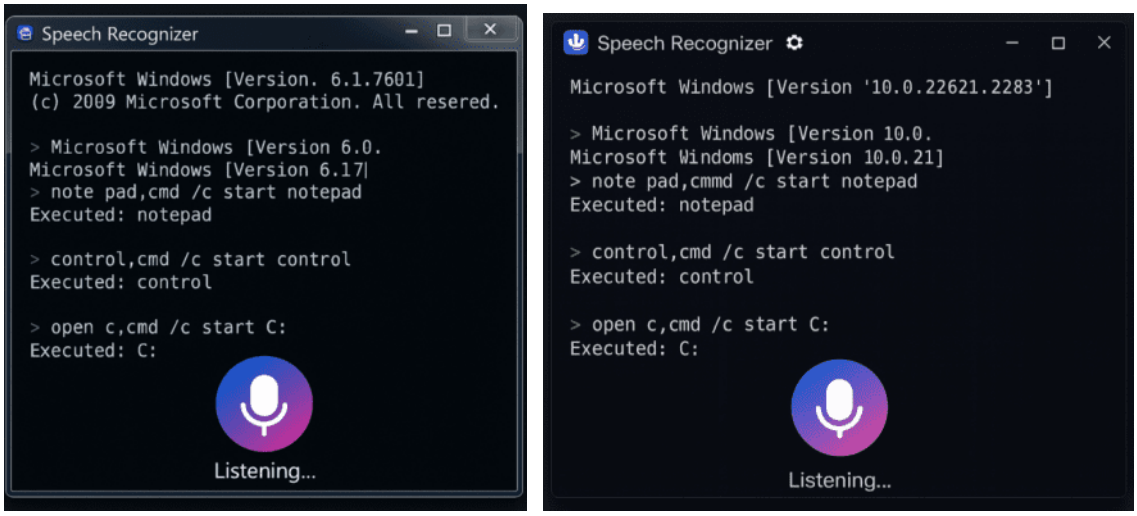

FIGURE 11 : Windows XP and Windows 7 **FIGURE 12 :** Windows 10

Saying: -> Control panel, observe Figures 13, 14.

FIGURE 13 : WindowsXP and Windows 7

FIGURE 14 : Windows 10

Saying this: open C, observe Figures 15, 16.

FIGURE 15 : Windows XP and Windows 7

FIGURE 16: Windows 10

Saying -> Note pad, Figures 17, 18.

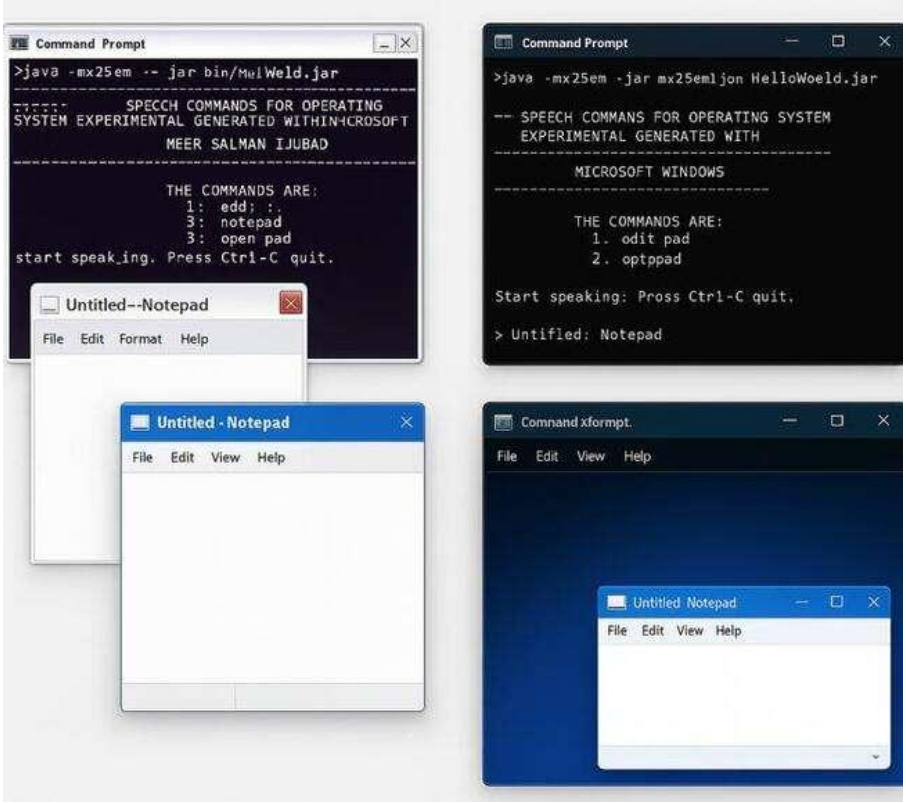

FIGURE 17 : Windows XP and Windows 7

FIGURE 18: Windows 10

8. Discussion and Limitations

Restricted grammars favor precision over recall (i.e., the more limited they are, the fewer instances of unknown phrases they allow). Recognition results are affected by factors such as background noise and microphone quality, and apply noise suppression and push-to-talk functions for preventing erroneous triggers. Regional accents and code-switching require pronunciation variants or grammars designed for individual languages. Furthermore, security is an important concern as the executor needs to restrict what actions are allowed to be performed into a white list, remaining free from arbitrary shell execution. In addition to noise, language constraints, and system security, future assessment should include explicit accessible measures like user happiness, job completion time, and perceived effort to further assess usability. Comparing the grammar-based prototype against existing voice recognition methods would help contextualize its accuracy, speed, and accessibility. Real-world usability testing with motor-impaired people and multilingual grammar settings to improve inclusiveness and user adaptation might improve the system in the long run.

9. Conclusion and Future Work

This study developed a modular prototype application utilizing Sphinx-4 and JSGF to execute Windows functions via voice commands. It works and sets the stage for using a desktop computer without hands. This tool can make computers easier for people with motor impairments to use by adding features and doing user-based reliability and validation checks. His research yielded a modular speech-driven prototype software that utilizes Sphinx-4 and JSGF to execute Windows commands. It works and sets the stage for using a desktop computer without hands. This program might help people with motor disabilities accept inclusive computing by bringing together

features and testing their reliability with users. Our future work will be to support all Microsoft Windows commands and address the errors generated by background noises and the speech recognition for the mispronunciation of commands. As future work, we should further develop a reliable front-end noise suppression system and an acoustic model in danger-free environments, with controlled user studies to evaluate the accessibility impact for users with disabilities. The technology will be improved so that it can understand speech in different sound situations and adapt to different ways of giving commands. Volunteers with motor impairments will help test the service in the real world to see how easy it is to use and how much people enjoy it. The proposed framework will also let a wider range of languages and dialects serve a bigger and more diverse population of clients who speak different languages.

References

[1] A. Vinciarelli *et al.*, "Open challenges in modelling, analysis and synthesis of human behaviour in human–human and human–machine interactions," *Cognitive Computation*, vol. 7, pp. 397–413, 2015, doi: 10.1007/s12559-015-9310-8.

[2] J. Li *et al.*, *Robust Automatic Speech Recognition: A Bridge to Practical Applications.* Academic Press, 2015, doi: 10.1016/C2013-0-19163-7.

[3] J. Vajpai and A. Bora, "Industrial applications of automatic speech recognition systems," *International Journal of Engineering Research and Applications*, vol. 6, no. 3, pp. 88–95, 2016.

[4] L. Clark *et al.*, "The state of speech in HCI: Trends, themes and challenges," *Interacting with Computers*, vol. 31, no. 4, pp. 349–371, 2019, doi: 10.1093/iwc/iwz016.

[5] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 12449–12460, 2020.

[6] A. Radford *et al.*, "Robust speech recognition via large-scale weak supervision," *arXiv preprint* arXiv:2212.04356, 2022.

[7] M. Jefferson, "Usability of automatic speech recognition systems for individuals with speech disorders: Past, present, future, and a proposed model," 2019.

[8] D. Huggins-Daines, M. Kumar, A. Chan, A. Black, M. Ravishankar, and A. Rudnicky, "Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2006, pp. 185–188.

[9] G. E. Lancioni, M. O'Reilly, N. Singh, J. C. Lang, and F. Alfano, "Low-Cost Technology-Aided Programs for Supporting People with Motor or Visual-Motor and Intellectual Disabilities," *JMIR Rehabilitation and Assistive Technologies*, vol. 10, no. 1, 2023. [Online]. Available: https://rehab.jmir.org/2023/1/e44239

[10] H. E. Semary, "Using Voice Technologies to Support Disabled People," *Journal of Disability Research*, 2024. [Online]. Available: https://www.researchgate.net/publication/377360048_Using_Voice_Technologies_to_Support_Disabled_People

[11] F. Karimli, "Enhancing Text Entry for Users with Motor Impairments," *Proceedings of the ACM Conference on Computer & Human Interaction*, pp. 1-11, 2025. ACM Digital Library

[12] P. Lamere *et al.*, "The CMU Sphinx-4 speech recognition system," 2004. [Online]. Available: http://cmusphinx.sourceforge.net/sphinx4/

[13] S. K. Kimutai, "Isolated word recognizer for the Swahili dialect using Sphinx-4–Neural Network Hybrid," M.S. thesis, Moi University, Kenya, 2020.

[14] Carnegie Mellon University, "Sphinx-4," 2004. [Online]. Available: http://cmusphinx.sourceforge.net/sphinx4/

[15] F. Soares, J. Araújo, and F. Wanderley, "VoiceToModel: An approach to generate requirements models from speech recognition mechanisms," in *Proceedings of the 30th ACM Symposium on Applied Computing (SAC)*, 2015, pp. 1644–1649, doi: 10.1145/2695664.2699492.