Assisted Technology Keyboard

Mayank Mayank

mma266@sfu.ca

Abstract. Our Assisted Technology Keyboard (ATK) is designed to assist users who face challenges with communication, specifically those who struggle with precise motor control. The keyboard magnifies grouped letters, provides text-to-speech for letters and full sentences, and offers auto-correct, auto-complete, and word suggestions based on context. By combining simplicity and flexibility, ATK addresses key limitations in traditional on-screen keyboards, thereby improving usability, accuracy, and overall communication speed.

Keywords: accessibility · assistive technology · text-to-speech · auto-complete · auto-correct · keyboard

1 Introduction

Application & Problem: The application developed is an assistive technology keyboard designed to support individuals who are non-verbal and experience tremors or reduced motor precision when using standard input devices. Patients who struggle with these difficulties are commonly observed to have traumatic brain injuries (TBI), who experience motor impairments that limit their ability to speak and communicate when using a standard keyboard, whether physical or on-screen. In addition to impaired motor skills, these injuries include slowed processing speed, impaired attention, or reduced cognitive efficiency. To address these challenges, users require assistive technology that helps them communicate effectively while recovering and accommodating for their injuries. This project is focused on enhancing core aspects of the keyboard, specifically usability, practicality, and effectiveness, to better accommodate the needs of these individuals and improve their ability to communicate.

Motivation & Background: The primary motivation is to provide a simple yet effective interface that users can rely on to communicate quickly and accurately. Existing literature in human-computer interaction (HCI) and assistive technologies emphasizes the importance of thoughtful design and predictive text to reduce keystroke count. Tools like SuperKeys served as initial inspiration.

Report Outline:

– **Section 2** describes the materials and technologies we used.

- **Section 3** details our methods for auto-correct, text-to-speech, and handling user jitter.
- **Section 4** highlights our results, both qualitative and quantitative.
- **Section 5** covers our accomplishments and lessons learned.
- **Section 6** details each team member's contributions.
- **Sections 7 and 8** conclude the paper with discussions and proposed future work.
- **Acknowledgements** and **Appendix** follow, and **References** are listed at the end.

2 Materials

To create our keyboard, we utilized a full-stack web application using React.js as the primary frontend, Flask as the primary backend, and various RESTful APIs. We chose React.js as the frontend because our application is relatively simple, single-paged, and we want to ship features as fast as possible. React checked all those boxes, and as a bonus it also has seamless compatibility with Flask. Consistent JSON formatting, ease of development, and API-driven architecture are all benefits of using React.js in combination with Flask.

We chose Flask as our primary backend because we specifically needed a backend architecture that utilizes Python. This is because we utilize various APIs that are Python specific - Datamuse, pyttsx3, etc. Therefore, we wanted to prioritize the backend and choose technologies that revolve around it. There are several features missing in our application that would typically be in a web application, such as a database, security, data layer etc. For the purposes of this application, we concluded that this was unnecessary and out of scope. For example, a database is not mandatory for the key features of our application, however, it could potentially be used in future iterations for user enhancement (e.g. a feature to save user preferences).

There are several features of our keyboard that accomplish the goal of accessibility and usability for our target audience:

1. **Large, space efficient keyboard.** We utilized React.js, HTML, and CSS to create a responsive keyboard with keys taking up a maximal area of the screen. With our colour coding, users can click into the coloured areas and get a magnified view of the keys, ensuring that the keys would be easy to press. It uses the common QWERTY keyboard layout, so people can use muscle memory from keyboards they may have used in the past.

2. **Hovering over keys.** If the user has difficulty pressing buttons, our keyboard has functionality that allows zero mouse clicking. Users can hover over any button or key for three seconds to automatically press down and activate the key.

3. **Auto-correct.** Powered by Google Gemini Flash 2.0, if a word is misspelled when typed, an auto-correct feature will activate to suggest potential words that may represent what the user is trying to type. This allows users to save time and effort when accidentally making small mistakes due to their tremors.

4. **Auto-complete.** Powered by difflib and a large local library of words, users can get suggestions to complete their current word, saving time, as well as increasing accuracy.

5. **Suggested words** Powered by Imagineville API and Datamuse API, users are suggested words that may come next based on their current sentence, making the process of communicating faster and increasing the accuracy of completing a sentence.

6. **Spoken words and characters.** Whenever a user types a letter, the letter is played out via speakers, which allows for instant feedback on people who may be deaf or hard of hearing. There is additional functionality for the user to playback all the current sentence they have typed so far, which further enhances the usability and practicality of the keyboard by allowing users to play their message aloud for others to interact with them. This is powered by pyttsx3 library, which automatically converts text to a .wav file, which can then be played on the frontend.

7. **Various quality of life features.** There are various quality of life features that are present on our keyboard, such as custom backspace, delete all, and undo buttons, all specifically designed for comfort, with consideration to our target audience. These features were created using React.js.

When designing this keyboard, we utilized www.cricksoft.com/us/superkeys as an inspiration. Therefore, all our design choices revolved around enhancing the existing keyboard, and adding additional functionality specific to our target audience.

3 Methods

For the auto-correct feature, we utilized Google Gemini Flash 2.0 to create a prompt for the closest word. This is done through the Gemini API in our backend. The prompt we sent to the Gemini API was "*Suggest the most likely correct word if the word is misspelled: 'query'. In your response, please only include the auto-corrected word and nothing else. If the word is already spelled correctly, or if you cannot match the word with any English word, please respond with 'null'.*", where 'query' is the word to apply auto-correct to. This ensures that the only response from the API call is the autocorrected word, without any extra fluff. When the user presses space bar, and the word is spelled incorrectly, the frontend displays a prompt asking if the user wants to correct the word or not. If the response is ”null” (in the case the word is already correct), then the prompt does not show.

For the auto-complete feature, we utilized **Datamuse**, a Python API that allows predictive word-finding. We sent a request to the Datamuse API, which contains a query, and returns five corresponding word suggestions. We utilize the ”rel_bga” query, which stands for ”words that are frequently followed by lastWord”. Afterwards, we return a JSON structure with the five most relevant suggestions, which is then displayed on the frontend.

For the sounded-out words, we utilize the **pyttsx3** library to convert words into .wav files. Every time the user completes a word, the backend API is called, which converts the word into a .wav file. Afterwards, it is saved, and then fetched by the frontend by returning the file name. These words are then played, which returns the sounded out word as audio to the user.

To measure the accuracy of our application, we created a custom JavaScript function to simulate jitter. This custom jitter allows us to quantify the results of the keyboard, and to see if it truly works as intended. With the jitter on, we can plot a graph of average number of correct letters per x seconds vs. the magnitude of tremor. Ideally, as the tremor increases, we would want the average number of correct letters to be relatively the same to account for people with more severe shakiness. Therefore, this means we want:

$$m >= 0 \tag{1}$$

where m is the slope in a linear function. We can test all the features of this application, including manually clicking, hovering, etc. to prove our keyboard is accurate.

To predict the next words suggested to the user, we used two API’s to increase robustness and variety of the suggestions. Two words were selected from Datamuse based on related words that frequently follow the last word in the user’s input. For example, if the input ends with ”wreak,” Datamuse might suggest ”havoc” as a likely next word. To choose the remaining three words we used Imagineville API to predict words based on the context of the user’s input. The three most probable words based on the context were chosen. The combination of these two API’s ensured that the user is suggested the most appropriate and relevant words based on their inputs.

4 Results

4.1 Qualitative Feedback

Beyond the numbers, we gathered some feedback from potential users and volunteer testers:

- **Ease of Use:** Most testers found the enlarged keys and color-coded layout intuitive and reassuring.
- **Hover Mode:** Users with simulated tremors appreciated being able to select keys without pressing down, noting it reduced accidental presses.
- **Auto-Correct & Word Prediction:** Participants reported fewer spelling mistakes and faster overall communication, indicating these features effectively reduced keystrokes.

Another thing that we improved upon after feedback from the professor was to make the buttons and font larger and more readable in consideration of our users with vision and accuracy problems.

4.2 Unexpected or Negative Results

Below listed are some of the negative and unexpected results:

- We noticed that our keyboard accuracy dropped to 0% as soon as we increased the 'Jitter Magnitude' to more than 60, although we do not anticipate this to be an issue for majority of our potential user base, we aim to make this functional for absolutely everyone.

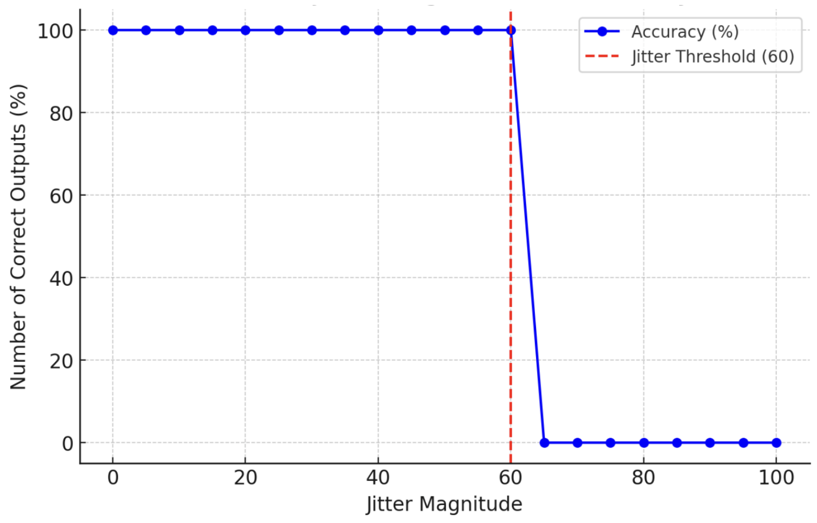

Fig. 1. jitter vs correct outputs

5 Accomplishments

Throughout the development of this project, we all learned numerous things not only in software development, but also in user-centered design, as well as thinking carefully about accessibility and varying user needs. We learned how to build APIs integrated with AI prompt engineering, how to create great web experiences for audiences with unique needs, and also how to work effectively in a software project environment.

We encountered numerous challenges during this project, though we could not overcome them all, we learned from each obstacle. Extracting requirements, creating tasks, and delegating them evenly among the team is a hurdle we faced early on in the project. We did not know how to transfer a two sentence prompt into an entire project, but we brainstormed a few ideas and were able to get a list of features to implement. This set us up for a good time trajectory and allowed us to communicate and work together more efficiently during the project. Another challenge that came up was how to build an application that is tailored towards people with tremors. Building an engaging app for healthy users is already difficult, so making one for a special audience makes it even tougher. However, we researched into the effects of tremors and did our best to put ourselves in the shoes of our target users, which helped us gain an understanding of what we should build and how we should build it.

One of the challenges we faced while improving user experience was increasing words per minute for users to communicate efficiently. Our team realized that it would require a lot of effort to input each letter and would drastically increase the time it takes to complete multiple words while struggling with tremors. During one of our meetings, we brainstormed and overcame our obstacle by coming up with solutions such as autocorrect, autocomplete and suggested words. We were able to evenly distribute the workload and implement these functions.

Another obstacle we faced as a team was finding additional functionalities to implement in our application that were feasible, achievable, and practical. One of the ideas being personalized machine learning that would learn the speech patterns and word choices and applying them to our autocorrect, autocomplete and suggested word functions. This idea was too grand and was not achievable in our limited time, as we would have to learn how to create a dataset, create a model, and train it. Another idea we thought about later was eye tracking as an alternative input using an eye-gaze assistive tool. This implementation required refactoring the code and implementing an eye-gaze assistive tool with an appropriate threshold to input letters. We had to move past this idea as it was somewhat beyond our reach along with the tight deadline. As a team, we were able to overcome this obstacle by brainstorming and implementing features that were within scope and could be realistically added with the remaining time.

6 Conclusion and Discussions

The development of the assisted technology keyboard successfully addresses many challenges faced by users who are non-verbal and have motor control issues, such as tremors. By prioritizing user-friendly design choices (including enlarged keys, hover selection, and immediate audio feedback), we demonstrated that thoughtful interface design can significantly improve typing accuracy and speed for this demographic.

One central discussion point is the balance between technology complexity and end-user needs. While advanced AI-driven language models and predictive features can enhance communication speed, maintaining a simple, reliable user interface remains crucial. Our experiences underline that robust user testing and iterative design are necessary to refine any accessibility tool.

In conclusion, this project demonstrates that strategic use of technology can help close the communication gap for individuals with limited motor precision. By building upon these foundational results, future work can scale and refine the keyboard to broader user needs, ultimately driving more inclusive and effective digital communication solutions.

7 Future Work

The current version of the assisted technology keyboard achieves our primary goals of improving usability and effectiveness for users that are non-verbal and suffer from tremors, but there are some areas within the system that could be further improved and implemented. An enhancement that can be implemented is an adaptive word prediction model that learns from the user's input history. This will create more personalized predictions that is tailored to the user's speech patterns, which can lead to suggesting complete sentences for the user.

Another function that our team wanted to implement was an alternative input method for selecting letters. This could have been achieved through joystick controller or touchscreen, but we mainly wanted to implement eye-tracking as a method of interaction with the keyboard. This would allow users that suffer from more severe motor impairments to use our assisted keyboard through eye-gaze, which would click or hover keys on the keyboard based on the place the user is looking.

Additionally, the keyboard's tremor test could be improved by allowing the user to click on command, opposed to automatically clicking every two seconds. This will allow more accurate testing of our keyboard as it would more closely emulate our target audience.

Lastly, an option for complete customization options for keyboard layout, key sizes and colours to tailor the user's needs. This would open up use of the keyboard to those suffering from additional impairments such as colour blindness. Alongside the customization, a bar that adjusts the error threshold when differentiating clicks that are accidental versus on purpose can be implemented.

Acknowledgements

When brainstorming for additional features we found some possible technologies to add from a Github repository by willwade. The repository is a curated list of opensource assistive technologies that can be implemented to create an assisted product. API's for Word Prediction were found here and used to implement suggested words for the user.

Appendix

The source code for our project can be found in our GitHub repository (link). As noted in the README.md file, follow these steps to run the project locally.

Get the project on your machine

1. Clone the repository:
   `git clone https://github.com/sfu-cmpt340/2025_1_project_09.git`
2. Change into the project directory: `cd 2025_1_project_09`

Backend

1. Setup and run the API server (from the root project directory):

```
cd server
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
export FLASK_APP=app.py
python -m flask run
```

2. If using PowerShell, set the Flask app like so: `$env:FLASK_APP = "app"`
3. The API will be available at http://127.0.0.1:5000

Frontend

1. Setup and run the web app (from the root project directory):

```
cd client
npm install
npm start
```

2. View the application at: http://localhost:3000/

References

1. L. S. Turkstra et al. Cognitive-communication disorders in adults with traumatic brain injury. *PMC*, 2016. Available at: `https://pmc.ncbi.nlm.nih.gov/articles/PMC4791928/`, (Accessed: 2025-03-21).
2. M. A. Struchen. Social communication and traumatic brain injury (tbi). Available at: `https://biausa.org/wp-content/uploads/Social-Communication-and-Traumatic-Brain-Injury-Professionals.pdf`, (Accessed: 2025-03-21).

[1] [2]