Voice Controlled Chrome Navigator

Using large language models

Arman Keshavarz
M.S. Computer Science, NYU
251 Mercer street, New York City, New York
ak12016@nyu.edu

Abstract— **The goal of this project was to be able to create a (semi) hands free tool that would be able to handle and execute voice inputted chrome commands from a local server using a large language model, and rpc (remote procedure call) commands. We were successfully able to build this tool that could handle voice commands one at a time and execute them almost flawlessly. This was created with the intention to be used by individuals who struggle with typing or have impaired motor functionalities.**

Introduction

Voice-controlled web browsing offers a faster, more natural, and more accessible way to navigate the internet, but most current solutions depend heavily on cloud-based speech recognition and NLP. Chrome Navigator addresses these problems by using locally-deployed machine learning models to understand and execute voice commands. The dataset was built by pairing natural language commands with their corresponding RPC calls and then expanding them using the OpenAI API to cover a wide range of phrasing, tones, and contexts. In this process we trained both a custom built LLM and a fine-tuned transformer model to process these commands to perform actions such as navigation, tab management, form filling, and content interaction. Because all processing happens locally and actions are sent directly through Chrome's APIs, the system delivers real-time performance while protecting user privacy and eliminating the need for external services. This demonstrates the potential for lightweight, domain-specific language models to power intelligent browser automation in resource-constrained environments.

I. Literature Review

1. WebNav: An Intelligent Agent for Voice-Controlled Web Navigation (Srinivasan & Patapati, 2025)

WebNav introduces a sophisticated browser-based voice navigation agent that divides tasks into strategic planning, assistant, and inference modules—the last of which directly interacts with the page's Document Object Model (DOM). A novel dynamic labeling engine allows WebNav to generate real-time labels for interactive elements, enabling accurate voice command mapping. Preliminary evaluations show improvements in response time and task completion accuracy over traditional screen readers, highlighting the promise of structured architectures for assisting visually impaired users.

2. Toward Voice-Assisted Browsers (Amman et al., CUI '20)

This paper examines the embedding of voice assistants directly into a desktop browser, focusing on the Firefox Voice extension. Through a small think-aloud user study, the authors investigate user interactions, uncovering usability challenges and opportunities when voice controls are placed within the browser context. The usability findings are critical to understanding real-world implications of voice command interfaces in browsers. These insights can guide user-centered improvements and help anticipate design challenges for Chrome Navigator.

II. Datasets

The purpose of this dataset was to be able to encompass almost all commands that any chrome user would be able to do. I.E. The dataset ideally spans all functionalities that chrome offers while also putting an emphasis on common commands such as searching, clicking and scrolling. Putting every single text input that users would want to do into the dataset is impossible but the corresponding rpc commands are far less so focusing on increasing the breadth of the rpc commands was an important part of the dataset creation. I originally started by just typing up around 50 common commands that I thought would be vital to the functionality of the chrome navigator and then figuring out the correct rpc command for said text input. A good example is this: {"utterance": "scroll down by 300 pixels", "rpc": {"method": "scroll", "params": {"direction": "down", "amount": 300}}} The best way to look at the dataset is the first column label is the key "utterance" and column we are trying to predict is the second key "params." In order to expand the dataset I had already created, I wrote a Python script that takes each of my seed commands and automatically generates a set number of brand-new commands using the OpenAI API. The script reads each seed from my original jsonl file and sends it to the model along with a system prompt that forces the output to only use the allowed RPC methods—navigate, click, type, and search—while also making sure there's variety in tone, websites, queries, and typing fields. For each seed, it generates exactly the number of new commands I specify (in my case 20), parses them, removes any duplicates, and appends them to a new jsonl file. Essentially, this script multiplies my original dataset by a fixed factor, keeping the structure consistent while greatly increasing the diversity and coverage of possible Chrome voice commands.

III. Methodology

The system architecture involved building a Flask server to host the trained model locally. I created a simple web server that would receive natural language text through an HTTP endpoint and return the corresponding RPC command. This Flask server runs on a localhost port and handles the inference requests from the Chrome extension. The server loads the trained model and tokenizer, processes incoming text, and generates the appropriate RPC output.

For the Chrome extension, I built a complete browser extension with background scripts, content scripts, and a manifest file. The extension handles voice input through the Web Speech API, sends the transcript to the local Flask server, receives the RPC command, and then executes it using Chrome's native APIs. The background script manages the communication between the voice input, the Flask server, and the browser actions, while the content script handles the actual execution of commands within web pages. This architecture ensures that all voice processing happens locally, maintaining user privacy while providing real-time responsiveness for browser control.

Once the dataset was built (outlined in section II), it was on to training the model. For the model training, I converted the dataset into a format suitable for fine-tuning the t-5 small transformer model. The conversion process flattened the RPC structure into simple text strings, so instead of the complex JSON format, the model would learn to output something like "scroll direction=down amount=200". I then split the data into training and validation sets using a 90/10 split to ensure the model could generalize properly. The final system architecture involved deploying the trained model locally on a Flask server that the Chrome extension could communicate with, eliminating the need for cloud-based services and maintaining user privacy.

Along with the fine tuning of the t-5 small model I designed and trained an encoder-decoder style large language model. The main challenge with this was the fact that I had to train on my cpu so the model size had to be very small so I chose to end up with 2.5M parameters, a size of 256 for the hidden dimension and a 4 headed attention along with a very limited vocabulary of 4000 words.

IV. Experimental Results

The results for the custom LLM were very encouraging when you take into account the fact that it was an LLM trained solely on an old intel cpu with a very limited model size and vocabulary. When using this model, we were able to see that it performed very well on inputs that were in the training set but struggled with stuff outside of the training set. This was the first thing I tested out so just seeing it work let me know that when using a pretrained LLM that it would perform far better and increase usability.

The fine-tuning for the pretrained LLM took about 3 hours and resulted in a testing loss of 0.0081 and validation loss of 0.0067. In terms of actual usability it is pretty much flawless, it can handle almost any single command that chrome supports.

V. Conclusion

The Chrome Navigator project successfully demonstrates that it's possible to create a voice-controlled browser extension that can handle almost all common Chrome functionalities without relying on external cloud services. By creating a comprehensive dataset of natural language commands paired with their corresponding RPC calls, and then training a local machine learning model on this data, the system can accurately convert user voice input into executable browser actions. The parameter standardization and data augmentation techniques I used were crucial for expanding the dataset coverage, making it possible to handle various numeric inputs and command variations that would be impossible to manually specify.

VI. Future Work

In the future I would like to adapt the dataset to be able to handle a chain of command voice input. This will require a massive expansion in the dataset but if it is fine-tuned with a GPU then this is totally feasible. I would also like to incorporate the use of agentic AI in this project.

Acknowledgment

Harrison Gao, Rasmika Billa, and Dr. Anasse Bari were of significant help to this project.

References

[1] Shi, T., & Pavlick, E. (2020). World Knowledge for Reading Comprehension: WebNavQA. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020), pp. 9138–9144. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.acl-main.828

[2] AlShakhs, M., & Ahmad, W. (2018). Toward Voice-Assisted Browsers: Enhancing Accessibility Through Speech Interfaces. Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '18), pp. 453–455. Association for Computing Machinery. https://doi.org/10.1145/3234695.3241025