5-2023

Cross-platform Development of Wake-Up-Word

Christopher Ryan Woodle

Cross-platform development of Wake-Up-Word

by

Christopher Ryan Woodle

A thesis submitted to the College of Engineering and Science of
Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Engineering

Melbourne, Florida
May, 2023

We the undersigned committee hereby approve the attached thesis,
"Cross-platform development of Wake-Up-Word"
by
Christopher Ryan Woodle

Veton Z. Kepuska, Ph.D.
Associate Professor
Computer Engineering and Sciences
Major Advisor

Samuel P.Kozaitis, Ph.D.
Professor
Computer Engineering and Sciences

Marius C. Silaghi, Ph.D.
Professor
Computer Engineering and Sciences

Philip J. Bernhard, Ph.D.
Associate Professor and Department Head
Computer Engineering and Sciences

Abstract

Title: Cross-platform development of Wake-Up-Word

Author: Christopher Ryan Woodle

Advisor: Veton Kepuska, Ph.D.

The goal of this project will be to explore cross-platform implementation of Wake-Up-Word (WUW). To enable the development of future speech-based artificial intelligence applications, it is important to have robust and accessible implementations of WUW. Adoption of Unix based operating systems continues to expand for server, backend, and embedded applications, therefore a WUW implementation in Unix will become essential. As web technologies continue to grow, WUW will also need to be implemented in web, using technologies such as JavaScript and Web Assembly (WASM).

This project encompasses porting the previous implementation of WUW from Microsoft Windows to Unix, building a new WUW model with optimizations, and new front-end processing implementations in web technologies.

Table of Contents

List of Figures

List of Tables

Chapter 1:
Introduction

Background of Wake-Up-Word

Wake Up Word (WUW) originally created by Veton Kepuska, Ph.D, is a method to request the attention of a computer using a spoken key word (Veton Kepuska 2011). The WUW application uses signal processing, feature extraction, hidden markov model (HMM) evaluation and support vector machines (SVM) to process speech data and determine if the speaker has requested the computer's attention.

Cross platform and open source

Having an accurate and dependable WUW implementation will enable speech-based artificial intelligence applications to continue to improve. This project addresses bringing WUW to Unix and web applications with developer accessibility as core focus. The source code and data sets are published online for free, under creative commons license.

Chapter 2
Porting to Unix

Technology choices

The original implementation of WUW in Windows was written in C++ using Visual Studio. To port the application to Unix environments, the application had to be modified to replace all Windows platform specific libraries and requirements with standard libraries. The Visual Studio toolchain also had to be replaced with Unix compatible build environments. CMake was selected for its popularity and ease of use over Makefiles.

To simplify maintaining build environments to publish binaries, a containerized environment using Docker was configured.

Chapter 3
Model building

New Model

For developers to use WUW with different key words, new WUW models will have to be created. The new Unix implementation includes a new model training process. To verify new training process, this project created a new model for the key word “Operator.” This new model can be compared to the original “Operator” model created for the Windows WUW.

Models for WUW consist of four components, three HMMs and one SVM model. The HMMs are trained on vectors of MFCC, LPC, Enhanced MFCC data. The SVM is a two-class radial basis kernel model. It determines if the triggered data is in vocabulary or out of vocabulary, and is trained using the scores produced by the HMMs.

To enable easier access to building new models, the model training process was scripted using JavaScript and NodeJS, and executed in containerized environments with Docker.

Corpus

The new operator model was built using the same corpus, WUW Corpus II, which was used to train the original “Operator” model. The corpus contains approximately 3000 recordings at 8khz. 597 of these samples were labeled as in-vocab, either labeled 001 or 006.

Upon completion of the new operator model, the corpus was published to GitHub. During development the corpus was cleaned of corrupted files and file checksums were added.

Voice Activity Detector tuning

The Voice Activity Detector (VAD) determines if audio is speech data. It uses a pre-trained embedded SVM model. If the audio is determined to be speech data, the feature vectors are then sent to the HMMs for evaluation.

After reviewing incorrectly classified samples, many revealed incorrect VAD triggers. There were two types of incorrect trigger, the first being delayed triggers, and the second only triggering for part of the utterance. In the following example, the audio sample contains only the desired utterance "Operator." The VAD was only triggered in the beginning of the utterance (highlighted section).

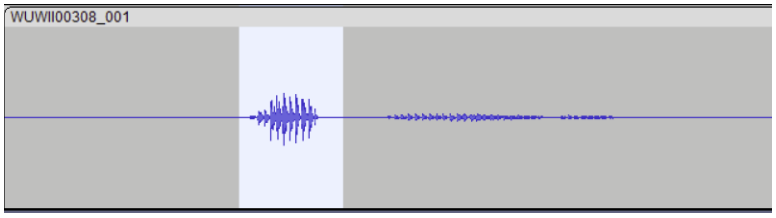

Figure 1: Partial VAD trigger

This gap in the utterance is the stop consonant "p" of "Operator." Utterances with a long gap cause the VAD to disable before the whole utterance is complete. The first attempt to correct this issue was to increase the minimum off count parameter in the VAD source code. Increasing from the default 15 frames to 20 frames yields the following improvement.

Figure 2: Improved partial VAD trigger

While more of the utterance was captured, the VAD was still disabling too early. This was theorized to be due to low speaker energy or an increased distance from the speaker to the microphone. The source data was manually amplified in the following section.

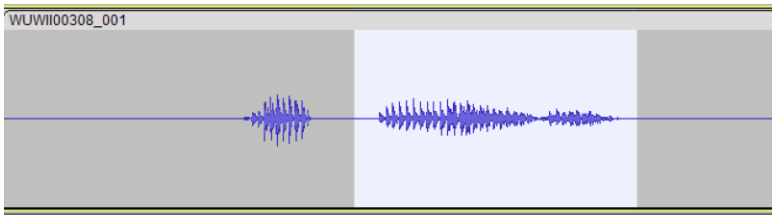

Figure 3: Amplified audio section

Running the VAD after amplification produced the desired results.

Figure 4: Amplified section VAD trigger

Model results

To evaluate the model results, true positive samples (samples containing the desired utterance) were selected from the corpus if the utterance was labeled with 001 (only "operator" utterance) or 006 ("operator" utterance used in a sentence), and with time stamps for the start and end of the utterance in the sample.

The final model trained using the increased VAD minimum off count and a subset of manually corrected samples resulted in a 3.3% improvement in accuracy.

Many of the remaining false negatives are due to low speech energy, which indicates that VAD embedded SVM was trained with louder speakers.

Table 1: Model Performance

Metric	***Original "Operator" Model***	***New "Operator" Model***
Accuracy	91.8%	95.1%
False positives	8	2
False negatives	275	168

Chapter 4
Wake-Up-Word for web

Technology selection

This project explores a universal approach to implementing WUW front end components in web technologies. TypeScript and AssemblyScript was chosen for their ability to compile the same code to both native JavaScript and Web Assembly. This will allow for direct comparison of benchmarks to determine the faster solution. WASM was expected to perform faster than native JavaScript.

For ease of development, utilizing a bundler application was necessary. This will allow consumers of WUW library components to have the same application programming interface across both server based NodeJS and browser based applications.

```
import { instantiate } from 'fft-as';

// Load library and wasm
instantiate().then(exports => {
    // Generate 512 samples of 20hz sine wave
    const LENGTH = 512;
    const FREQUENCY = 20;
    const samples = Array.from(Array(LENGTH).keys())
        .map(x => Math.sin(x * FREQUENCY))
        .concat(new Array(LENGTH).fill(0));

    // Compute FFT
    const result = exports.fft(samples);
    console.log(result);

});
```

Figure 5: Universal web design example

Base64 Rollup plugin

Due to limitations in browser based WASM loading, the binary was base64 encoded and embedded into the Rollup bundle. This required a custom plugin for the bundler. Embedding the binary was not without tradeoffs. Encoding binary to base64 has on average a 30% file size increase, resulting in slower initial load times. This tradeoff was deemed acceptable to remove the requirement of web server WASM hosting for browser-based applications. In the future as browser modules improve, this encoding step may become unnecessary.

Front end components

Benchmarking the FFT implementation as conducted in NodeJS. Two different tests were conducted. The “cold boot” test measured the execution time of a single FFT of a 512-sample sine wave, 100 times, starting a new process each time. The “consecutive calls” test measured the average execution time of one hundred different 512 sample sine waves without exiting the process.

Table 2: FFT Benchmark

Metric	*WASM*	*Native JavaScript*
Cold boot	4.5 ms	6 ms
Consecutive calls	4.2 ms	0.36 ms

The cold boot test produced expected results with the WASM runtime having faster execution time. However, the consecutive calls test produced unexpected results. Subsequent calls to the native JavaScript implementation executed progressively faster, indicating the runtime engine V8 of NodeJS preforming optimizations. These results indicate that JavaScript may be a more suitable runtime than WASM after some warmup period.

Chapter 5
Conclusion

Opportunities for improvement

While minor tweaks to the VAD parameters produced improvement, replacing the VAD with a different technology entirely, such as neural networks paired with dedicated neural hardware, could yield better accuracy while achieving the intent of reduced compute. Similarly, replacing the two class SVM with a neural network could also increase performance.

For web based WUW, metrics indicate NodeJS based WUW frontend is performant enough to be viable. More investigation into Web Workers should be conducted to determine if WUW would be suitable for website integrations, as processing for WUW should be executed in a background thread to prevent the user interface from being interrupted.

References

Kepuska, V. (2011). Wake-Up-Word Speech Recognition. InTech. doi: 10.5772/16242

Appendix

Open-source repositories

All applications developed for this project can be found at the following links:

https://github.com/wake-up-word/wuw-linux

https://github.com/wake-up-word/wuw-windows

https://github.com/wake-up-word/wuw-corpus-ii

https://github.com/wake-up-word/wuw-js