Howl: A Deployed, Open-Source Wake Word Detection System

Raphael Tang,[1*] **Jaejun Lee,**[1*] **Afsaneh Razi,**[2] **Julia Cambre,**[2]
Ian Bicking,[2] **Jofish Kaye,**[2] and **Jimmy Lin**[1]

[1] David R. Cheriton School of Computer Science, University of Waterloo
[2] Mozilla

Abstract

We describe Howl, an open-source wake word detection toolkit with native support for open speech datasets such as Mozilla Common Voice (MCV) and Google Speech Commands (GSC). We report benchmark results of various models supported by our toolkit on GSC and our own freely available wake word detection dataset, built from MCV. One of our models is deployed in Firefox Voice, a plugin enabling speech interactivity for the Firefox web browser. Howl represents, to the best of our knowledge, the first fully productionized, open-source wake word detection toolkit with a web browser deployment target. Our codebase is at howl.ai.

1 Introduction

Wake word detection is the task of recognizing an utterance for activating a speech assistant, such as "Hey, Alexa" for the Amazon Echo. Given that such systems are meant to support fully automatic speech recognition, the task seems simple. However, it introduces a different set of challenges because these systems have to be always listening, computationally efficient, and, most of all, privacy respecting. Therefore, researchers treat it as a separate line of work, with most recent advancements driven by neural networks (Sainath and Parada, 2015; Tang and Lin, 2018).

Unfortunately, most existing toolkits are closed source and often specific to a target platform. Such design choices restrict the flexibility of the application and add unnecessary maintenance as the number of target domains increases. We argue that using JavaScript is a solution: unlike many languages and their runtimes, the JavaScript engine powers a wide range of modern user-facing applications ranging from mobile to desktop ones.

To this end, we have previously developed Honkling, a JavaScript-based keyword spotting system (Lee et al., 2019). Leveraging one of the lightest models available for the task from Tang and Lin (2018), Honkling efficiently detects the target commands with high precision. However, we notice that Honkling is still quite far from being a stable wake word detection system. This gap mainly arises from the model being trained as a speech commands classifier instead of a wake word detector; its high false alarm rate results from the limited number of negative samples in the training dataset (Warden, 2018).

In this paper, to achieve greater real-world impact, we close this gap in the Honkling ecosystem and present Howl, an open-source wake word detection toolkit with support for open datasets such as Mozilla Common Voice (MCV; Ardila et al., 2019) and the Google Speech Commands dataset (GSC; Warden, 2018). Howl is the first in-browser wake word detection system which powers a widely deployed consumer application, Firefox Voice.[1] By processing the audio in the browser and being completely open source, including the datasets and models, Howl is a privacy-respecting, non-eavesdropping toolkit that users can trust. With a false reject rate of 16% at five false alarms per hour of speech, our deployed model has enabled Firefox Voice to provide a completely hands-free experience to over 8,000 users in the 9 days since its launch in August 2020.[2]

2 Background and Related Work

Although mainstream voice technologies such as Siri and Alexa are driven by proprietary wake word detection systems, open toolkits like Porcupine and

[*] Equal contribution. Order decided by coin flip.

[1] `https://github.com/mozilla-extensions/firefox-voice`

[2] `https://twitter.com/jofish/status/1293017304423215104`

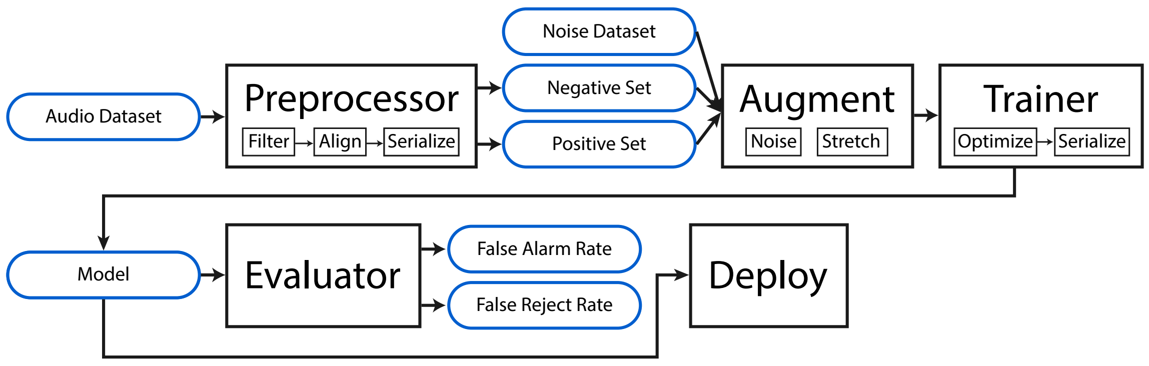

Figure 1: An illustration of Howl's end-to-end pipeline and its control flow. First, we preprocess the incoming audio dataset by filtering for the wake word vocabulary, aligning the speech, and saving the negative and positives examples to disk. Next, we introduce a noise dataset and augment the data on the fly at training time. Finally, we evaluate the optimized model and, if the results are satisfactory, export it for deployment.

Snowboy also exist. Such ecosystems provide an open-source modeling toolkit, some data, and deployment capabilities. Unfortunately, these ecosystems are still closed at heart; they keep their data, models, or deployment proprietary. As far as open-source ecosystems go, Precise[3] represents a step in the right direction, but its datasets are limited, and its deployment target is the Raspberry Pi.

We further make the distinction between wake word detection and speech commands classification toolkits such as Honk (Tang and Lin, 2017). These frameworks focus on classifying fixed-length audio as one of a few dozen keywords, with no evaluation on a sizable negative set, as required in wake word detection. While these trained models may be used in detection applications, they are not rigorously tested for such.

3 System Description

We present a high-level description of our toolkit and its goals (see Howl's architecture in Figure 1). For specific details, we refer users to our code repository, as linked in the abstract.

3.1 Requirements

Howl is written in Python 3.7+, with notable dependencies being PyTorch (Paszke et al., 2019) for model training, Librosa (McFee et al., 2015) for audio preprocessing, and the Montreal Forced Aligner (MFA; McAuliffe et al., 2017) for speech data alignment. We release Howl under the Mozilla Public License v2, a file-level copyleft free license. For speedy model training, we recommend a CUDA-enabled graphics card with at least 4GB of VRAM; we used an Nvidia Titan RTX in all of our experiments. For resource-restricted users, we suggest exploring Google Colab[4] and other cloud-based solutions.

3.2 Components and Pipeline

Howl consists of the three following major components: audio preprocessing, data augmentation, and model training and evaluation. These components form a pipeline, in the written order, for producing deployable models from raw audio data.

Preprocessing. A wake word dataset must first be preprocessed from an annotated data source, which is defined as a collection of (audio, transcription) pairs, with predefined training, development, and test splits. Since Howl is a frame-level keyword spotting system, it relies on a forced aligner to provide word- or phone-based alignment. We choose MFA for its popularity and free license, and hence Howl structures the processed datasets to interface well with MFA.

Another preprocessing task is to parse the global configuration settings for the framework. Such settings include the learning rate, the dataset path, and model-specific hyperparameters. The toolkit reads in most of these settings as environment variables, which enable easy shell scripting.

Augmentation. For improved robustness and better model quality, we implement a set of popular

[3]https://github.com/MycroftAI/mycroft-precise

[4]https://colab.research.google.com/

augmentation routines: time stretching, time shifting, synthetic noise addition, recorded noise mixing, SpecAugment (without time warping; Park et al., 2019), and vocal tract length perturbation (Jaitly and Hinton, 2013). These are readily extensible, so practitioners may easily add new augmentation modules.

Training and evaluation. Howl provides several off-the-shelf neural models, as well as training and evaluation routines using PyTorch for computing the loss gradient and the task-specific metrics, such as the false alarm rate and reject rate. These routines are also responsible for serializing the model and exporting it to our browser-side deployment.

Pipeline. Given these components, our pipeline, visually presented in Figure 1, is as follows: First, users produce a wake word detection dataset, either manually or from a data source like Common Voice and Google Speech Commands, setting the appropriate environment variables. This can be quickly accomplished using Common Voice, whose ample breadth and coverage of popular English words allow for a wide selection of custom wake words; for example, it has about a thousand occurrences of the word "next." In addition to a positive subset containing the vocabulary and wake word, this dataset ideally contains a sizable negative set, which is necessary for more robust models and a more accurate evaluation of the false positive rate.

Next, users (optionally) select which augmentation modules to use, and they train a model with the provided hyperparameters on the selected dataset, which is first processed into log-Mel frames with zero mean and unit variance, as is standard. This training process should take less than a few hours on a GPU-capable device for most use cases, including ours. Finally, users may run the model in the included command line interface demo or deploy it to the browser using Honkling, our in-browser keyword spotting (KWS) system, if the model is supported (Lee et al., 2019).

3.3 Data and Models

For the data sources, Howl works out of the box with Mozilla Common Voice, a general speech corpus, and Google Speech Commands, a commands recognition dataset. Users can quickly extend Howl to accept other speech corpora such as LibriSpeech (Panayotov et al., 2015) or the Hey Snips dataset (Coucke et al., 2019). Howl also accepts any folder that contains audio files and interprets them as recorded noise for data augmentation, which covers popular noise datasets such as MUSAN (Snyder et al., 2015) and Microsoft SNSD (Reddy et al., 2019).

For modeling, Howl provides implementations of convolutional neural networks (CNNs) and recurrent neural networks (RNNs) for wake word detection. These models are from the existing literature, such as residual CNNs (Tang and Lin, 2018), a modified listen–attend–spell (LAS) encoder (Chan et al., 2015; Park et al., 2019), and MobileNetv2 (Sandler et al., 2018). Most of the models are lightweight since the end application requires efficient inference, though some are parameter heavy to establish a rough upper bound on the quality, as far as parameters go. Of particular focus is the lightweight `res8` model (Tang and Lin, 2018), which is directly exportable to Honkling, the in-browser KWS system. For this reason, we choose it in our deployment to Firefox Voice.

Model	Dev/Test	# Par.
EdgeSpeechNet (Lin et al., 2018)	–/96.8	107K
`res8` (Tang and Lin, 2018)	–/94.1	110K
RNN (de Andrade et al., 2018)	–/95.6	202K
DenseNet (Zeng and Xiao, 2019)	–/97.5	250K
Our `res8`	**97.6/97.8**	111K
Our LSTM	95.6/95.2	128K
Our LAS encoder	97.6/97.7	478K
Our MobileNetv2	97.3/97.3	2.3M

Table 1: Model accuracy (%) on Google Speech Commands with the number of parameters. `res8` achieves the best accuracy considering the small model size.

4 Benchmark Results

To verify the correctness of our implementation, we first train and evaluate our models on the Google Speech Commands dataset, for which there exists many known results. Next, we curate a wake word detection datasets and report our resulting model quality. Training details are in the repository.

Commands recognition. Table 1 summarizes the metrics collected from Howl for the twelve-keyword recognition task from Speech Commands (v1), where we classify a one-second clip as one of "yes," "no," "up," "down," "left," "right," "on," "off," "stop," "go," unknown, or silence. We report average accuracy collected from fifty iterations. The results indicate that our implementations are com-

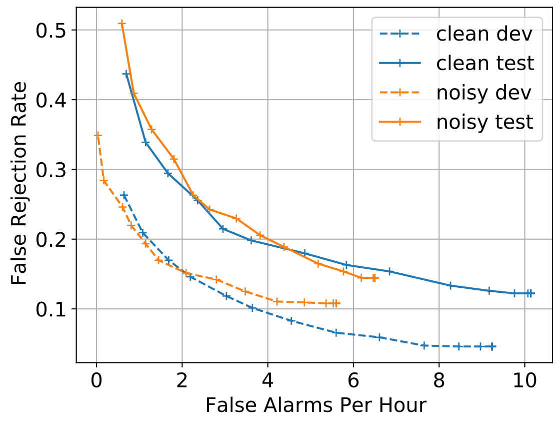

Figure 2: Receiver operating characteristic (ROC) curves for the wake word. The threshold ranges from 0.05 to 1.00 with a marker for every increment of 0.05.

petitive with the state of the art, with the res8 model achieving the highest accuracy of 97.8% on the test set, despite having fewer parameters. Our other implemented models, the LSTM, LAS encoder, and MobileNetv2, compare favorably.

Wake word detection. For wake word detection, we target "hey, Firefox" for waking up Firefox Voice. From the single-word segment of MCV, we use 1,894 and 1,877 recordings of "hey" and "Firefox," respectively; from the MCV general speech corpus, we select all 1,037 recordings containing "hey," "fire," or "fox." We additionally collect 632 recordings of "hey, Firefox" from volunteers. For the negative set, we use about 10% of the entire MCV speech corpus. We choose the training, dev, and test splits to be 80%, 10%, and 10% of the resulting corpus, stratified by speaker IDs for the positive set. For robustness to noise, we use portions of MUSAN and SNSD as the noise dataset. We arrive at 31 hours of data for training and 3 hours each for dev and test.

For the model, we select res8 (Tang and Lin, 2018) for its high quality on Speech Commands (see evaluation results above) and easy adaptability with our browser deployment target. We follow the pipeline mentioned in the previous section to train ten models with different seeds; details are not repeated, and hyperparameters can be found in the repository.

In Figure 2, we present the resulting receiver operating characteristic (ROC) curves generated from the averaged metrics. As we increase the threshold in increments of 0.05, we naturally observe lower false alarm rates at the expense of higher false reject rates. From the figure, we find that, at a threshold of 0.8, Howl achieves five false alarms per hour of speech with an acceptable 16% false reject rate. Our negative set contains diverse adversarial examples that misrepresent real-world usage, e.g., many utterances of "Firefox," which are responsible for at least 90% of the false positives. Thus, combined with preliminary results from live testing the system ourselves, we comfortably choose the operating point at five false alarms per hour.

We finally note that the discrepancy between the dev and test curves is likely explained by differences in the data distribution, not hyperparameter fiddling, because there are only 76 and 54 clips in the positive dev and test sets, respectively.

5 Browser Deployment

To protect user security and privacy, wake word detection must be directly performed on the user's device. This setting introduces various technical challenges, as the available resources are often limited and may not be accessible. In the case of Firefox Voice, our target application, the platform is Firefox, where the major challenge is the limited support in machine learning frameworks.

However, our previous work demonstrates the feasibility of in-browser wake word detection with Honkling (Lee et al., 2019). Our application is written purely in JavaScript and supports different models using TensorFlow.js. Since our task is to provide an accurate wake word detection system for Firefox Voice, we rewrite the audio processing logic to match the new Python pipeline and optimize various preprocessing routines to substantially reduce the computational burden.

To measure the performance of our application, we refer to the built-in energy impact metric of Firefox, which reports the CPU consumption of each open tab. To establish a reference, playing a YouTube video reports an average energy impact of 10, while a static Google search reports 0.1. Our wake word detection model yields an energy impact of only 3, which efficiently enables hands-free interaction for initiating the speech recognition engine. Our wake word detection demo and browser-side integration details can be found at https://github.com/castorini/howl-deploy.

6 Conclusions and Future Work

This paper introduces Howl, the first in-browser wake word detection system which powers a widely deployed application, Firefox Voice. Leveraging a continuously growing speech dataset, Howl enables a community-based endeavour for building a privacy-respecting and non-eavesdropping wake word detection system. To expand the scope of Howl, our future work includes embedded systems as deployment targets, where the computational resources are even more constrained, with some systems lacking even modern memory managers.

7 Acknowledgments

This work was supported by the Canada First Research Excellence Fund and the Natural Sciences and Engineering Research Council of Canada.

References

Douglas Coimbra de Andrade, Sabato Leo, Martin Loesener Da Silva Viana, and Christoph Bernkopf. 2018. A neural attention model for speech command recognition. *arXiv:1808.08929*.

Rosana Ardila, Megan Branson, Kelly Davis, Michael Henretty, Michael Kohler, Josh Meyer, Reuben Morais, Lindsay Saunders, Francis M. Tyers, and Gregor Weber. 2019. Common Voice: A massively-multilingual speech corpus. *arXiv:1912.06670*.

William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. 2015. Listen, attend and spell. *arXiv:1508.01211*.

Alice Coucke, Mohammed Chlieh, Thibault Gisselbrecht, David Leroy, Mathieu Poumeyrol, and Thibaut Lavril. 2019. Efficient keyword spotting using dilated convolutions and gating. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*.

Navdeep Jaitly and Geoffrey E. Hinton. 2013. Vocal Tract Length Perturbation (VTLP) improves speech recognition. In *Proceedings of the ICML Workshop on Deep Learning for Audio, Speech and Language*.

Jaejun Lee, Raphael Tang, and Jimmy Lin. 2019. Honkling: In-browser personalization for ubiquitous keyword spotting. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*.

Zhong Qiu Lin, Audrey G. Chung, and Alexander Wong. 2018. EdgeSpeechNets: Highly efficient deep neural networks for speech recognition on the edge. *arXiv:1810.08559*.

Michael McAuliffe, Michaela Socolof, Sarah Mihuc, Michael Wagner, and Morgan Sonderegger. 2017. Montreal Forced Aligner: Trainable text-speech alignment using Kaldi. In *Proceedings of the Eighteenth Annual Conference of the International Speech Communication Association*.

Brian McFee, Colin Raffel, Dawen Liang, Daniel P. W. Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. 2015. librosa: Audio and music signal analysis in Python. In *Proceedings of the 14th Python in Science Conference*.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. LibriSpeech: An ASR corpus based on public domain audio books. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*.

Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin Dogus Cubuk, and Quoc V. Le. 2019. SpecAugment: A simple augmentation method for automatic speech recognition. In *Proceedings of the Twentieth Annual Conference of the International Speech Communication Association*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035.

Chandan K. A. Reddy, Ebrahim Beyrami, Jamie Pool, Ross Cutler, Sriram Srinivasan, and Johannes Gehrke. 2019. A scalable noisy speech dataset and online subjective test framework. In *Proceedings of the Twentieth Annual Conference of the International Speech Communication Association*.

Tara N. Sainath and Carolina Parada. 2015. Convolutional neural networks for small-footprint keyword spotting. In *Proceedings of the Sixteenth Annual Conference of the International Speech Communication Association*.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

David Snyder, Guoguo Chen, and Daniel Povey. 2015. MUSAN: A music, speech, and noise corpus. *arXiv:1510.08484*.

Raphael Tang and Jimmy Lin. 2017. Honk: A PyTorch reimplementation of convolutional neural networks for keyword spotting. *arXiv:1710.06554*.

Raphael Tang and Jimmy Lin. 2018. Deep residual learning for small-footprint keyword spotting. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*.

Pete Warden. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv:1804.03209*.

Mengjun Zeng and Nanfeng Xiao. 2019. Effective combination of DenseNet and BiLSTM for keyword spotting. *IEEE Access*.