KEYWORD SPOTTING ON BRILLIANT LABS MONOCLE

Aghyad Deeb [1] Emeka Ezike [1] Jayson Lin [1]

ABSTRACT

This project aims to develop a keyword spotting model tailored for the Brilliant Labs' Monocle, a pocket-sized augmented reality (AR) device. The proposed keyword spotting model can enhance user interaction by enabling seamless and intuitive control through voice commands.

The methodology involves collecting and curating a dataset of relevant voice commands and training a recognition model using Tensorflow. Then, using Tensorflow Lite Micro (TFLM), this model will be ported to the Monocle's microcontroller unit (MCU) in order to run on-device inference with minimal computational overhead while maintaining high accuracy. This resource-efficient model should run on the Monocle, where its performance can be evaluated. A variety of metrics will assess the model's accuracy and inference latency.

By creating a tailored keyword spotting model for Brilliant Labs Monocle, this project contributes to the advancement of AR technology, fostering a more interactive and engaging experience. Additionally, the model's adaptability to the constraints of a microcontroller environment showcases the feasibility of implementing sophisticated machine learning applications on resource-constrained devices, paving the way for innovative developments of Tiny Machine Learning in AR.

1 INTRODUCTION

While AR devices hold tremendous promise for transforming the way we interact with the digital and physical worlds, current devices still face several inconveniences that hinder widespread adoption and seamless integration into daily life.

One major hurdle is the form factor and comfort of AR hardware. Many existing devices, especially smart glasses and headsets, can be bulky and conspicuous. Users may find them uncomfortable to wear for extended periods, and the aesthetics may not align with mainstream fashion preferences.

Affordability is another critical factor influencing the widespread adoption of AR devices. Currently, many AR devices come with a hefty price tag in the thousands of dollars, making them less accessible to the general consumer market. The high costs are primarily associated with the complex technology, advanced optics, and specialized components required for an effective AR experience.

Brilliant Labs, a Singapore-based tech startup, aims to address these hurdles with the introduction of their Monocle, an open-source, pocket-sized AR lens that can be clipped on any eyewear or held to the eye and has a much more accessible price tag of $349 (Brilliant-Labs, b).

Being an open-source product, there is currently no established method for interacting with the Monocle other than the two buttons it has. The adoption of these devices requires an efficient and dependable control mechanism. Numerous methods exist for controlling AR devices, each carrying its own set of benefits and constraints. The overarching objective is to furnish users with instinctive and smooth interactions, thereby enriching their AR experiences.

Keyword spotting (KWS) represents a method that has gained popularity for its convenience and versatility. It involves recognizing specific spoken keywords or phrases to trigger actions or commands. Keyword spotting is efficient, allowing users to initiate functions with a simple voice prompt. In terms of convenience, keyword spotting stands out because it requires minimal effort and is easily integrated into daily tasks. Users can quickly and naturally issue commands without the need for extensive training or memorization of specific gestures.

To address the limitations of the Brilliant Labs Monocle, featuring a Nordic nRF52832 microcontroller with restricted Flash memory (512KB) and RAM (64KB) (Brilliant-Labs, a), TensorFlow Lite Micro (TFLM) emerges as an ideal solution for implementing efficient keyword spotting. TFLM, a lightweight, open-source machine learning framework tailored for microcontrollers and resource-constrained edge devices, offers a primary advantage in deploying compact machine learning models on devices with limited computational resources. This is especially beneficial for the Monocle MCU, where memory and power constraints are crucial considerations. The implementation process involves training and quantizing a keyword spotting model suitable for the Monocle MCU. TFLM provides tools and converters for the conversion of TensorFlow models into a format compatible with microcontrollers. Additionally, developers can customize the build to include only necessary operations, further optimizing the model for the specific hardware architecture.

Currently, there is a lack of locally running keyword-spotting models on low-powered MCUs for widespread consumer use. This paper aims to initiate a trend by porting a keyword-spotting model to run locally on a Brilliant Labs Monocle.

2 Background and Related Work

The recent trend towards edge computing in KWS applications signifies a shift from cloud-based processing to local processing, reducing latency and preserving privacy. This advancement is crucial in applications where real-time response is critical, such as in wake word detection, safety-critical wearable devices, or when providing immediate feedback in smart home systems.

The first works of KWS appeared in the late 1980s, laying the groundwork for the technology used today (Rohlicek et al., 1989). Traditional KWS models have employed various algorithms like sliding window, garbage models, and convolutional neural networks on Mel-frequency cepstrum coefficients, and transformer-based models (Tang et al., 2018; Wei et al., 2021). Recent work has found that a DS-CNN model achieves an accuracy of 95.4% and is compact, making it effective to be used on edge devices (Zhang et al., 2018). Additional work also shows that by replacing the digital preprocessing with a proposed analog front-end, the energy required for data acquisition and preprocessing can be reduced for KWS (Cerutti et al., 2022). One of KWS's contributions today is its crucial role in smart voice assistants like Google Assistant and Siri. These systems typically begin with wake-word detection followed by keyword spotting, which is less compute-intensive compared to general automatic speech recognition (ASR) and can be performed on-device with low latency, an important feature for successful edge device deployment (ARM Community, 2022). The ongoing improvements in KWS, driven by machine learning and edge computing, continue to expand its applications, making it an increasingly integral component of modern voice-activated systems in small edge devices.

In the realm of ML-powered edge devices, the Brilliant Labs Monocle stands out as a promising addition to the field, boasting distinctive features and applications that position it well for edge ML development. Weighing a mere 15g, the Monocle's compact and lightweight design makes it an ideal complement to the evolving landscape of ML-empowered wearable technology. Equipped with performance hardware like the previously mentioned FPGA accelerator, the Monocle is empowered to effectively handle machine learning and computer vision use cases.

Notably, Brilliant Labs' emphasis on development and integration flexibility adds to the Monocle's appeal, making it a valuable investment for research endeavors. This flexibility streamlines development processes and ensures that the device remains relevant for ongoing updates and utilization in the years to come. For programming convenience, the Monocle supports AR Studio for VSCode, facilitating the development, testing, and storage of applications. Moreover, its integration with mobile apps through Bluetooth, compatible with both iOS and Android, enhances its functionality.

As mentioned previously, the Brilliant Labs Monocle is powered by a Nordic nRF52832 microcontroller. Therefore, initiating development on a Nordic nRF52DK board serves as the initial step, as successes achieved on this board seamlessly translate to the Monocle's MCU. The nRF52DK, a versatile single-board development kit, is specifically designed for Bluetooth Low Energy, Bluetooth mesh, NFC, ANT, and 2.4 GHz proprietary development. Its foundation on the nRF52805, nRF52810, and nRF52832 System on Chips (SoCs) allows for the creation of a broad spectrum of applications, particularly in the realms of IoT and wearable devices.

A notable limitation of the board, which may be addressed in the future, is the absence of support on Edge Impulse. Presently, only the nRF52840 series enjoys compatibility, necessitating a separate and more intricate method for implementing the KWS model on the MCU. Despite this challenge, a compelling illustration of the nRF52 DK's prowess in machine learning is evident in the 'ElephantEdge' wildlife tracker challenge.

In this competition, the winning design, 'EleTect,' successfully leveraged the nRF52840 SoC, a pivotal component adjacent to the nRF52 DK. This IoT and TinyML-based wildlife tracker showcased a diverse array of models designed for tasks such as monitoring elephant movements, predicting behaviors, and detecting mood swings (Cerutti et al., 2022). Despite the specific constraints posed by the nRF52840 series, 'EleTect' demonstrated the board's capability to excel in sophisticated machine learning applications.

Given this context, we posit that employing the nRF52 DK to explore the feasibility of porting firmware and software to the Monocle is paramount to the success of our project. The exemplary performance in the 'ElephantEdge' challenge underscores the potential of adapting and optimizing applications for the Monocle, despite the current limitations posed by Edge Impulse compatibility.

Lastly, this paper is an extension of Harrison Zhang's research, which centered on the utilization of an nRF52DK board as an intermediary for compiling TFLM firmware and a KWS model onto a Brilliant Labs Monocle. Harrison's contribution was pivotal in assembling the essential components of the project, including TFLM firmware, Monocle MicroPython, and a pre-downsized KWS model. However, his Makefile was incomplete and unable to seamlessly integrate the gathered components.

The work of this paper builds upon Harrison's foundation by enhancing the Makefile to ensure the proper compilation of firmware and model onto a nRF52DK board. We address the deficiencies in the Makefile, providing a comprehensive solution that streamlines the compilation process and ensures the successful assembly of the project components.

Furthermore, we extend Harrison's efforts by developing scripts to facilitate experimentation with data collection and augmentation for audio recorded by the Monocle. This additional step enhances the project's capabilities, allowing for a more comprehensive exploration of audio data and contributing to the overall advancement of the research initiated by Harrison Zhang.

3 Methods and Challenges

The main goal is to be able to run ML models locally on the Monocle. The Monocle has two processing units that can be used to run the models. The first is the Bluetooth MCU, Nordic nRF52832. The second is the FPGA, Gowin GW1N-LV9MG100C6/I5. While we speculate that the FPGA would be able to run models more efficiently and would be better for leveraging parallel processing of the Matrix Matrix Multiplications, we opted for working on the Bluetooth MCU given the lack of experience of the team with FPGAs and the time constraint.

The original firmware of the Monocle incorporated MicroPython such that developers can build software on top of the firmware. Modifying this firmware would provide a developer-friendly environment and gives us a starting point to work with.

When it comes to the framework for running the Machine Learning models, we use TFLM as it's a very common framework for ML on microcontrollers with extensive community support.

The first step is to incorporate TFLM with the Monocle's MicroPython firmware. To make this work, the TFLM repository files were added to the root folder of the firmware of the monocle. The next step is to compile the files of TFLM with the monocle's firmware. The work of Harrison Zhang was a significant help in achieving this part. We weren't able to use his code immediately, however. When we tried to run the makefiles in the repository he created, we faced a lot of errors that led to us being unable to compile the firmware with TFLM immediately. We spent several days trying to debug and understand why the code wasn't working.

After getting nowhere, we decided to spend time learning how to write makefiles to be able to understand how to debug the complex makefile that we could not get to work. To explain how we got TFLM to compile with the firmware, we explain the process needed to fix the makefiles, which was where most of our time was spent. We started interpreting all the makefiles that contribute to building the Monocle's firmware. We found that the Monocle builds on top of three makefiles provided by MicroPython by adding a makefile that adds compatibility to the Monocle.

The makefile in the original Monocle's repository (Labs, 2023) does the following:

1. Configure the compilation toolchain to work with the monocle by using the "arm-none-eabi" toolchain
2. Adds optimization flags to be used in compilation
3. Adds the directories of the header files needed for compilation to be included as flags in the compilation command
4. Add the source files needed to create the modules that control the monocle's sensors and other source files necessary for compatibility

To build the firmware, the Monocle's original makefile does the following:

1. Uses one of MicroPython's makefiles to compile C files into object files
2. Compiles all object files into an 'application.elf' files
3. Changes the format from 'application.elf' to 'application.hex'

We went through the makefiles inside MicroPython, and the most important part is that they add compilation rules to create object files out of '.c' and '.cpp' files. These rules are then used to compile all the source files of the monocle's firmware to create the 'application.elf' file and so on.

We then went through the modified makefile that included the compilation of TFLM in addition to the compilation of the firmware into one hex file. This file was created by combining the original makefile of TFLM and the makefile of the mMnocle's firmware. The file adds the source files of TFLM, which are written in C++ with the extension '.cc' and adds the necessary flags for compilation, in addition to the original components of the Monocle's makefile.

After understanding how the final makefile is structured, we tried to build the firmware once again to debug it. We noticed that the compiler declared that several object files were missing. After investigating these missing object files, we noticed that all of them were supposed to be compiled from '.cc' files, which led us to understand that there was a problem with the compilation of some C++ files, although other C++ files were compiling successfully.

Our first intuition was that this was due to incorrect specification of the directories, so we created rules to print the directories the makefile was using and they were all correct. We then checked the C++ files compiler, which was correctly setup. We looked deeper into the makefile rule responsible for compiling C++ files into object files. The rule used the pattern '%.cpp' to create object files, and most of Tensorflow files had the extension '.cc'. We added a rule to compile '.cc' files into object files and this fixed the problems we were having with building the firmware and now we had a working version of the firmware with TFLM.

Since TFLM is written in C++, we need a C++ file to use it. In order to do that, we create a new MicroPython module. This module, which can be found at "/modules/kws.c" contains one function that runs keyword spotting indefinitely. The module defines a run method and attaches it to the appropriate MicroPython object. The implementation of the run function is handled by another C++ file, located at "/modules/helpers/kws_helper.cc". This helper file loops indefinitely running the micro speech model defined in the TFLM examples, defines the operations needed to run the model, runs the model on two pre-recorded spectrogram files, the first is a recording saying the word "yes", the other saying the word "no". The function then prints the classification results of the model, which successfully classifies each word in its corresponding class.

In the effort to run the model on the Monocle, we also had to extract audio from the Monocle's microphone. Using the Monocle MicroPython API, we were able to extract audio data stored on the FPGA. However, processing the audio proved to be a challenge due to the lack of audio modules in MicroPython. Therefore, we had to manually create .wav files by writing a function that properly inserts relevant byte information, but as shown in "example_monocle_recording_.wav", the audio quality was shockingly poor. However, it is unclear whether this is because of the microphone or the .wav conversion function.

Lastly, we also needed to create functions that converted .wav files to spectrograms in order for the KWS model to take the audio as input. This also involved research on audio spectrogram transformation in order to create the functions needed to convert the data. In "spectrogram.py", the conversion functions we used to create spectrograms from .wav files are detailed.

4 Insights and Findings

Our work shows that it's possible to configure TFLM to work with the monocle's firmware. Further, we show that running a model locally on the monocle's Bluetooth MCU is possible despite the limited computational ability, without facing significant latency.

From our estimations, inference takes about 1200 milliseconds on an audio sample that is converted into a 1960 byte spectrogram sample. This is fairly fast and should suffice for an adequate user experience.

5 Next Steps

To further enhance the capabilities of the KWS model for Brilliant Labs Monocle, consider the following comprehensive next steps:

Audio Signal Processing Explore and implement signal processing techniques to enhance the quality of the microphone recordings. This may involve noise reduction, echo cancellation, or other methods to improve the accuracy of keyword recognition.

Model Fine-Tuning Fine-tune the KWS model specifically for the Monocle's microphone and noise pattern. This involves training the model on a dataset that captures the unique acoustic characteristics of the Monocle environment, improving its performance in real-world scenarios.

Platform Porting Adapt the KWS model to run directly on the Monocle device, moving away from the development kit. This requires the model to be optimized for the target hardware and takes advantage of the Monocle's specific capabilities in order to perform real-time keyword spotting based on the audio input from the device's built-in microphone.

FPGA Integration Investigate the feasibility and benefits of utilizing the FPGA on the Monocle instead of the Bluetooth MCU for KWS processing. Evaluate how FPGA acceleration can enhance the model's performance and overall efficiency.

Tensorflow with MicroPython Explore the possibility of configuring TensorFlow to work directly with MicroPython on the Monocle device. This could streamline the deployment process and reduce dependencies, allowing for a more seamless integration of the KWS model into the Monocle ecosystem.

Benchmarking and Optimization Conduct benchmarking tests to assess the model's inference speed and resource utilization on the Monocle. Optimize the model and its implementation to ensure efficient use of the device's limited resources while maintaining acceptable performance.

User Interface Integration Integrate the KWS functionality into the Monocle user interface, providing a user-friendly experience for configuring, activating, and interacting with the keyword spotting feature.

Community Engagement Foster community engagement by sharing updates, seeking feedback, and encouraging contributions from the Brilliant Labs Monocle user community. Collaborate with developers and users to gather insights and improve the KWS model continuously.

6 Contributions

Aghyad Deeb was responsible for understanding the makefiles and getting the compilation to work. He helped find the crucial addition necessary to allow the Makefile to properly compile TFLM and the KWS model. In doing so, he had to explore the overall structure of the codebase and learned about the ins and outs of Makefiles.

Emeka Ezike was responsible for accessing audio data from the monocle. He researched and implemented methods to convert the audio bytes stored on the FPGA into .wav files and spectrogram bytes compatible with the KWS search model using the limited MicroPython modules. Furthermore, Emeka helped research the existing MicroPython KWS implementations and helped identify key issues and rule out false solutions.

Jayson Lin was responsible for exploring the overall structure of Harrison's codebase. He helped identify the functions of numerous header files, C files, MicroPython files, and Makefiles. Furthermore, he contributed to troubleshooting the nRF52DK board, so that applicable steps could be written for flashing the firmware to the nRF52DK board.

Acknowledgments

We express our sincere gratitude to the teaching fellows for their support and guidance, and specifically Matthew Stewart for connecting us with knowledgeable individuals.

Special thanks to Harrison Zhang for his insightful TFLM Micropython work and detailed explanations that significantly contributed to the success of our endeavors. His generosity in sharing knowledge played a crucial role in our project.

Lastly, we extend our gratitude to Professor Vijay Reddi for being an exceptional instructor. Professor Reddi lead a learning environment that fostered curiosity, critical thinking, and a deep understanding of the subject matter. His dedication to education has been an inspiration throughout our project.

All group members have collectively agreed to allow the publication of our final report video on YouTube by the course staff.

References

ARM Community. Fast and Accurate Keyword Spotting Using Transformers. https://community.arm.com/arm-research/b/articles/posts/fast-and-accurate.keyword-spotting-using-transformers, 2022. Accessed: 2023-12-10.

Brilliant-Labs. Brilliant Labs Documentation. https://docs.brilliant.xyz/, a. Accessed: 2023-12-10.

Brilliant-Labs. Brilliant Labs Monocle Product Page. https://brilliant.xyz/products/monocle, b. Accessed: 2023-12-10.

Cerutti, G., Cavigelli, L., Andri, R., Magno, M., Farella, E., and Benini, L. Sub-mw keyword spotting on an mcu: Analog binary feature extraction and binary neural networks. 2022.

Labs, B. Micropython ported to the monocle. https://github.com/brilliantlabsAR/monocle-micropython, 2023.

Rohlicek, J., Russell, W., Roukos, S., and Gish, H. Continuous hidden markov modeling for speaker-independent word spotting. In *Proceedings of the 14th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pp. 627–630, 1989.

Tang, R., Wang, W., Tu, Z., and Lin, J. An experimental analysis of the power consumption of convolutional neural networks for keyword spotting, 2018.

Wei, B., Yang, M., Zhang, T., Tang, X., Huang, X., Kim, K., Lee, J., Cho, K., and Park, S.-U. End-to-end transformer-based open-vocabulary keyword spotting with location-guided local attention. In *Interspeech 2021*, pp. 361–365, 2021. doi: 10.21437/Interspeech.2021-1335. URL https://www.interspeech2021.org/.

Zhang, Y., Suda, N., Lai, L., and Chandra, V. Hello edge: Keyword spotting on microcontrollers, 2018.