Mbed Torch Fusion OS: running a PyTorch model on Mbed OS platforms

Christoph Karl Heck, Till Aust, Heiko Hamann
christoph.heck@uni-konstanz.de
Master's Project Cyber-Physical Systems (WS 2023/24)
Cyber-Physical Systems Group
Department of Computer and Information Science, University of Konstanz

May 3, 2024

Abstract

In our EU-funded project WatchPlant, we are developing a sensor network of bio-hybrid sensor nodes. In order to save energy and provide live information, we require onboard processing. To enable the use of PyTorch deep learning models for onboard processing on our bio-hybrid sensor nodes (PhytoNodes), we present Mbed Torch Fusion OS, a framework for running PyTorch models on Mbed OS platforms. Addressing a long-standing challenge, we provide a practical and convenient way to deploy PyTorch models on Mbed OS platforms with Mbed Torch Fusion OS. In this work, we present different approaches and explain their limitations. Furthermore, we show and test the integration of a PyTorch softmax model into Mbed OS using the P-Nucleo-WB55RG hardware and ExecuTorch. Moreover, we demonstrate the adaptability of our solution, showing how it can be easily applied to other Mbed OS platforms, enhancing its versatility and potential for widespread use.
Code available at: `https://github.com/ChristophKarlHeck/mbed-torch-fusion-os`
Tutorial available at: `https://youtu.be/jc_GCxAAO20`

1 Introduction

In our EU-funded project WatchPlant Hamann et al. [2021], we propose a green approach for urban air quality monitoring using a network of bio-hybrid sensor nodes. These bio-hybrid sensor nodes consist of a PhytoNode Buss et al. [2022] that interacts with a plant (e.g., *Hedera helix*, also called ivy) by measuring the plant's electrophysiology. From these measurements, we can infer the plant's environmental state Buss et al. [2023], Chatterjee et al. [2015] and thus estimate the air quality. The goal is to deploy many bio-hybrid sensor nodes in urban areas to get an excellent spatial resolution of air quality measurements. The PhytoNodes should provide air quality information to edge devices of nearby citizens and central servers for policymakers. The PhytoNodes should be energy autonomous, which we can reach by energy harvesting methods. However,

this constrains the available energy and makes transferring raw data impractical [Abadade et al., 2023]. Therefore, we must have onboard processing of the raw plant physiological data.
Given the success of deep learning, we propose to use the deep learning library PyTorch [Paszke et al., 2019] for developing and deployment on our Mbed OS based PhytoNodes. To the best of our knowledge, there is no existing work on deploying PyTorch models on Mbed OS platforms shown on website [Mbe, d]. Hence, in this work, we propose the framework Mbed Torch Fusion OS to bridge this gap. Additionally, we use open-source tools that further application software, such as the I/O handling, is easy to integrate.

2 Related work

This section presents other possible approaches on how to run PyTorch deep learning models on Mbed OS platforms and their limitations. Thereby, we focus on required software dependencies and maintenance activity.

Open Neural Network Exchange

Open Neural Network Exchange (ONNX) is the open standard that allows users to easily convert their model into an equivalent model running in another framework, for example, converting a PyTorch model into an ONNX standard model and then the ONNX standard model into a TensorFlow [Abadi et al., 2016] model. Shah [2017] shows that Facebook, Microsoft, IBM, Huawei, Intel, AMD, Arm, and Qualcomm support the ONNX standard, and the GitHub release history Onn [a] displays continuous development. We analyze the ONNX model as a directed acyclic graph using Netron [Roeder, 2017]. The website Onn [b] shows that ONNX allows for quantization, which maps floating point values to an 8-bit quantization space form.

Open Neural Network Compiler

Lin et al. [2019] proposes the `Open Neural Network Compiler` (ONNC) that transforms a model represented in ONNX standard to a model inference `.cpp` file. We merge the application software with the model inference file and the CMSIS-NN library from Lai et al. [2018]. The CMSIS-NN software library consists of efficient neural network kernels designed to optimize performance and minimize memory usage on Arm Cortex-M processors. Hence, we use the ARM toolchain to build the binary that the μ-controller can execute. Figure 1 shows the concept of ONNC.

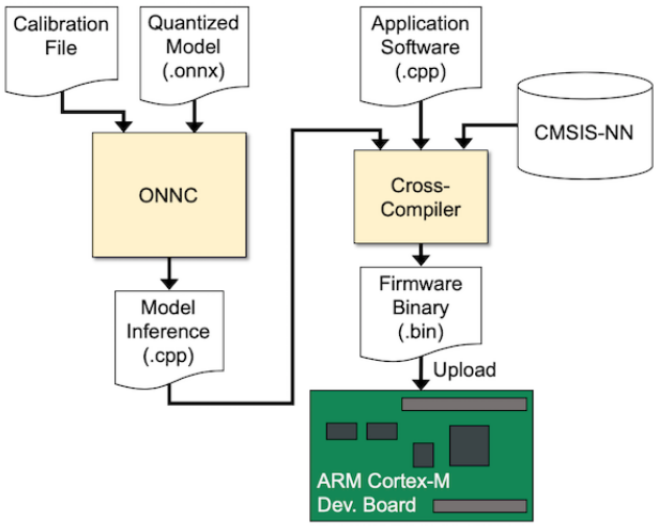

Figure 1: The transformation process of ONNC proposed by GitHub user cthsieh [2019].

We start going through the tutorial by GitHub user cthsieh [2019] Moreover, we stopped when we saw that the ONNC docker image provided by docker hub user chengchiwang [2020] is more than four years old, and the last release was on April 30, 2020. Additionally, cthsieh [2019] describes the usage of a calibration file without reference. Due to the lack of active development and missing documentation, ONNC is not a promising solution for integrating a PyTorch model into Mbed OS.

NN4MC

Aguasvivas Manzano et al. [2019] propose the neural networks for μ-controllers (NN4MC) transformation process. The GitHub users Aguasvivas and Simpson [2020] provide a NN4MC Python and Aguasvivas et al. [2020] a C++ implementation.

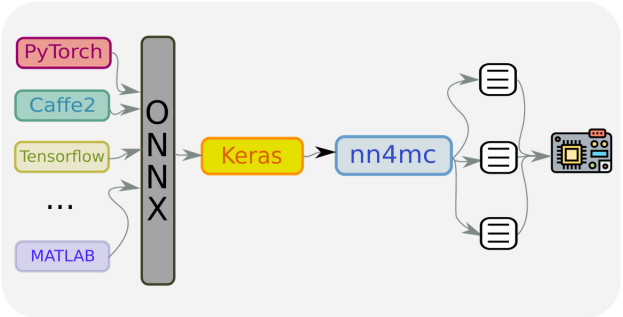

Figure 2: The transformation process of NN4MC proposed by Aguasvivas and Simpson [2020].

The process uses ONNX as an exchange format. However, up to this point, it does not support PyTorch models, and we must follow the same approach as using the ONNC, except for converting our model to a Keras model. Chollet et al. [2015] propose Keras as a Python deep learning API running on top of either PyTorch, JAX, or TensorFlow. Additionally, the repositories of NN4MC have not been maintained since February 2022, and Alkhouski [2020] explains that the conversion from ONNX to Keras does not support LSTM layers. Therefore, we do not use NN4MC.

TensorFlow Lite

The Mbed OS tutorial [TFL] shows an example of running the TensorFlow Lite model in Mbed OS. We followed the instructions in the tutorial and encountered errors because TensorFlow has changed its targets and no longer contains the `mbed` folder. The TAGS option is also no longer supported by the TFLM Makefile, and the Mbed OS tutorial does not mention the version (release) of TensorFlow they use. Given the outdated and non-functional nature of the tutorial, we decided to explore an alternative approach. We opted against converting our PyTorch model into a TensorFlow model using ONNX and then further converting it into a TensorFlow Lite model, as this method would have introduced a significant number of additional software dependencies. Furthermore, this approach would require much more software dependencies.

Conclusion

To the best of our knowledge, there is a gap, as there is no straightforward way of utilizing PyTorch models on Mbed OS capable μ-controllers. The presented approaches are either outdated or have lots of software dependencies, for example, converting the model multiple times into different frameworks, which make them impractical for development and deployment.

3 Methods

In this section we present the used tools. We begin by describing the hardware, followed by introducing the used software frameworks ExecuTorch and Mbed OS.

3.1 Hardware

The PhytoNode builds up on an STM32WB55 microcontroller. For development, we use the P-Nucleo-WB55RG development platform, which features a dual-core, multi-protocol wireless STM32WB55 microcontroller. It incorporates an ARM Cortex-M4 core with Digital Signal Processing (DSP) extension dedicated to applications and a Cortex M0+ responsible for managing the radio layer, adapting to various RF protocols such as Bluetooth Low Energy (BLE), Thread, and Zigbee. The P-Nucleo-WB55RG has 1 Mbyte of flash memory and 256 Kbyte of static random access memory (SRAM). The Mbed OS platform description [Des] contains the Nucleo pinout and further hardware details. The target name for our hardware on the host OS is `NUCLEO_WB55RG`.

3.2 ExecuTorch

We use ExecuTorch to directly use PyTorch models on a higher level, avoiding converting our models multiple times through different frameworks, as described in Section 2. At this point in time, the support, active development, and community of ExecuTorch seems promising. The number of citations from Paszke et al. [2019] underscores that PyTorch is a popular deep-learning framework in research.
ExecuTorch is a complete solution that allows running inference directly on mobile and edge devices, such as wearables, embedded devices, and microcontrollers, similar to our hardware. It is a component of the PyTorch Edge ecosystem, ensuring the deployment of PyTorch models to these devices. The ExecuTorch GitHub page [Exe, b] describes it as lightweight runtime optimized for utilizing full hardware capabilities, including CPUs, neural processing units (NPUs), and DSPs. The documentation [Exe, a] provides an extensive overview of ExecuTorch.

3.3 Mbed OS

Smith [2022] describes that the original Mbed OS [Arm, b] project from ARM has been paused since the beginning of 2022. Therefore, we use the Mbed OS fork [Mbe, c] from the Mbed Community, namely, the Mbed Community Edition (CE) along the project. Due to the reason that we use the Mbed OS community edition, we are free of ARM-specific configuration. We can use open-source tools like VS Code and CMake to realize the project instead of using limited environments like Mbed Studio, Mbed CLI, and Keil Studio Cloud. We must include our precompiled ExecuTorch libraries, which the ARM developer tools do not support. Mbed OS is an open-source embedded operating system designed specifically for Arm Cortex-M microcontroller, including security, connectivity, a real-time operating system (RTOS), and sensors and I/O drivers. Simmonds [2017] shows that the Mbed OS uses the toolchain to build our hardware's bootloader, kernel, and root filesystem. We use the arm toolchain throughout the project and the default complete profile of Mbed OS, not the bare metal profile. One key concept is that Mbed OS redefines multiple standard C library functions to enable them to work predictably and familiarly on a remote embedded target device. The Mbed OS documentation [Mbe, b] of ARM provides essential concepts and further information about Mbed OS.

4 Implementation and results

This section describes the implementation approach using an Ubuntu 22.04 `x86_64` host system and the P-Nucelo-WB55RG as the target system. Section 3.1 describes the details of P-Nucleo-WB55RG hardware. Here, we focus on the integration of ExecuTorch [Our, c] into Mbed OS [Our, d] using CMake [Our, b] as build system because both ExecuTorch and Mbed OS provide CMakeLists.txt files by default. We use the ARM toolchain [Our, a] to cross-compile the software for the Cortex-M4 architecture because the official Debian/Ubuntu toolchain has an unfixed bug reported by Teuwen [2020] that causes issues during Mbed OS compilation. We shortcut the procedure of exporting a trained PyTorch model into a .pte-file by using a provided script [Aot]

of ExecuTorch to generate a `.pte` file of a simple model that applies a softmax operation to an input tensor.

4.1 Software Versions

In Section 3.2, we explain why we chose ExecuTorch as our framework. However, the preview release of ExecuTorch should only be used for test purposes because the ExecuTorch code and APIs change quickly. ExecuTorch in the first stable pre-release (v0.1.0) [Exe, c] does not provide the `arm_executor_runner.cpp` file and its corresponding `CMakeLists.txt` file. Hence, we work with the current main branch of ExecuTorch and freeze the commits/versions of ExecuTorch (ceb1f1d), Mbed OS (862f462), and CMake (v3.28.3). The commit/version freeze ensures that the patches we implement are correct and always lead to a reproducible solution.

4.2 Project Structure

The underlying design concept is to make it easy to adapt the integration of ExecuTorch into Mbed OS to other supported μ-controllers by Mbed OS besides the P-Nucleo-WB55. Therefore, we integrate the code base of ExecuTorch and Mbed OS as git submodules into our code base and change as little as possible in ExecuTorch and Mbed OS by patches. To adapt the solution to other μ-controllers, we must patch the `arm-none-eabi-gcc.cmake` file to the corresponding architecture and change the target in the `cmake-variants.yaml` file.

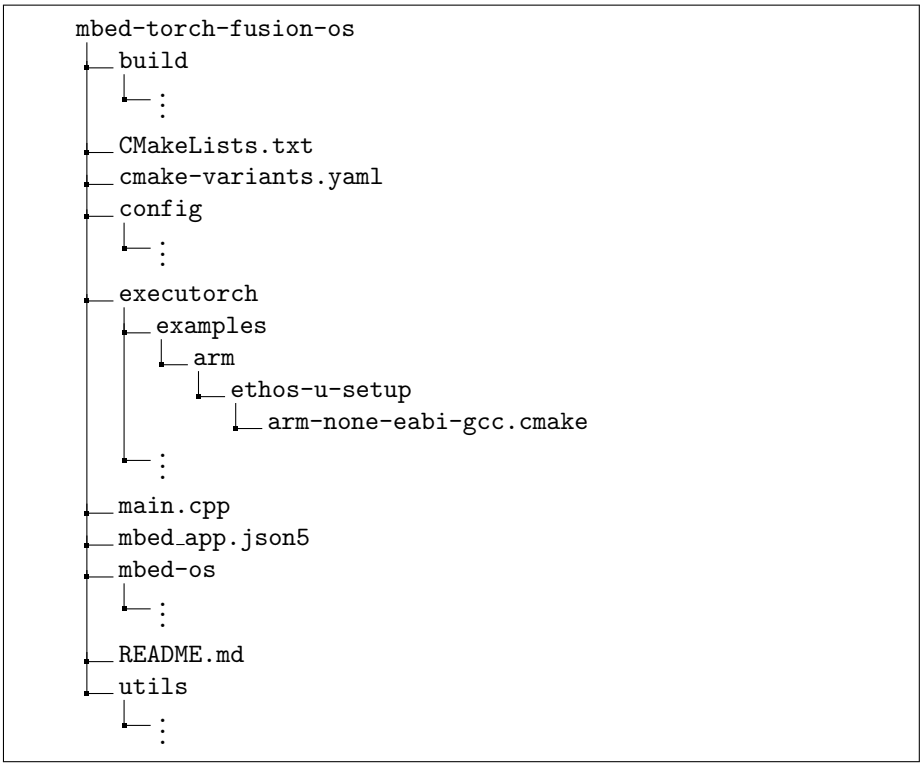

Figure 3: Code structure of Mbed Torch Fusion OS.

CMake uses the `CMakeLists.txt` file to generate the `build` directory corresponding to the target's architecture. We use Ninja in version [Nin] to build the Mbed OS binary and link the precompiled ExecuTorch libraries to it. Hence, we flash the binary onto the target using Ninja and a previously chosen upload method. The `cmake-variants.yaml` file defines variants of build types like develop, debug, and release, as well as the target and the upload method. We use pyOCD [PyO] as the upload method because it supports debugging on Arm Cortex-M μ-controllers and Upl shows that it works with almost all Mbed boards. The `config` directory contains all additional config files needed to integrate ExecuTorch into Mbed OS, for instance, the `NUCLEO_WB55RG.cmake` config file to compile the fusion for our hardware. The `utils` directory contains the scripts and patches to set up ExecuTorch and Mbed OS. The `main.cpp` is an adjusted program of the `arm_executor_runner.cpp` [Arm, a] provided by ExecuTorch. The `mbed_app.json5` is part of the configuration system of the git submodule `mbed-os` [Mbe, a] and `executorch` represents the ExecuTorch repository as a second git submodule.

4.3 ExecuTorch Libraries and Debugging Challenges

We link the following precompiled ExecutTorch libraries into the executable of Mbed OS during the build process to run a PyTorch model on the target:

- libexecutorch.a

- libportable_kernels.a
- libarm_portable_ops_lib.a
- libextension_runner_util.a

The `libexecutorch.a` library provides the runtime environment of ExecuTorch. The `libportable_kernels.a` library represents the kernels in the neural network, and the `libarm_portable_ops_lib.a` library defines the operations the neural network executes on our target device. In our example, `libarm_portable_ops_lib.a` only contains the softmax operation. Because we want to save as much space as possible on the target device, we need to define a specific configuration file of the operations we include to execute the model in Mbed OS. When exporting a PyTorch model to a `, we must analyze the operations we need beforehand.pte` file to avoid operations on our target we do not need. If we want to use additional operations, we must add them to `custom_ops.yaml` file [Cus].
We encountered bugs during the implementation, and debugging the entire Mbed Torch Fusion OS using the libraries does not work because they contain the precompiled object files. However, we found a solution. Instead of using the libraries themselves, we utilize the source files of the libraries in the build procedure. Hence, instead of the libraries, we include the paths to the source files in the `CMakeLists.txt` file. The `utils` directory includes a suitable script called `find_cpp_files_from_lib.sh` that prints all source files in the corresponding library. In the first release of Mbed Torch Fusion OS, the source files corresponding to the versions mentioned in Section 4.1 are already included as commented statements in the `CMakeLists.txt`.

4.4 Realization

We first compile the necessary libraries of ExecuTorch mentioned in chapter 4.3. Here, we suit the ExeuTorch CMake configuration file to our hardware, fix the bugs that ExecuTorch still has in our version, and provide suitable patches to fix them automatically. Hence, we integrate the necessary configuration files with PyOCD and OpenOCD as upload methods into Mbed OS. Finally, we test our fusion of ExecuTorch and Mbed OS. The appendix 5 shows our concrete description of how we merged ExecuTorch and Mbed OS, what errors occurred, and how we solved them with our patches.

5 Conclusion

This work presents a straightforward solution to run PyTorch models on Mbed OS-supported hardware. We merge Mbed OS and ExecuTorch in a memory-efficient manner and run a PyTorch softmax model on the P-Nucleo-WB55RG hardware. Additionally, we describe how our solution can be easily adapted to other targets.
The following steps include extending the framework to run arbitrary PyTorch models on P-Nucleo-WB55RG hardware. Therefore, we will integrate the process of exporting a PyTorch model to ExecuTorch. The ExecuTorch Tutorial [Exe, d] explains the process and its options. We aim to automate this

process as much as possible. Further open challenges include automating optimization steps, such as automatically only deploying used operations, model precision, or model pruning.
Finally, we intend only to deploy pre-trained models because training only on limited microcontroller hardware seems unreasonable.

Acknowledgement

We thank Jamie Smith and the community of Mbed CE for maintaining and developing Mbed OS that allows us to include foreign libraries from ExecuTorch and makes it possible to debug Mbed Torch Fusion OS using VS Code. This work was supported by European Union's Horizon 2020 FET research program under grant agreement No. 101017899 (WatchPlant).

References

Aot arm compiler. https://github.com/pytorch/executorch/blob/ceb1f1d05ceab420644a3633264f13547dc02411/examples/arm/aot_arm_compiler.py. Accessed: 03-16-2024.

Arm executorchrunner. https://github.com/pytorch/executorch/blob/ceb1f1d05ceab420644a3633264f13547dc02411/examples/arm/executor_runner/arm_executor_runner.cpp, a. Accessed: 03-16-2024.

Arm mbed os. https://github.com/ARMmbed/mbed-os, b. Accessed: 03-16-2024.

Custom yaml configuration. https://github.com/pytorch/executorch/blob/ceb1f1d05ceab420644a3633264f13547dc02411/kernels/portable/custom_ops.yaml. Accessed: 03-16-2024.

St-nucleo-wb55rg description. https://os.mbed.com/platforms/ST-Nucleo-WB55RG/. Accessed: 03-1-2024.

Executorch documentation. https://github.com/pytorch/executorch, a. Accessed: 03-16-2024.

Executorch github. https://github.com/pytorch/executorch, b. Accessed: 03-16-2024.

Executorch does not provide. https://github.com/pytorch/executorch/tree/v0.1.0/examples/arm, c. Accessed: 03-16-2024.

Executorch tutorial. https://pytorch.org/executorch/main/tutorials/export-to-executorch-tutorial.html, d. Accessed: 03-16-2024.

Mbed os configuration. https://os.mbed.com/docs/mbed-os/v6.16/program-setup/advanced-configuration.html, a. Accessed: 03-16-2024.

Mbed os documentation. https://os.mbed.com/docs/mbed-os/v6.16/introduction/index.html, b. Accessed: 03-16-2024.

Arm mbed os fork. https://github.com/mbed-ce/mbed-os, c. Accessed: 03-16-2024.

Mbed Platforms. https://os.mbed.com/platforms/, d. Accessed: 03-21-2024.

Ninja github. https://github.com/ninja-build/ninja/tree/a1f879b29c9aafe6a2bc0ba885701f8f4f19f772. Accessed: 03-16-2024.

Onnx releases. https://github.com/onnx/onnx/releases, a. Accessed: 03-21-2024.

Onnx quantization. https://onnxruntime.ai/docs/performance/model-optimizations/quantization.html, b. Accessed: 03-21-2024.

Our arm toolchain. https://developer.arm.com/-/media/Files/downloads/gnu-rm/10.3-2021.10/gcc-arm-none-eabi-10.3-2021.10-x86_64-linux.tar.bz2, a. Accessed: 03-16-2024.

Our cmake version. https://github.com/Kitware/CMake/tree/5e984bb35232116a54de7db39788cb162ca7c263, b. Accessed: 03-16-2024.

Our executorch version. https://github.com/pytorch/executorch/tree/ceb1f1d05ceab420644a3633264f13547dc02411, c. Accessed: 03-16-2024.

Our mbed os version. https://github.com/mbed-ce/mbed-os/tree/862f46233c3d074048b8f4003aceea4b0d254694, d. Accessed: 03-16-2024.

Pyocd github. https://github.com/pyocd/pyOCD/tree/975fe95ed8978278485c31cbb1f4b62ab1689237. Accessed: 03-16-2024.

Tensorflow lite mbed os tutorial. https://os.mbed.com/docs/mbed-os/v6.16/mbed-os-pelion/machine-learning-with-tensorflow-and-mbed-os.html. Accessed: 02-26-2024.

Upload methods. https://github.com/mbed-ce/mbed-os/wiki/Upload-Methods. Accessed: 03-16-2024.

Youssef Abadade, Anas Temouden, Hatim Bamoumen, Nabil Benamar, Yousra Chtouki, and Abdelhakim Senhaji Hafid. A comprehensive survey on TinyML. *IEEE Access*, 11:96892–96922, 2023. doi: 10.1109/ACCESS.2023.3294111.

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

Sarah Aguasvivas and Cooper Simpson. Nn4mc python implementation. https://github.com/correlllab/nn4mc_py, 2020. Accessed: 03-16-2024.

Sarah Aguasvivas, Cooper Simpson, and Dana Hughes. Nn4mc c++ implementation. https://github.com/correlllab/nn4mc_cpp, 2020. Accessed: 03-13-2024.

Sarah Aguasvivas Manzano, Dana Hughes, Cooper Simpson, Radhen Patel, and Nikolaus Correll. Embedded neural networks for robot autonomy, 2019. International Symposium of Robotics Research (ISRR 2019).

Andrei Alkhouski. Notimplementederror: Can't modify this type of data. https://github.com/gmalivenko/onnx2keras/issues/94, 2020. Accessed: 03-5-2024.

Eduard Buss, Tim-Lucas Rabbel, Viktor Horvat, Marko Krizmancic, Stjepan Bogdan, Mostafa Wahby, and Heiko Hamann. PhytoNodes for environmental monitoring: Stimulus classification based on natural plant signals in an interactive energy-efficient bio-hybrid system. In *Proceedings of the 2022 ACM Conference on Information Technology for Social Good*, GoodIT'21, pages 258–264, 2022. doi: 10.1145/3524458.3547266.

Eduard Buss, Till Aust, Mostafa Wahby, Tim-Lucas Rabbel, Serge Kernbach, and Heiko Hamann. Stimulus classification with electrical potential and impedance of living plants: comparing discriminant analysis and deep-learning methods. *Bioinspiration & Biomimetics*, 18(2):025003, 2023. doi: 10.1088/1748-3190/acbad2.

Shre Kumar Chatterjee, Saptarshi Das, Koushik Maharatna, Elisa Masi, Luisa Santopolo, Stefano Mancuso, and Andrea Vitaletti. Exploring strategies for classification of external stimuli using statistical features of the plant electrical response. *Journal of the Royal Society, Interface*, 12(104):20141225, 2015. doi: 10.1098/rsif.2014.1225.

chengchiwang. Onnc docker. https://hub.docker.com/r/onnc/onnc-community, 2020. Accessed: 03-16-2024.

François Chollet et al. Keras. https://keras.io, 2015.

cthsieh. Onnc tutorial. https://github.com/ONNC/onnc-tutorial/blob/master/lab_2_Digit_Recognition_with_ARM_CortexM/lab_2.md, 2019. Accessed: 03-16-2024.

Heiko Hamann, Stjepan Bogdan, Antonio Diaz-Espejo, Laura García-Carmona, Virginia Hernandez-Santana, Serge Kernbach, Andreas Kernbach, Alfredo Quijano-López, Babak Salamat, and Mostafa Wahby. WatchPlant: Networked bio-hybrid systems for pollution monitoring of urban areas. volume ALIFE 2021: The 2021 Conference on Artificial Life, pages 37–45, 2021. doi: 10.1162/isal_a_00377.

Liangzhen Lai, Naveen Suda, and Vikas Chandra. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *arXiv preprint arXiv:1801.06601*, 2018.

Wei-Fen Lin, Der-Yu Tsai, Luba Tang, Cheng-Tao Hsieh, Cheng-Yi Chou, Ping-Hao Chang, and Luis Hsu. Onnc: A compilation framework connecting onnx to proprietary deep learning accelerators. In *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 214–218, 2019. doi: 10.1109/AICAS.2019.8771510.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

Lutz Roeder. Netron, visualizer for neural network, deep learning, and machine learning models. Zenodo. http://doi.org/10.5281/ZENODO.7087890, 2017. Accessed: 03-21-2024.

Saqib Shah. Onnx supporters. https://www.engadget.com/2017-10-11-microsoft-facebooks-ai-onxx-partners.html?guce_referrer=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnLw&guce_referrer_sig=AQAAABryXn0SZ-1Hvtguzse5_ibM0Tj2NaVI5wuoE8sOZKaf4EVXWw-finaVqn7MVyfj2uwh7_8gGlnwEYC7QDdJ0Vi3EZrnlNLiZ3krzyigaPY6wsdwndoWQ_B-8hoh9X61fZ0B3_cu1UYkiVOeoppD36Vlw66Ap62VAfVN2B5VMOpj, 2017. Accessed: 03-21-2024.

Chris Simmonds. *Mastering Embedded Linux Programming*. Packt Publishing Ltd., Birmingham, UK, 2017.

Jamie Smith. Mbed ce: A community fork of mbed os. https://forums.mbed.com/t/help-wanted-mbed-ce-a-community-fork-of-mbed-os/17998, 2022. Accessed: 03-5-2024.

Philippe Teuwen. Toolchain bug. https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=953844, 2020. Accessed: 03-16-2024.

Appendix

ExecuTorch setup, bug fixes, and obligatory patches

To precompile the mentioned libraries from above, we set up ExecuTorch with the corresponding script `build_et_libs.sh`. The script sets all third-party software to the versions mentioned in Section 4.1. Next, it creates its own Python environment to avoid version conflicts with the host system and installs the requirements[1] of ExeuTorch.

```
Preparing metadata (pyproject.toml) ... error
error: subprocess-exited-with-error
Preparing metadata (pyproject.toml) did not run
successfully.
exit code: 1
```

[1] https://github.com/pytorch/executorch/blob/ceb1f1d05ceab420644a3633264f13547dc02411/install_requirements.sh

```
note: This error originates from a subprocess,
and is likely not a problem with pip.
error: metadata-generation-failed
```

Listing 1: Error while installing the requirements of ExecuTorch.

Installing the requirements from ExecuTorch causes the error in Listing 1. We solved the error by pre-installing specific dependencies[2] and suited the solution of the GitHub user `admin0987iu` to our use case by specifying the versions of the dependencies. Listing 2 shows our refined solution.

```
pip3 cache purge
pip3 install meson==1.3.2
python3 -m pip install --upgrade pip
pip3 install importlib-metadata==7.0.2
python3 -m pip install setuptools==69.1.1
pip3 install subprocess.run==0.0.8
pip3 install wheel==0.42.0 setuptools pip --upgrade
pip3 install toml==0.10.2
pip3 install numpy==1.26.4 \
--use-deprecated=legacy-resolver
pip3 install setuptools-metadata==0.1.5
```

Listing 2: Solution to install the requirements of ExecuTorch succesfully.

ExecuTorch uses FlatBuffers as a serialization library because it focuses on optimal memory efficiency across various platforms, and its architecture enables direct access to serialized data without the need for initial parsing or unpacking. Therefore, we build the FlatBuffers compiler called `flatc` using FlatBuffers (v24.3.7)[3] as cross-platform serialization library to export the model into a .pte file.

```
WARNING: An incompatible version of flatc is on the PATH
at executorch/third-party/flatbuffers/cmake-out/flatc.
  Required version: flatc version 24.3.25
  Actual version: flatc version 24.3.7
```

Listing 3: FlatBuffers causes incompatible version warnings.

Due to the reason that we stick to version 24.3.7 and FlatBuffers continue developing, we get a version mismatch warning shown in Listing 3. To get rid of this warning, we patch line 30 of the script `install_flatc.sh`[4] as shown in Listing 4.

```
- git describe --tags \
- "$(git rev-list --tags --max-count=1)" | sed 's/^v//'
+ git describe --tags | sed 's/^v//'
```

Listing 4: Our `install_flatc.patch` solves version mismatch warning.

The patch removes the warning because the new command in Listing 4 describes the latest commit associated with a tag without explicitly fetching it. So, the new git command uses the local version and does not fetch the latest version of FlatBuffers.
In the next step we export the PyTorch softmax model to a .pte file model using `flatc` and the `aot_arm_compiler`[5] script. Therefore we add `flatc` to `PATH` and patch the `aot_arm_compiler` script to avoid the error in Listing 5 using our `aot_arm_compiler.patch` file.

```
[ERROR utils.py:110]
Error while saving to softmax.pte: 'bytes'
object has no attribute 'write_to_file'
```

Listing 5: Error in executorch/examples/arm/aot_arm_compiler.py file

[2] https://github.com/zylon-ai/private-gpt/issues/942
[3] https://github.com/google/flatbuffers/tree/6ff9e90e7e399f3977e99a315856b57c8afe5b4d
[4] https://github.com/pytorch/executorch/blob/ceb1f1d05ceab420644a3633264f13547dc02411/build/install_flatc.sh
[5] https://github.com/pytorch/executorch/blob/ceb1f1d05ceab420644a3633264f13547dc02411/examples/arm/aot_arm_compiler.py

The ExecuTorch code tries to call the `write_to_file` method on a property of an object `exec_prog.buffer`. This is wrong because the method `write_to_file` is a member of the `exec_prog` object. Therefore we change the line as shown in Listing 6.

```
- save_pte_program(exec_prog.buffer, model_name)
+ save_pte_program(exec_prog, model_name)
```

Listing 6: Our `aot_arm_compiler.patch` file enables the creation of `softmax.pte` file.

Next, we change the `arm-none-eabi-gcc.cmake` file to enable cross-compilation for Arm Cortex M4 architecture. To automate the procedure, we create a `cmake_file_m4.patch` file that adjusts the CMake configuration for our hardware. We must adjust the configuration file to the new target if we use another *mu*-controller with another architecture. Listing 7 shows the configuration for P-Nucleo-WB55RG hardware.

```
set(TARGET_CPU "cortex-m4" CACHE STRING "Target CPU")
string(TOLOWER ${TARGET_CPU} CMAKE_SYSTEM_PROCESSOR)

set(CMAKE_SYSTEM_NAME Generic)
set(CMAKE_C_COMPILER "arm-none-eabi-gcc")
set(CMAKE_CXX_COMPILER "arm-none-eabi-g++")
set(CMAKE_ASM_COMPILER "arm-none-eabi-gcc")
set(CMAKE_LINKER "arm-none-eabi-ld")

set(CMAKE_EXECUTABLE_SUFFIX ".elf")
set(CMAKE_TRY_COMPILE_TARGET_TYPE STATIC_LIBRARY)
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)

set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 14)

set(GCC_CPU ${CMAKE_SYSTEM_PROCESSOR})
string(REPLACE "cortex-m85" "cortex-m4" GCC_CPU ${GCC_CPU})

add_compile_options(
    "-mthumb"
    "-mcpu=cortex-m4"
    "-mfpu=fpv4-sp-d16"
    "-mfloat-abi=softfp")

add_compile_definitions(
    "$<$<NOT:$<CONFIG:DEBUG>>:NDEBUG>")

add_link_options(
    "-Wl,--start-group"
        "-lstdc++"
        "-lsupc++"
        "-lm"
        "-lc"
        "-lgcc"
        "-lnosys"
    "-Wl,--end-group"
    "-specs=nosys.specs"
    "-Wl,--cref")

# Compilation warnings
add_compile_options(
    -Wno-psabi./7
)
```

Listing 7: Our patched `arm-none-eabi-gcc.cmake` file for P-Nucleo-WB55RG.

For the Cross-compilation itself, we make the toolchain available in `PATH` and extract the `Buck2` binary into the `/tmp` folder. ExecuTorch uses Buck2 as the build system for the required libraries. Here, we use the Buck2 release[6] from 1st March 2024.
In the next step we configure our project and generate the native build system with the command shown in Listing 8.

```
cmake                                                    \
    -DBUCK2=/tmp/buck2                                   \
    -DCMAKE_INSTALL_PREFIX=$(pwd)/cmake-out              \
    -DEXECUTORCH_BUILD_EXECUTOR_RUNNER=OFF               \
    -DCMAKE_BUILD_TYPE=Release                           \
    -DEXECUTORCH_ENABLE_LOGGING=ON                       \
    -DEXECUTORCH_BUILD_ARM_BAREMETAL=ON                  \
    -DEXECUTORCH_BUILD_EXTENSION_RUNNER_UTIL=ON          \
    -DFLATC_EXECUTABLE="$(which flatc)"                  \
    -DCMAKE_TOOLCHAIN_FILE=$(pwd)/examples
    /arm/ethos-u-setup/arm-none-eabi-gcc.cmake           \
    -B$(pwd)/cmake-out                                   \
    $(pwd)
```

Listing 8: Configuration for libexecutorch.a, libportable_kernels.a and libextension_runner_util.a and generation of a native build for a system for P-Nucleo-WB55RG.

However, the command fails with error message exhibited in listing 9.

```
Command failed:
Unknown target `generated_lib_all_ops`
from package `root//kernels/portable`.
```

Listing 9: Error message while generating nativ build system.

We solve the error by applying our `targets.patch` file to the `targets.bzl`[7] file.

```
    executorch_generated_lib(
+       name = "generated_lib_all_ops",
+       deps = [
+           ":executorch_all_ops",
+           "//executorch/kernels/portable:operators",
+       ],
+       **generated_lib_common_args
+   )
+
+   executorch_generated_lib(
        name = "generated_lib_aten",
        deps = [
            ":executorch_aten_ops",
@@ -120,3 +129,4 @@
        ],
        define_static_targets = True,
    )
+
```

Listing 10: Our `targets.patch` file to make build of required libraries possible.

The patch (Listing 10) removes the first instance of `executorch_generated_lib` and makes sure that the configuration uses only our defined custom operations and ATen operations. The most C++ interfaces of PyTorch are built on the tensor library ATen.[8] Therefore, we need the ExecuTorch ATen operations.

[6] https://github.com/facebook/buck2/releases/download/2024-03-01/buck2-x86_64-unknown-linux-gnu.zst

[7] https://github.com/pytorch/executorch/blob/ceb1f1d05ceab420644a3633264f13547dc02411/kernels/portable/targets.bzl

[8] https://pytorch.org/executorch/stable/concepts.html

```
cmake --build $(pwd)/cmake-out -j4 --target          \
install --config Release
```

Listing 11: Building libexecutorch.a, libportable_kernels.a and libextension_runner_util.a libraries for P-Nucleo-WB55RG.

We build the following libraries `libexecutorch.a`, `libportable_kernels.a`, and `libextension_runner_util.a` with the command shown in Listing 11 using four cores of the host machine. In the last step, we cross-compile the library `libarm_portable_ops_lib.a` containing the softmax operation for arm architectures using the setup and build command in Listing 12.

```
cmake                                                 \
    -DCMAKE_INSTALL_PREFIX=\$(pwd)/cmake-out          \
    -DCMAKE_BUILD_TYPE=Release                        \
    -DCMAKE_TOOLCHAIN_FILE=\$(pwd)/examples
    /arm/ethos-u-setup/arm-none-eabi-gcc.cmake        \
    -DEXECUTORCH_SELECT_OPS_LIST="aten::_softmax.out" \
    -B\$(pwd)/cmake-out/examples/arm                  \
    $(pwd)/examples/arm

cmake --build \$(pwd)/cmake-out/examples/arm          \
--config Release
```

Listing 12: The final two commands to cross-compile libarm_portable_ops_lib.a library.

So far, we set up ExecuTorch and continue making Mbed OS ready for our hardware.

Mbed OS setup for P-Nucleo-WB55RG

First, we make our P-Nucleo-WB55RG hardware available for Mbed OS. Therefore, we create a `NUCLEO_WB55RG.cmake` file that defines the configuration for the upload methods OpenOCD and PyOCD. The name of the configuration file must be in upper case and identically named as the board name in the `cmake-variants.yaml` file. Listing 13 presents the adjusted section of the configuration file compared to other configuration files from different targets.

```
# Config options for PYOCD
# ------------------------------------------------------

set(PYOCD_UPLOAD_ENABLED TRUE)
set(PYOCD_TARGET_NAME STM32WB55RGVX)
set(PYOCD_CLOCK_SPEED 4000k)

# Config options for OPENOCD
# ------------------------------------------------------

set(OPENOCD_UPLOAD_ENABLED TRUE)
set(OPENOCD_CHIP_CONFIG_COMMANDS
    -f ${OpenOCD_SCRIPT_DIR}/board/st_nucleo_wb55.cfg)
```

Listing 13: Definition of PyOCD and OpenOCD as upload methods for NUCLEO_WB55RG.

We provide the correct `NUCLEO_WB55RG.cmake` file in the `config` folder of our Mbed Torch Fusion OS project and copy it into the `upload_method_cfg` folder of Mbed OS. The host system must install both PyOCD and OpenOCD. Therefore, we install PyOCD in the virtual Python environment of Mbed OS and OpenOCD via the package manager of the Ubuntu 22.04 x86_64 host system.

```
$ source mbed-os/venv/bin/activate
$ python3 -m pip install -U pyocd
$ pyocd pack --update
```

```
$ pyocd pack --install stm32wb55rg
$ deactivate
```

Listing 14: Installation of PyOCD on the host OS.

```
$ sudo apt-get install openocd
```

Listing 15: Installation of OpenOCD on the host OS.

Listing 14 and 15 show the corresponding installation commands.

Mbed OS and ExecuTorch compilation and test

We must provide the model as a header file for the final compilation of Mbed OS and ExecuTroch. The header file consists of an array with bytes and a C attribute specifier that tells the compiler to place the array in the specific section named *.sram.data*. So, we convert the `softmax.pte` file into the `model_pte.h` file using the `pte_to_header.py`[9] script. We store the header file in the root directory of our Mbed Torch Fusion OS project to make it available for the compiler and compile the entire project in the command line as shown in Listing 16. We show in a separate video[10] how to compile it using VS Code and it is also described in the `README.md`[11] of Mbed OS.

```
mkdir build && cd build
cmake .. -GNinja -DCMAKE_BUILD_TYPE=Develop \
-DMBED_TARGET=NUCLEO_WB55RG
ninja flash-SoftMaxTorchRunner
```

Listing 16: The final build commands.

Afterwards, we flash the binary onto the P-Nucleo-WB55RG hardware and encounter the error mentioned in Listing 17. The error shows the register context at the time of exception and locations in the code where the error happened.

```
I executorch:main.cpp:68] Model PTE file loaded.
Size: 960 bytes.
E executorch:program.cpp:108] Program identifier
'X' != expected 'ET12'
I executorch:main.cpp:72] Program loading failed
@ 0x0x20000E20: 0x23
F executorch:result.h:165] In function CheckOk(), assert
failed: hasValue_

++ MbedOS Fault Handler ++

FaultType: HardFault

Context:
R    0: 0000004B
R    1: 0000000A
R    2: 00000000
R    3: 00000000
R    4: 0804DCC7
R    5: 0804E314
R    6: 00000023
R    7: 20000E20
R    8: 00000000
R    9: 00000000
R   10: 00000000
R   11: 00000000
```

[9] https://github.com/pytorch/executorch/blob/ceb1f1d05ceab420644a3633264f13547dc02411/examples/arm/executor_runner/pte_to_header.py

[10] https://youtu.be/jc_GCxAAO20

[11] https://github.com/mbed-ce/mbed-os/wiki/Project-Setup:-VS-Code

```
R  12: 00000001
SP   : 20003CB8
LR   : 0800F10B
PC   : 08006CB6
xPSR : 010F0000
PSP  : 20003C50
MSP  : 2002FFC0
CPUID: 410FC241
HFSR : 40000000
MMFSR: 00000000
BFSR : 00000000
UFSR : 00000001
DFSR : 00000008
AFSR : 00000000
Mode : Thread
Priv : Privileged
Stack: PSP

-- MbedOS Fault Handler --
```

Listing 17: The Mbed OS HardFault error.

We debug the error described in Section 4.3 and solve the error by creating the `pte_to_header.patch` file. The patch removes the compiler instruction *.sram.data* in the header file that places the variable `model_pte` in `main.cpp` in the memory section called *.sram.data*. However, the data aligns to a 16-byte boundary in the section managed by Mbed OS. We do that because we want to hand over the memory management responsibility to Mbed OS. Otherwise, we would not find the model, and the P-Nucleo-WB55RG crashed with the error in Listing 17.
For debugging and analyzing the outputs of P-Nucleo-WB55RG, we cannot use the Mbed CLI2 because it requires a specific arm configuration in the `arm-none-eabi-gcc.cmake` file. We make the console output of the serial interface of the P-Nucelo-WB55RG human-readable by using Minicom.[12] The proprietary configuration is not obtainable by MBed CE but only by MBed OS.

[12] https://help.ubuntu.com/community/Minicom