Computing ecosystems: neural networks and embedded hardware platforms

Teresa Pelinski*
Centre for Digital Music, Queen Mary University of London, t.pelinskiramos@qmul.ac.uk

Franco Caspe
Centre for Digital Music, Queen Mary University of London, f.s.caspe@qmul.ac.uk

Andrew McPherson
Dyson School of Design Engineering, Imperial College London, andrew.mcpherson@imperial.ac.uk

Mark Sandler
Centre for Digital Music, Queen Mary University of London, mark.sandler@qmul.ac.uk

Embedded hardware platforms such as single-board computers (e.g., Raspberry Pi, Bela) or microcontrollers (e.g., Teensy, Arduino Uno) offer an entry point for beginners into physical computing. However, deploying neural networks into these platforms is challenging for various reasons: It requires lower-level software development skills, as machine learning toolkits are typically not incorporated into these platforms. Besides, the long compilation times burden debugging and quick prototyping and experimentation. Due to the low-resource nature of embedded hardware platforms, neural networks are usually trained on a host machine, which involves a back-and-forth of data, platforms and programming languages. We inquire how these computing ecosystems might be designed to facilitate prototyping and experimentation and integrate into existing programming workflows.

CCS CONCEPTS • Embedded software • Embedded hardware • Neural networks

Additional Keywords and Phrases: Computing ecosystems, inference paradigms

ACM Reference Format:

First Author's Name, Initials, and Last Name, Second Author's Name, Initials, and Last Name, and Third Author's Name, Initials, and Last Name. 2018. The Title of the Paper: ACM Conference Proceedings Manuscript Submission Template: This is the subtitle of the paper, this document both explains and embodies the submission format for authors using Word. In Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY. ACM, New York, NY, USA, 10 pages. NOTE: This block will be automatically generated when manuscripts are processed after acceptance.

This work was presented at the **CHI2023 Workshop [WS2] - Beyond Prototyping Boards: Future Paradigms for Electronics Toolkits**
CHI '23, April 23-28, 2023, Hamburg, Germany

1 INTRODUCTION

Embedded hardware platforms such as single-board computers (e.g., Raspberry Pi, Bela) or microcontrollers (e.g., Teensy, Arduino) offer an entry point for beginners into physical computing. Many of these platforms provide open-source APIs that, among other things, abstract the complexities of interacting with peripherals or dealing with interrupts on real-time systems. Some platforms provide their own IDE (Integrated Development Environment, e.g. Arduino IDE) where the programmer writes the code on the host machine (e.g. laptop) and the IDE cross-compiles it and uploads it onto the embedded board via USB or SD card. Other platforms allow writing code directly onto the device through graphical interfaces (e.g. the Bela browser-based IDE). These APIs and IDEs support prototyping and experimentation through abstracting lower-level software development processes such as configuring compilers and finding and building libraries. However, prototyping and experimentation are also supported through non-technical means: many embedded platforms have well-documented code bases, tutorials and examples and online learning communities that communicate with one another and the platform developers through web-based forums [1]. Partly as a result, single-board computers and microcontrollers are extensively used as teaching resources in electronics and computing courses [2].

As beginning programmers gain experience, they might want to experiment with tools not included with an embedded platform by default. For instance, there are increasing online resources that teach how to program and run machine learning models, and commonly used deep learning frameworks (e.g., Pytorch[1], Tensorflow[2]) abstract the mathematical complexities of deep learning through modular approaches. Moreover, there exist many tutorial Jupyter notebooks that allow interactively following template code, and some platforms provide online servers and (restricted) free-of-charge access to GPUs, reducing the hurdle of locally setting up development environments. These tools and resources facilitate the experimentation and prototyping with such models of users with limited programming knowledge.

Integrating deep learning workflows with embedded physical computing workflow offers exciting possibilities for embedded hardware platforms, particularly for interaction design. For example, sensor signals can be fed into neural networks, which can be trained to recognise gestures and generate complex gesture-to-sound mappings in musical interfaces. However, machine learning workflows prove to be particularly complex on embedded hardware platforms for a variety of reasons: often, compiling and deploying neural networks into such platforms requires lower-level software development skills, since the embedded system's own APIs and IDE do not provide support for such libraries nor for the movement of data between host and embedded device. In other cases, the machine learning libraries do not support bare-metal execution and must be deployed on an operating system with virtual memory and threading support. Below, we discuss the two main neural network inference paradigms on embedded systems and their computing ecosystems, and a few ideas that could improve the ease of prototyping with deep learning in embedded systems.

2 INFERENCE PARADIGMS

In machine learning, *inference* refers to the process by which a trained model makes a prediction in response to input data. On embedded systems, the model *training* will typically occur on the host platform, whilst inference can occur either on the embedded hardware platform itself or in a different environment. Below, we discuss both approaches.

2.1 On-board inference

Neural networks tend to be computationally expensive, especially if they have many layers. Running inference on an embedded hardware platform is challenging due to the limited computational resources available on the board, which can

[1] https://pytorch.org/
[2] https://www.tensorflow.org/

make the prediction slow or impossible if the model is too computationally complex. However, there exist many scenarios in which on-board inference is desirable, particularly in low-latency applications. For instance, systems with strict real-time deadlines (for example, musical instruments) cannot afford to rely on unbounded network or server response times. Besides the challenges of building neural networks which are lightweight enough to run on embedded hardware platforms, there exist many practical difficulties associated with software development: Often inference engines (e.g., Torchscript[3], Tensorflow Lite[4]) have not been compiled for the desired target platform, resulting in the need to build the libraries from source, which can be considerably challenging due to dependency conflicts and a lack of platform-specific instructions. Moreover, there is a back-and-forth movement of data, platforms (host computer, GPU cluster and embedded platform), and programming languages (e.g. C++ in the embedded platform, python in the host): the dataset might be recorded on the embedded platform, but the machine learning model will be typically trained on the host platform (or a computing cluster) yet perform inference on the embedded platform. Finally, for those platforms in which compilation occurs on-device (self-hosting platforms), the limited computational capabilities complicate prototyping since compilation times tend to be considerably longer. Alternatively, the programmer can set up a cross-compilation environment, which in turn requires considerable software development skills. USB hardware accelerators that can be plugged into embedded platforms are also available; however, interfacing with them can be daunting for a non-expert developer.

2.2 Off-board inference

Inference can also be run on a platform with larger computational resources, such as a laptop or a server in a computing cluster. The embedded hardware platform can stream some sensor values to the server running the model and later receive the model's prediction. For example, an embedded hardware platform could extract some features of an input audio signal (e.g. pitch and loudness) send those features to a server running a deep learning model, and once the model's inference in the server is finished, receive the model's prediction and do some operation with it (e.g. use the model's output values to control a synthesiser). This approach is typically unsuitable for hard real-time systems due to the server's unbounded response times. It also requires setting up a communication protocol between the embedded platform and the platform running the inference. However, since the embedded platform only sends and receives data, it might be more easily incorporated into the user's existing workflows. For instance, in the previous example, the audio features could be sent through OSC from the embedded platform to a Jupyter notebook running in the laptop.

3 LESS VISCOUS COMPUTING ECOSYSTEMS

In both approaches, there is a computing ecosystem involving an exchange of data, platforms and programming languages. Embedded platforms typically prioritise simple prototyping and experimentation with electronics, but this simplicity breaks down when trying to integrate machine learning workflows. The Cognitive Dimensions of Notations Framework by Blackwell and Green [3] is a useful analytical tool here: the authors speak of the *viscosity* of a system, which refers to its resistance to change. The process of compiling inference engines for embedded platforms is viscous since it often takes many iterations, either due to toolchain complexity or finding a network architecture with adequate performance that fits within the available processing constraints. Although Blackwell and Green use "viscosity" to refer to the number of actions needed to accomplish a goal, here we might extend it to the time it takes for a system to change. In this regard, computing ecosystems involving a back-and-forth of data, platforms and programming languages, and with long compilation and

[3] https://pytorch.org/docs/stable/jit.html
[4] https://www.tensorflow.org/lite

debugging times can become highly viscous. We are interested in investigating how these ecosystems and their workflows can be configured so that they can be approachable by non-expert programmers through easy prototyping and experimentation and integrate into existing programming workflows.

In our previous work, we developed a pipeline[5] [4] to record datasets and run neural networks (on-board inference) on the Bela embedded hardware platform [5]. We chose Bela due to its low input-output latency (1 ms) and its established maker community [6] and online support. Our pipeline offers a documented step-by-step tutorial to record a dataset, train a lightweight neural network, cross-compile the C++ inference code, and run it in real-time on the embedded platform. Whilst we believe this is a first step towards bridging the deep learning and physical computing maker communities and practices, the pipeline still relies on a variety of tools and languages (C++, python, Docker, CMake, etc.). Below we discuss a few ideas that could improve the ease of prototyping with deep learning in embedded systems by reducing the number of required tools and languages.

One idea to improve the ease of prototyping with deep learning on embedded platforms is to offer Jupyter notebook integration. Popular workflows in deep learning and data science involve python and Jupyter notebooks. Jupyter notebooks allow running code in blocks, which simplifies the debugging process and facilitates interactively changing and testing parts of the code. Moreover, they support data visualisation through plots and tables. Jupyter notebook integrations in embedded platforms could ease the process of recording datasets and facilitate the movement of data between laptop and embedded platform by keeping a consistent language (python) and environment (Jupyter) across platforms. These integrations are already available in some embedded platforms such as AMD Adaptive Computing Platforms[6], which support Jupyter notebooks through pynq[7]. However, physical computing platforms such as Bela or Arduino do not seem to have a stable Jupyter notebook integration that could, for instance, allow dynamic access to the sensor data through a python API. A python API serving sensor data from the embedded platform would allow hosting the Jupyter notebook on the laptop and potentially train the models on the same notebook, reducing even further the back-and-forth of platforms and languages.

Pytorch is one of the most popular frameworks in deep learning. However, its underlying C++ inference engine (TorchScript), requires dynamic memory allocation and a filesystem to operate, which makes its implementation unfeasible in microcontrollers. Moreover, C++ inference engines tend to rely on resizable data structures that dynamically allocate memory, which can make them inappropriate for real-time implementation [7], although for some frameworks, those dynamic allocations can be bypassed by calling the model's forward method before the real-time loop starts [8]. Practitioners typically rely on inference engines with C++ static runtimes such as TFLite. However, the conversion from Pytorch models into TFLite-compatible models is not always possible due to differences in layer implementations and computing primitives. Thus, a Pytorch-friendly static inference engine would avoid cumbersome conversions across deep learning frameworks.

For platforms where filesystems are available, such as Bela or Raspberry Pi, a further improvement could be to incorporate popular inference engines (e.g., TFLite, TorchScript) into the embedded platform's APIs, so that programmers can easily load trained models onto their physical computing workflows without the need of running complex compilation recipes. In particular, if the input and output dimensionality are fixed (e.g., fixed buffer size and sample-rate in an audio input-output model), pre-compiled wrappers would enable loading different models without the need to recompile on-device or cross-compile on the laptop. Thus, the user could train models and export models in their laptop and then simply copy and run them in the embedded platform without the need to re-compile the inference program each time.

[5] https://github.com/pelinski/bela-dl-pipeline
[6] https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html
[7] http://www.pynq.io

REFERENCES

[1] S. Kuznetsov and E. Paulos, 'Rise of the Expert Amateur: DIY Projects, Communities, and Cultures', Dec. 2010, p. 304. doi: 10.1145/1868914.1868950.

[2] P. Abichandani, V. Sivakumar, D. Lobo, C. Iaboni, and P. Shekhar, 'Internet-of-Things Curriculum, Pedagogy, and Assessment for STEM Education: A Review of Literature', *IEEE Access*, vol. 10, pp. 38351–38369, 2022, doi: 10.1109/ACCESS.2022.3164709.

[3] A. F. Blackwell and T. Green, 'Notational Systems—The Cognitive Dimensions of Notations Framework', in *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science*, J. M. Carroll, Ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, pp. 103–133.

[4] T. Pelinski, R. Díaz, A. L. Benito Temprano, and A. McPherson, 'Pipeline for recording datasets and running neural networks on the Bela embedded hardware platform', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Mexico City, Mexico, 2023.

[5] A. McPherson and V. Zappi, 'An environment for submillisecond-latency audio and sensor processing on BeagleBone black', in *138th Audio Engineering Society Convention*, Warsaw, Poland, May 2015. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=17755

[6] F. Morreale, G. Moro, A. Chamberlain, S. Benford, and A. McPherson, 'Building a Maker Community Around an Open Hardware Platform', in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, Denver, CO, USA, May 2017. [Online]. Available: https://dl.acm.org/doi/10.1145/3025453.3026056

[7] J. Chowdhury, 'RTNeural: Fast Neural Inferencing for Real-Time Systems'. Jun. 06, 2021. Accessed: Dec. 01, 2021. [Online]. Available: http://arxiv.org/abs/2106.03037

[8] D. Stefani, S. Peroni, and L. Turchet, 'A Comparison of Deep Learning Inference Engines for Embedded Real-Time Audio Classification', in *Proceedings of the 25th International Conference on Digital Audio Effects (DAFx20in22)*, Vienna, Austria, Sep. 2022, pp. 256–263. [Online]. Available: https://iris.unitn.it/handle/11572/364734