CFU Playground: Full-Stack Open-Source Framework for Tiny Machine Learning (TinyML) Acceleration on FPGAs

Tatchemo Guiafaing Ronald VR512344

2nd August 2024

1 Introduction

TinyML is the deployment of machine learning (ML) algorithms onto low-cost, low-power, and resource-constrained micro-controller (MCU) systems. It enables on-device ML, using a fraction of the compute resources needed for traditional ML systems. The ML models running on-device can be employed for intelligent, on-device sensor analytics, unlocking always-on ML use cases.

While there are many benefits to TinyML, the heterogeneity of MCU hardware and limited resources available on them presents new challenges. Conversely, a full-stack framework allows users to explore experimental and bespoke architectures customized and co-optimized for embedded ML. There is a necessity for a rapid deploy-profile-optimization feedback loop to permit both ML hardware and software developers to achieve significant returns from relatively small investments in customization. Utilizing CFU Playground's design and evaluation loop can yield substantial speedups between 55× and 75×.

1.1 Challenges in Machine Learning Techniques

Given the pressing need for energy efficiency when running ML on embedded platforms, custom processor support and hardware accelerators could provide essential solutions. However, the field of ML is still in its infancy and is rapidly evolving. Thus, it is desirable to avoid massive non-recurring engineering (NRE) costs upfront, especially for low-cost embedded ML systems.

This framework also enables users to perform design space exploration between resources allocated to the CFU, CPU, or memory systems. Exploiting this ability and strategically using parallelism and pipelining led to an overall speedup of 75×. An undergraduate-level intern with minimal FPGA and hardware experience achieved this speedup in under four weeks, thanks to the various abstraction layers.

FPGAs facilitate complex design space exploration for customized microarchitectures beyond just the ML accelerator but also include the CPU. The FPGA platform offers the opportunity to customize the processor to perform the application's computations efficiently. Given that ML computed strategies tend to be regular and repetitive, focusing on improving the execution of these "hot" computations while using standard instructions for the rest can be highly beneficial. A small amount of custom hardware for these hotspots, leveraging the bit-level flexibility of FPGAs, can lead to substantial performance improvements.

1.2 Could FPGA Be The Reliable Future of ML and Computer Vision?

In recent years, FPGA-based ML accelerators have gained momentum in the race to speed up ML tasks due to their efficiency compared to GPUs, lighter investment than ASICs, and overall flexibility resulting from their reconfigurability. Consequently, many tools have been designed to support FPGA design for ML.

Nevertheless, conducting a direct quantitative comparison against other tools is challenging. We believe there is no "one size fits all" solution for designing custom ML hardware accelerators. Therefore, the best choice of workflow is task-dependent and is ultimately up to the developer.

We provide a qualitative comparison of CFU Playground against prior work along various dimensions, highlighting how CFU Playground adds flexibility on top of existing solutions, particularly useful in resource-constrained embedded systems where discrete accelerators are not always feasible.

CFU Playground's ability to support the Renode emulator—emulating the physical hardware system—offers users a unique experience. While following a hardware-in-the-loop process and running on a physical board is recommended, Renode emulation can facilitate testing CFUs on other boards without requiring physical access. Renode simulates the CPU's ISA alongside cycle-accurate Verilog simulations of the CFU, allowing for functional correctness testing.

Additionally, it simulates RAM, ROM, and UART, capturing waveforms from CFU operations, which is invaluable for identifying hardware design errors in user-defined CFUs.

To aid in the advancement of the field, efforts have been made to benchmark these heavily resource-constrained TinyML systems, driving standardization and innovation in the industry.

CFU Playground comes packaged with MLPerf Tiny deep learning workloads for benchmarking. Two noteworthy use cases illustrate how an agile design flow like ours can support the benchmarking ecosystem and research community. The first occurs when a user has a specific idea to accelerate an existing model. Rapid prototyping and benchmarking before embarking on the costly and time-consuming process of building an ASIC accelerator enable the exploration of new hardware architectures.

The second use case addresses the hardware lottery problem, facilitating the exploration of new model architectures.

There are existing tools and workflows for accelerating ML with custom hardware, but the tools for ASIP accelerators are often commercial and entail a costly ramp-up process.

2 Practical Computerization of Three Datasets on an FPGA

The main objective is to computerize three different datasets: **EMNIST** (available within the PyTorch API), **KMNIST** (available within the PyTorch API), and **Imagenette** (available within the PyTorch API), and thereby provide results before and after pruning and quantization of those datasets.

Results obtained from each dataset after pruning and quantization were achieved in lesser time, even though they were not as accurate as those derived from convolutional neural networks. The FPGA platform allows for an accelerator unit to be tightly coupled into the CPU pipeline, which can be invoked easily by adding new custom instructions that complement the CPU's standard functions.

This approach is an alternative to treating ML accelerators as discrete processing units.

Along with programming ease, tight coupling is beneficial for latency-bound applications needed for TinyML. Furthermore, the bit-level flexibility of FPGAs allows efficient implementation of operations on small data sizes and non-standard data representations, including packing, unpacking, and converting between data types.

As visible on the GitHub pages, the accuracy after the last epoch, precision after the epoch, and F1 score after the epoch were all lower than the typical results obtained from neural networks.

Thus, the acceleration on FPGAs for Machine Learning techniques is remarkably fast compared to traditional ML computations, which tend to be regular and repetitive, with the same sequence of operations repeated millions of times. Unfortunately, the results obtained were significantly lower.

To fully leverage these techniques in the future, we must ensure a higher accuracy is achieved for both precision and the F1 score after the epochs. Here are some strategies that might help:

2.1 Retraining

- Retrain after Pruning/Quantization: Fine-tune the model after pruning and quantization to recover any lost accuracy and improve performance.
- Use Knowledge Distillation: Utilize a smaller model to learn from a larger one to help retain accuracy following transformations.

2.2 Hyperparameter Tuning

- Optimize Hyperparameters: Readjust learning rate, batch size, and optimization algorithms after applying pruning and quantization.