GENN: Enable Flexible and Efficient AI for Resource-Constrained Platforms

Yan Zhu, Kaija Mikes, Karthik Ganesan, Natalie Enright Jerger
University of Toronto
Toronto, Ontario

1 Problem Statement

Deep learning (DL) is experiencing increased interest in resource-constrained devices [15]. However, modern DL frameworks, e.g., TensorFlow and PyTorch, are designed and optimized for high-performance platforms [13]. For usability, most frameworks use interpreted languages and require extensive libraries like Nvidia CUDA; executing them on resource-constrained devices remains challenging [6]. Therefore, it is crucial to enable more memory and environment-friendly AI for platforms such as low-end Internet of Things (IoTs), simulators, and high-level synthesis (HLS).

2 Related Work and Motivation

While prior work implements DL models in C/C++ [2, 4, 9] to eliminate the resource constraints, they suffer from several drawbacks:

(1) Due to the memory or operating system (OS) requirements, some tools can only run on certain systems, which limits portability and flexibility. For instance, CMSIS-NN [9] is board-specific and only works for Arm Cortex-M processors. uTensor [2] requires Mbed OS, making both simulation and real device execution difficult. TensorFlow Lite Micro [4] employs dynamic memory allocation, which is not supported by HLS tools such as Vivado [17].
(2) Some tools only provide C/C++ DNN layer functions and do not support automatic C model generation; this limits usability as users have to manually port customized models into compiled languages.
(3) Many frameworks support TensorFlow to C conversion but fail to support increasingly popular PyTorch [5].
(4) More straightforward tools exhibit worse inference time as they are not well-optimized while the better-performing ones are not intuitive to use [14].
(5) While floating point (FP) and quantized DNNs have been investigated, implementation in C of PyTorch sparsity has not been explored.

	GENN	CMSIS-NN [9]	uTensor [2]	TensorFlow Lite Micro [4]
Platform Requirement	None	Arm Cortex-M processors	Mbed OS	Dynamic Memory Allocation
Auto-Generation	✓	×	✓	✓
DL Framework	PyTorch	None	TensorFlow	TensorFlow
Usability	Easy	Very Hard [14]	Medium [14]	Hard [14]
Sparsity	✓	×	×	×

Table 1: GENN and prior work comparison. The Platform Requirement refers to the conditions required to run models on the tools.

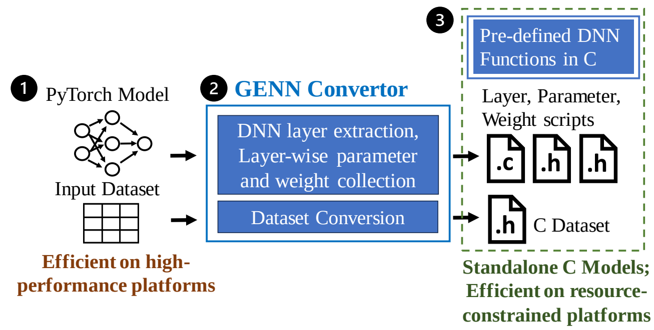

Figure 1: Overall pipeline of converting PyTorch model into standalone C package.

3 Design

Based on the insights above, we develop GENN, an automatic PyTorch-to-C model conversion pipeline (Figure 1) with a high degree of generality, flexibility, and usability.

GENN converts a trained FP, quantized, or sparse FP PyTorch network into standalone C for resource-constrained DNN inference. The converter processes the model layer by layer and stores the PyTorch parameters and weight tensors in an ordered dictionary. After looping over the entire network, the GENN converter prints out the parameters and weights into C format and creates a main file that declares the DNN layers. The generated C code, together with our pre-defined GENN DL functions, runs on a variety of platforms as it has no external library dependencies and performs no dynamic memory allocation. To save memory, we replace floating point multiplications and division with bit-shifting operations [9] for our quantized models.

To demonstrate the effectiveness of GENN, we convert eight models from different IoT applications [1, 15] into C as a benchmark suite (**Table 2**). For each model, we provide the FP, int16, and sparse versions. For sparse models, 90% of their weights are pruned.

4 Evaluation

We evaluate GENN on a simulator and a hardware platform. We use Thumbulator [7], a cycle-accurate simulator for the ARM Cortex-M0+ CPU, running at 24 MHz, for simulation. For real hardware evaluation, we use an STM32-NUCLEO-F411RE board which has a 32-bit ARM Cortex-M4 CPU, running at 100 MHz and with 128 KB of RAM and 512 KB of flash memory. We only run five out of the eight benchmark models on the real device, as the rest do not fit into the on-chip memory.

Simulator Results: Figure 2 illustrates the simulation inference time. The quantized models show a significant speedup. This is because the ARM-M0+ CPU that the Thumbulator models does not have floating-point hardware; the floating-point operations are emulated in software, leading to a significant slowdown. Also, we

Dataset	Model	Architecture	Type	Accuracy (%)	Size (KB)
MNIST [10]	MLP	Linear → ReLU → Linear	Float	96.79	397.54
			Quant	96.77	198.77
			Sparse	94.38	159.34
	CNN	(Conv2D → MaxPool2D → ReLU)×2 → Linear → ReLU → Linear	Float	98.73	85.31
			Quant	98.36	42.66
			Sparse	96.85	17.46
Electrocardiogram (ECG) [12]	MLP	(Linear → PReLU)×13 → Linear → Sigmoid	Float	97.32	487.42
			Quant	96.00	243.71
			Sparse	95.00	100.28
Keyword Spotting (KWS) [16]	MLP	(Linear → ReLU)×3 → AdaptiveAvgPool1d → Linear	Float	95.05	8535.00
			Quant	94.89	4267.50
			Sparse	94.89	1711.80
	CNN	(Conv1D → ReLU)×2 → Linear → Linear → ReLU → AdaptiveAvgPool1d → Linear	Float	96.17	729.25
			Quant	96.00	364.63
			Sparse	91.53	146.98
	DS_CNN	Conv1D → (Conv1D → ReLU)×8 → AdaptiveAvgPool1d → Linear	Float	98.40	109.76
			Quant	98.00	54.88
			Sparse	93.29	25.98
Human Action Recognition (HAR) [8]	MLP	(Linear → ReLU)×2 → Linear	Float	90.87	6674.02
			Quant	91.75	3337.01
			Sparse	89.00	1339.63
	CNN	(Conv2D → MaxPool2D → ReLU)×2 → (Linear → ReLU)×2 → Linear	Float	91.08	8553.23
			Quant	91.08	4276.61
			Sparse	82.35	3425.43

Table 2: Models offer by GENN benchmark suite (DS CNN: depthwise separable convolutional neural network).

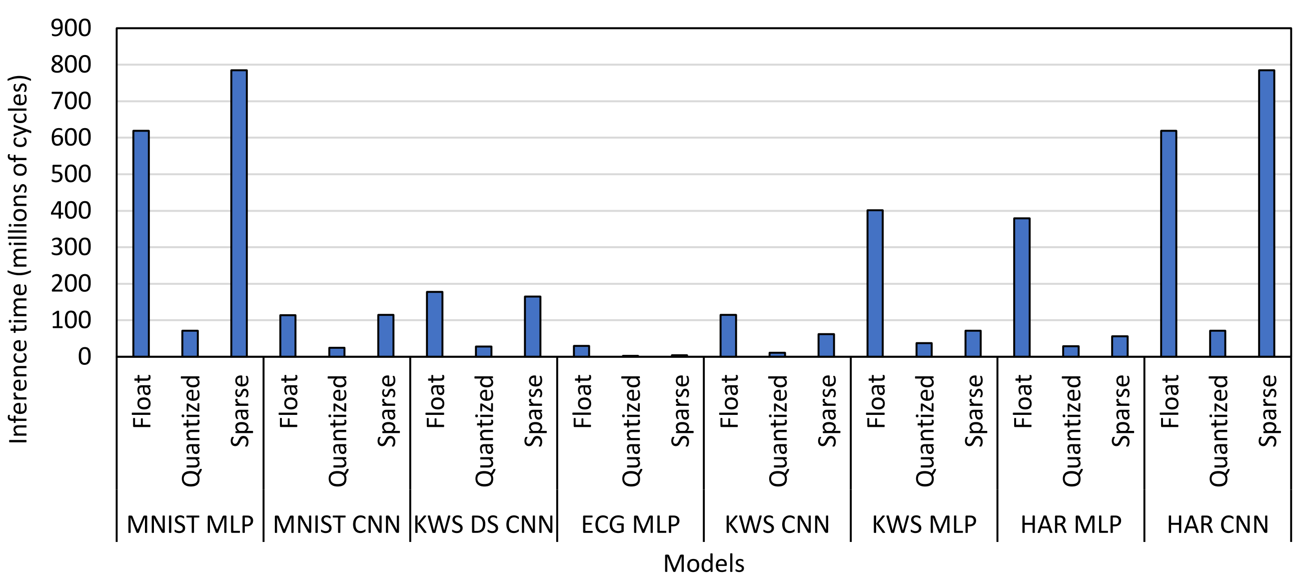

Figure 2: Inference time for GENN DNNs on Thumbulator.

can directly run our models on the simulator, implying the ease of use and generality of GENN.

Hardware Results: Figures 3 and 4 illustrate the memory footprint and inference time of the DNNs. On average, quantization and sparsity reduce the memory footprint by 44% and 60%, respectively.

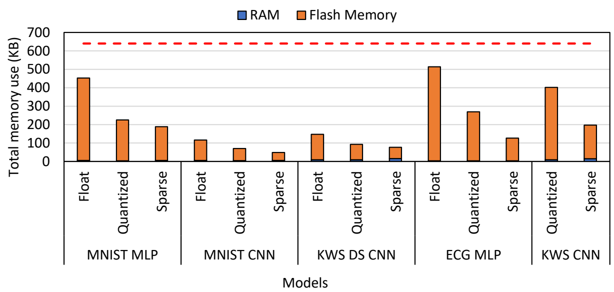

Figure 3: Memory footprint for GENN DNNs.

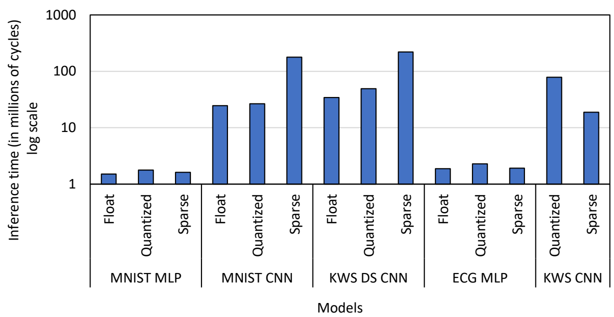

Figure 4: Inference time for GENN DNNs on real device.

For inference, we see a slight increase in runtime for the quantized model. This is because the quantized models require extra quantization and dequantization operations during inference, which adds overhead. For the sparse models, we store only the indices and values of the non-zero weights. Thus the overhead of this irregular indexing adds to the runtime of the sparse models. Optimizing the runtime of sparse models is a focus of our future work.

We were unable to get CMSIS-NN [9] and TensorFlow Lite Micro [4] to work out of the box. However, prior work shows that standalone C models can outperform CMSIS-NN [9] and TensorFlow Lite Micro [4] in both inference time and memory footprint [3, 11, 18].

5 Conclusion

Memory and platform restrictions lead to difficulties running DNNs on resource-constrained devices. We develop GENN, an automatic PyTorch model to C converter with good usability and flexibility that allows more constraint-free low-end AI. We open-source our codebase and plan to add more features to GENN, including supporting quantized sparse models and adding more advanced DL layer functions.

References

[1] Salma Abdel Magid, Francesco Petrini, and Behnam Dezfouli. 2020. Image Classification on IoT Edge Devices: Profiling and Modeling. *Cluster Computing* 23, 2 (jun 2020), 1025–1043. https://doi.org/10.1007/s10586-019-02971-9

[2] Michael Bartling. 2019. uTensor TinyML AI inference library. https://github.com/uTensor/uTensor.

[3] Usman Ali Butt. 2021. On the deployment of Artificial Neural Networks (ANN) in low-cost embedded systems. https://webthesis.biblio.polito.it/19692/

[4] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Shlomi Regev, Rocky Rhodes, Tiezhen Wang, and Pete Warden. 2021. TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems. arXiv:2010.08678 [cs.LG]

[5] Google. 2023. PyTorch vs. Tensorflow Interest over time. https://trends.google.com/trends/explore?date=today%205-y&q=%2Fg%2F11gd3905v1,%2Fg%2F11bwp1s2k3

[6] Oliver Hahm, Emmanuel Baccelli, Hauke Petersen, and Nicolas Tsiftes. 2016. Operating Systems for Low-End Devices in the Internet of Things: A Survey. *IEEE Internet of Things Journal* 3, 5 (Oct 2016), 720–734. https://doi.org/10.1109/JIOT.2015.2505901

[7] Matthew Hicks. 2016. *Thumbulator: Cycle accurate ARMv6-m instruction set simulator.* https://bit.ly/2RJX36A.

[8] Reyes Ortiz Jorge, Anguita Davide, Ghio Alessandro, Oneto Luca, and Parra Xavier. 2012. Human Activity Recognition Using Smartphones.

[9] Liangzhen Lai, Naveen Suda, and Vikas Chandra. 2018. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. arXiv:1801.06601 [cs.NE]

[10] Yann LeCun. [n. d.]. https://pytorch.org/vision/main/generated/torchvision.datasets.MNIST.html#torchvision.datasets.MNIST

[11] Jianjia Ma. 2020. Neural Network on Microcontroller. https://github.com/majianjia/nnom

[12] George Moody and Roger Mark. 2005. MIT-BiH Arrhythmia Database. https://www.physionet.org/content/mitdb/1.0.0/

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv:1912.01703 [cs.LG]

[14] Christos Profentzas, Magnus Almgren, and Olaf Landsiedel. 2021. Performance of Deep Neural Networks on Low-Power IoT Devices. In *Proceedings of the Workshop on Benchmarking Cyber-Physical Systems and Internet of Things* (Nashville, Tennessee) *(CPS-IoTBench '21)*. Association for Computing Machinery, New York, NY, USA, 32–37. https://doi.org/10.1145/3458473.3458823

[15] Tausifa Jan Saleem and Mohammad Ahsan Chishti. 2021. Deep learning for the internet of things: Potential benefits and use-cases. *Digital Communications and Networks* 7, 4 (2021), 526–542. https://doi.org/10.1016/j.dcan.2020.12.002

[16] Pete Warden. 2017. Launching the speech commands dataset. https://blog.research.google/2017/08/launching-speech-commands-dataset.html

[17] Xilinx. [n. d.]. Vivado Design Suite User Guide. https://docs.xilinx.com/v/u/en-US/ug902-vivado-high-level-synthesis

[18] Raphael Zingg and Matthias Rosenthal. 2020. Artificial Intelligence on Microcontrollers. https://github.com/InES-HPMM/Artificial_Intelligence_on_Microcontrollers/blob/master/Artificial_Intelligence_on_Microcontrollers.pdf