



Article

MAIL: Micro-Accelerator-in-the-Loop Framework for MCU Integrated Accelerator Peripheral Fast Prototyping

Jisu Kwon  and Daejin Park * 

School of Electronic and Electrical Engineering, Kyungpook National University, Daegu 41566, Republic of Korea; kjisu96@knu.ac.kr

* Correspondence: boltanut@knu.ac.kr; Tel.: +82-53-950-5548

Abstract: The resource-constrained MCU-based platform is unable to use high-performance accelerators such as GPUs or servers due to insufficient resources for ML applications. We define a Micro-Accelerator (MA) that can accelerate ML operations by being connected to the on-chip bus peripheral of the MCU core. ML applications using general-purpose accelerators have a well-equipped SDK environment, making design and verification flow straightforward. In contrast, MA must be connected to the MCU core and on-chip bus interface within the chip. However, evaluating the interaction between the MCU core and an MA is challenging, as it requires the MA to connect with the core and the on-chip bus interface during target software execution. The cost of fabricating physical MA hardware is enormous, compounded by licensing issues with commercial cores. We propose a MA-in-the-loop (MAIL) framework that integrates a custom-designed MA into an emulation platform. This platform enables virtual execution by loading software onto the MCU, allowing observation of hardware-software interactions during ML execution. The proposed framework in this paper is a mixture of software that can emulate the environment in which general ML applications run on the MCU and RTL simulations to profile the acceleration on the MA. To evaluate the flow of ML software execution and performance changes according to the various architectures of MA in the framework, the MA can be reconfigured at runtime to explore the design space. To benchmark our proposed framework, we compared TinyML application profiles to the pure software execution. Experimental results show that the MA-accelerated framework performs comparably to actual MCUs, validating the efficacy of the proposed approach.



Academic Editor: Alexander Barkalov

Received: 29 December 2024

Revised: 15 January 2025

Accepted: 19 January 2025

Published: 21 January 2025

Citation: Kwon, J.; Park, D. MAIL: Micro-Accelerator-in-the-Loop Framework for MCU Integrated Accelerator Peripheral Fast Prototyping. *Appl. Sci.* **2025**, *15*, 1056. <https://doi.org/10.3390/app15031056>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: tiny machine learning; embedded system; microcontroller unit (MCU); accelerator; system emulator

1. Introduction

Attempts to accelerate machine learning software to hardware accelerators have been steadily considered important. It is necessary to ensure that the accelerator architecture to overcome bottlenecks in machine learning applications is suitable for the target neural computation and host system. Optimizing the accelerator architecture plays a huge role in improving the performance of machine learning applications. To address this, we divided the accelerators into three layers according to the weight parameter and the activation memory size of the neural architecture that can be fitted.

The first layer is a high-performance GPU-based on-cloud layer [1]. Various large-scale deep learning models can be applied with the device's abundant resource and power supply. The second layer is the on-board layer, and it has an accelerator connected with

interface protocol on the same system (e.g., PCB board) with the host device [2]. While the on-board layer has lower capability compared to the on-cloud layer, it provides sufficient resources to load and execute deep neural networks. The accelerator potential of deep learning models has been proven on various platforms such as FPGAs and ASICs [3]. However, applying tiny machine learning (TinyML), which enables artificial intelligence on low-cost, low-energy, resource-constrained microcontroller unit (MCU) devices, is a significant challenge.

TinyML requires highly efficient neural architectures and optimized inference libraries to operate within strict memory and power constraints [4,5]. These works evaluate acceleration performance and accelerator structure for typical deep neural network computation without considering internal MCU hardware characteristics. Therefore, we introduce a third layer, which is an on-chip accelerator for TinyML. It connects to the core peripheral bus inside the MCU chip and controls the accelerator behavior via memory-mapped special function register (SFR) access. We define this peripheral-level accelerator for TinyML in the on-chip layer as a *Micro-Accelerator (MA)*.

Tensorflow Lite for Microcontrollers (TFLM) [6] and recently conducted works make TinyML applications distributable to MCUs. These studies introduced a TinyML-optimized network architecture search mechanism and inference library to address the memory, latency, and energy constraints of MCUs [7,8]. The reason why it is difficult to apply the aforementioned upper two acceleration layers to TinyML applications running on MCU is the versatility of the accelerator. General-purpose accelerators like GPUs and NPU are offered to support various neural network models and deep learning architectures. TinyML software (<https://github.com/tensorflow/tflite-micro> (accessed on 21 November 2024)) embedded in on-chip flash-memory has static behavior similar to the hardware. This is because the software execution environment of an MCU exists in the form of firmware stored in on-chip flash memory, and once deployed, the firmware executes the same code repeatedly until it is reprogrammed using an external debugger device. To overcome these limitations, we propose a micro-accelerator design for TinyML that integrates tightly coupled software and hardware, enabling them to drive each other effectively. The internal hardware-software interaction of the MCU increases the complexity of the core bus-connected peripheral design. Peripherals of the MCU do not have their functional characteristics determined at design time but are determined at runtime by embedded software in on-chip flash memory. For cycle-accurate verification of RTL-designed MA, it is necessary to inspect the hardware triggers based on an understanding of the core bus connections of the target MCU. In addition, dominant overheads from general accelerators are memory transfers between host and device. To mitigate these challenges, the on-chip MA leverages direct memory access (DMA) blocks, eliminating the need for sequential register read/write operations and enhancing efficiency in neural computing tasks.

Figure 1 is the key idea for our approach. When designing accelerators, resource constraints are always present, requiring careful consideration of the trade-off between hardware area and delay. Therefore, it is necessary to design and evaluate the accelerator RTL to determine how to allocate limited resources effectively to achieve optimal performance. Applying the concept of pareto optimum, which represents the most optimal point among resources with trade-offs, to accelerator design ensures a specification that mitigates resource waste. To design a pareto-optimum MA architecture that is suitable for the target MCU device neural computation, iterative parameter exploration is required for accelerator RTL. MA parameters can be defined as memory-mapped register configurations by neural computation scale and control flow scenarios. The parameter exploration is required to find the pareto optimum in MA overhead and performance gain. Therefore, the parameter exploration requests a compressive MCU system simulation that includes

iterative cycle-accurate MA RTL simulation. However, before reaching the interaction stage between memory, DMA, and MA peripheral for neural computation acceleration, a number of component initialization must be conducted in the core. A comprehensive RTL simulation of a MA takes an excessively long time due to clocks consumed for instruction fetching. Approaches using high-level abstracted hardware and analytical models to mitigate RTL simulation time costs have significantly reduced simulation times [9,10]. However, this approach results in some loss of accuracy during the hardware abstraction process. Therefore, we propose a micro-accelerator-in-the-loop (MAIL) framework by adopting a software instruction executive emulators and RTL simulators mixed framework [11].

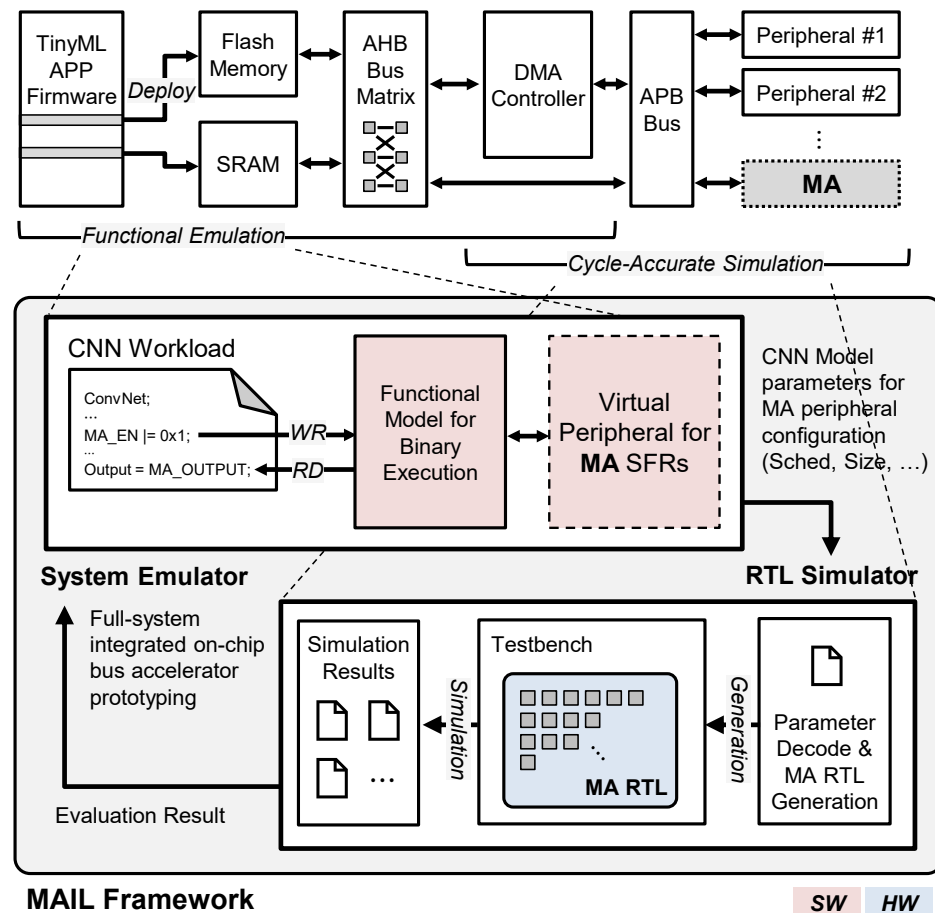


Figure 1. Micro-Accelerator-in-the-Loop Framework Overview. MAIL framework inputs scalable TinyML firmware and generates MA according to parameter from application. MAIL makes fast MA prototyping and evaluating MA within the RTL simulator. The RTL simulation result becomes feedback to the system emulator to be utilized as an MA parameter decision.

In this paper, we make the following specific contributions:

- We introduce the *MA* concept to represent MCU core bus attached peripheral-level TinyML hardware accelerator with the DMA. The proposed MA can be inserted for atomic neural network computation. The proposed MA supports atomic neural network computations by facilitating data transactions directly through the memory-mapped I/O (MMIO), eliminating the need for interface protocols on the PCB board. We believe that our MA can overcome gap between TinyML and general purpose accelerator for MCU, i.e., our MA can be used to allocate fine-grained acceleration for TinyML software.

- We propose *MAIL* framework for MA design parameter exploration. The proposed framework takes as input (1) DMA controller clock, (2) memory access clock, and (3) computation data. Based on these inputs, the framework outputs MA RTL profile generated from cycle-accurate RTL simulation. A key challenge in our proposed approach is to reduce the cost of iterative empirical MA parameter search according to precise RTL hardware cost measurement, such as simulation.
- We tried to find a pareto-optimal point of area-delay product (ADP) using the proposed framework. Therefore, we evaluate a trade-off between area and latency according to the various MA RTL generation scenarios for MA insertion to neural network computations.

2. Related Works

Accelerator Technologies and Applications in TinyML. DSORT-MCU [12] demonstrates real-time small object detection by leveraging the built-in accelerator of RISC-V core-based MCUs, with a focus on optimizing energy efficiency and latency. Deeploy [13] proposes a methodology for efficiently deploying small language models across multiple accelerators on heterogeneous MCUs, prioritizing energy efficiency. Ng et al. [14] explore mixed-precision hardware accelerators designed for FPGA-based TinyML applications, emphasizing a balance between accuracy and latency in resource-constrained environments. Tensor Processing Unit (TPU), a systolic array-based DNN accelerator, is primarily designed for cloud workloads. Its high computational requirements and lack of optimization for low-power environments make it unsuitable for TinyML applications. Therefore, Section 3.2 introduces a parameterized approach to resizing the systolic array [15]. TNN [16] is integrated into a RISC-V-based SoC, demonstrating the capability to achieve low power consumption alongside reasonable accuracy. iMCU [17] employs a 28 nm in-memory computing architecture that integrates computation and memory access, thereby minimizing data movement and maximizing energy efficiency in TinyML environments. Manor et al. [18] developed a custom hardware accelerator designed to accelerate real-time inference for TensorFlow Lite for Microcontrollers models, achieving low latency and high memory efficiency.

Frameworks for Systematic Evaluation of DNN Accelerators. CFU Playground [19] proposes a fully open-source framework for leveraging FPGAs in TinyML acceleration. It offers an integrated approach to designing hardware accelerators and software stacks, enabling developers to deploy and optimize FPGA-based TinyML models efficiently. TFApprox [20] is a framework designed for fast emulation of approximate hardware accelerators for DNNs on GPUs. It supports various approximation techniques such as quantization and pruning, allowing rapid performance evaluation of accelerators.

Frameworks for Accelerator Design and Evaluation. Gemmini [21] provides a hardware–software-coupled framework for evaluating and optimizing DNN accelerators. Based on modular hardware design integrated with RISC-V, it facilitates the analysis of performance and energy efficiency for various DNN workloads. SCALE-Sim [22] is a framework for simulating matrix multiplication-based systolic array accelerators. It enables profiling of performance and energy efficiency for CNN workloads by adjusting hardware parameters such as array size and memory hierarchy. Timeloop [23] provides a comprehensive platform for optimizing memory hierarchy and dataflow mappings of DNN accelerators. It supports custom hardware configurations and workload mapping to explore various design scenarios. Accelergy [24] introduces a framework for fast and accurate estimation of energy consumption at the architectural level for hardware accelerators. Integrated with tools like Timeloop, it supports energy-efficient design space exploration. MAESTRO [25] evaluates resource utilization across various dataflows (e.g., weight station-

ary, output stationary) in DNN accelerators, facilitating analysis of trade-offs between data reuse, performance, and hardware costs.

While simulator-based frameworks like those mentioned above provide rapid evaluation of accelerator performance under static scenarios, there is a lack of research adopting a full-system perspective that considers the interaction among CNN software, on-chip buses, and accelerator hardware. Addressing this limitation could offer significant insights into the holistic optimization of accelerator systems.

3. Micro-Accelerator-in-the-Loop Framework

Figure 1 shows the overall structure of the MAIL framework. MAIL evaluates simultaneously RTL-designed MAs used to accelerate TinyML firmware, and applications on MCU-based platforms. Input data preprocessing and output handling dependent on the MCU device in TinyML applications are functionally replicated with a high-level system emulator. The bottleneck that needs acceleration in MA is neural network computation. The MA designed in RTL simulates memory-mapped I/O access for data exchange in a cycle-accurate simulator. The MCU emulator that makes up the MAIL framework was implemented based on the open-source QEMU [26].

3.1. MA Peripheral Mixed Emulation

The MA defined in this paper is placed on-chip to act as a peripheral connected to the MCU's core bus. MA is used to accelerate TinyML software running on MCU. The on-chip peripheral is configured and triggered by software embedded in MCU flash memory. Similarly, the MA also accesses the necessary data for calculations by setting registers from instruction executions in the on-chip flash memory. However, it is laborious to integrate an MA designed in RTL into the already existing commercial MCU for evaluation. Currently, most of the cores, target of the TinyML application, protect internal RTL code as confidential for commercial purpose. Even if we implement a full MCU chip in RTL and attach MA peripheral, there will be overhead in full RTL simulation for reflecting interaction between software. A custom MA designed in RTL requires cycle-accurate simulation, while accessing or initializing the remaining peripherals is sufficient for high-abstraction-level functional emulation.

The existing project emulates the software execution of a general-purpose MCU, partially replicating memory access by instructions, general-purpose I/O (GPIO), and some interrupts. We propose the MAIL framework by modifying the emulator to emulate the process of initializing the hardware in the MCU system emulator and accessing memory-mapped registers from the TinyML firmware to the custom MA peripheral. The MAIL emulates the MA peripheral attached to the core peripheral bus for Arm core-based MCU. MCU manufacturers using Arm cores provide memory-mapped register address regions and register bitfield information for all peripherals present in the MCU with the system view description (SVD) format. The MAIL adopted a QEMU structure that parses SVD and automatically generates a template for each peripheral that converts firmware-level register access to an emulator program. Therefore, we added the MA peripheral register information to the SVD file and enabled MA access with the TinyML firmware instruction in the emulator, as shown in Figure 2.

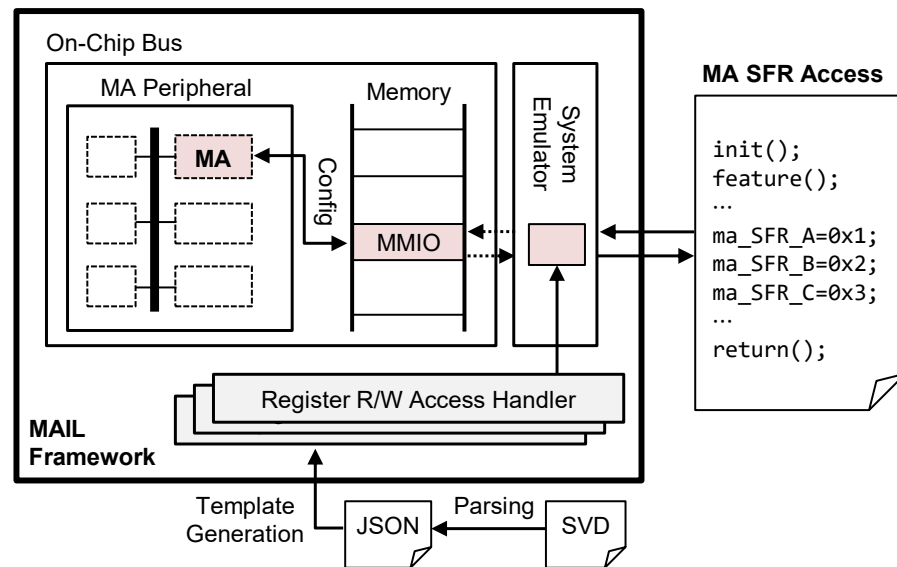


Figure 2. Parse from SVD files to automatically generate MA special function register handlers. Register handlers receive MA register access request occurred in TinyML application and generates configurable MA peripheral inside microcontroller emulator.

3.2. MA RTL Generation and Cycle-Accurate Simulation

If the MA peripheral registers are accessible in the MAIL framework, we can define the emulator behavior when a read or write access to each register occurs at the firmware level. Like any other peripheral, the MA also triggers computations by writing enable values to memory-mapped registers. At this time, the MAIL framework remotely executes an external RTL simulator to cycle-accurately evaluate the operations performed within the generated MA. The MAIL includes a MA RTL generation engine for flexible and scalable MA evaluation when running RTL simulators remotely. The MAIL's RTL simulator receives information about the MA, e.g., channel, width, and height, from the system emulator part, generates the MA RTL design, and runs the simulation. Since the parameters that define the MA RTL structure are specific to the TinyML application, a register must be defined to connect the firmware layer and the emulator layer. This register includes the remote execution code required for the RTL simulator. When the TinyML firmware writes a parameter to the MA's SFR, the emulator transfers the written register value to the RTL simulator. When transferring SFR values for MA to the RTL simulation, a file format is utilized. Among MA's SFRs, the Enable field in the CR register triggers a callback to remotely execute the RTL simulator. Based on the SFR values written to the file, the RTL simulator determines the data stream input policy for MA and executes the simulation accordingly. MAIL evaluates the clock latency of using MA on Cortex-M4 architectures.

Figure 3 shows the process by which the MA RTL is generated according to the neural network model information written in the MA's four register fields. A four 32-bit area for MA is reserved in the existing MCU register empty space. The MA RTL structure and the DMA controller model are generated by the configured values in the register. The purpose of the MAIL framework is to evaluate the performance of the MA when used in TinyML firmware, rather than to obtain accurate classification results through the inference of actual input data. Therefore, the actual data are not transferred to the MA's INPUT, OUTPUT, and WEIGHT registers, and only the column and row sizes of the matrix are passed to the RTL simulator to reconstruct the matrix operation with dummy data. The INPUT and WEIGHT registers of the MA are used to define the matrix size when generating the MA RTL. In the 32-bit register, the most significant 16 bits (MSB) represent the row size, while the least significant 16 bits (LSB) represent the column size. At this moment, the neural network

components supported by MA in MAIL framework are convolution neural network (CNN) and fully connected (FC) layer. Model information is also written in the CR register because the structure of the generated RTL varies depending on the type of neural operation. The remaining fields in the CR register are values to model clock timing due to DMA request protocols and memory accesses between MA and DMA (discussed in Section 3.3).

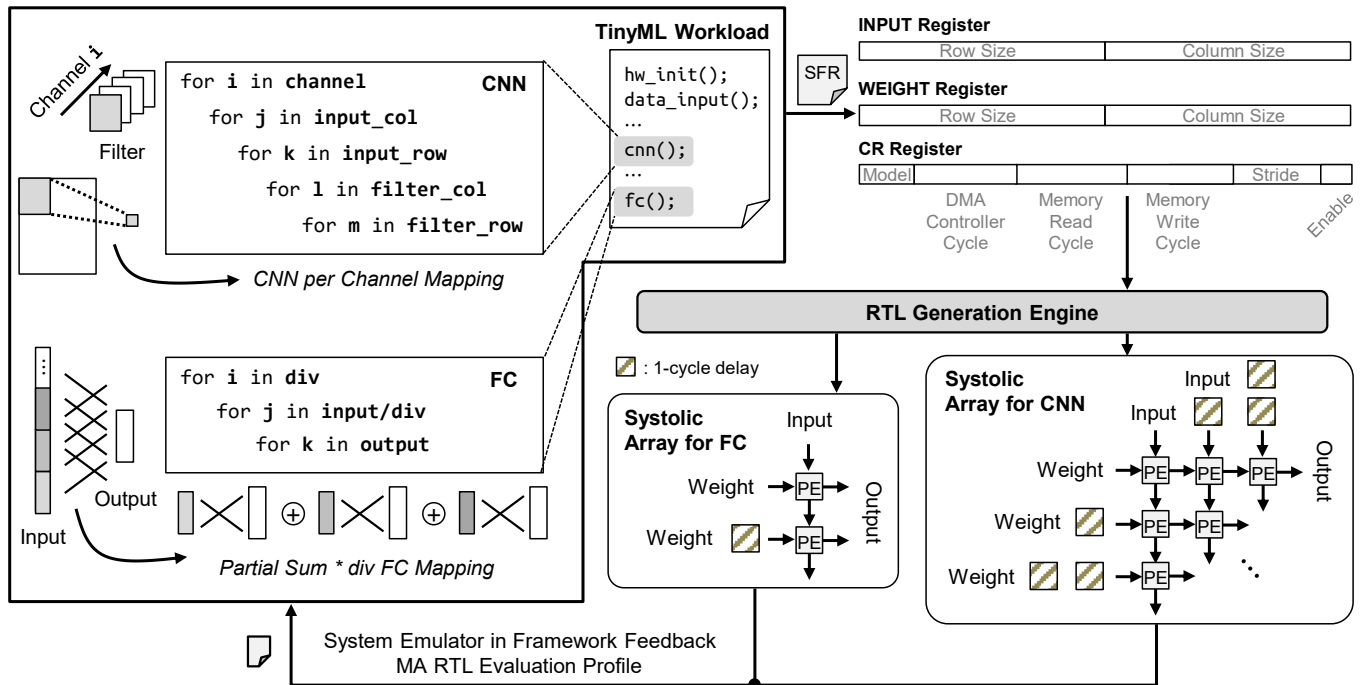


Figure 3. MAIL framework MA RTL generation scheme: FC and CNN layer. Generated MA RTL pass through cycle-accurate RTL evaluation and feedback profile result to system emulator.

The configurations written to the MA SFR are used to generate the MA RTL and surrounding testbench. Currently, MAIL supports two fundamental neural computations: FC and CNN. The RTL generation engine uses conventional systolic arrays to create accelerators [27]. There are three registers inside the processing element (PE). The weight is passed from the left side to the right side, and the input is passed from the top to the bottom. The processing element then performs a multiply-and-accumulate (MAC) operation using the weight and input values. The MA generated for the FC model has as many PEs as the number of output nodes and computes the MAC of weight and input for each cycle. There are clock cycle delays, depending on the column index of weight. The MAC result for each PE is the output when all inputs and weights were consumed at the PEs. The MA generated for the CNN model, the PE matrix is generated by the product of the output column and row made by the filter and the filter column. A CNN MA sums each PE in a row into one output value. Similar to FC MA, a delay written to the register is used for weights and inputs flowing into the PE matrix. We employed a conventional systolic accelerator structure, rather than state-of-the-art designs, for the RTL generation process implemented by MAIL. Because the scope of this paper is not the accelerator design but rather the introduction of a framework that enables performance evaluation by linking MA and TinyML applications.

The internal PE structure of the systolic array generated as MA's RTL is illustrated in Figure 4. Considering the characteristics of TinyML, we assumed a typical systolic array rather than a custom accelerator architecture dedicated to specific applications. Therefore, MA utilizes PEs that employ a weight-stationary dataflow. The weights used for MAC operations in each PE are preloaded from a buffer. The input data, including activations,

flow from top to bottom, simultaneously being used for multiplication with the weights and passing through to the next PE in the downward direction. The partial sum required to generate the output flows from left to right.

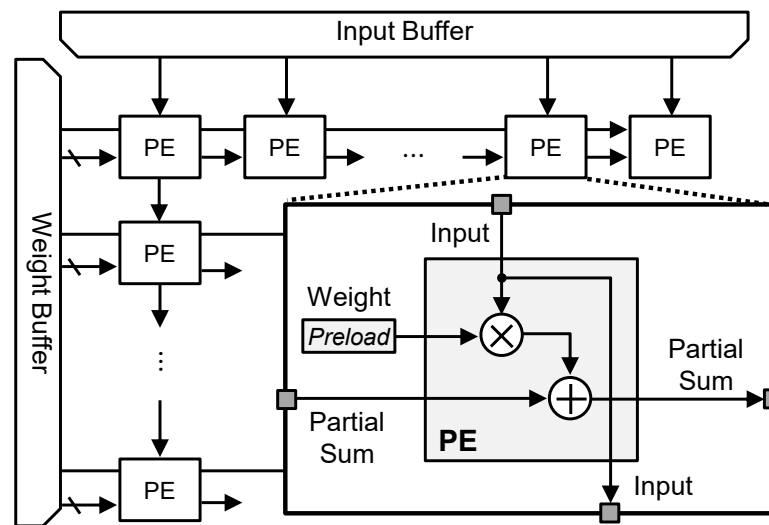


Figure 4. MA internal PE's weight and input buffer connection structure and dataflow direction.

3.3. MA and MCU Data Interaction

In this paper, MA is defined as peripherals connected to MCU core peripheral bus. Access to this peripheral is through SFRs mapped to memory addresses. To connect the MA to the core as a peripheral, we need to map a memory address space with registers that enable TinyML software to configure the MA hardware. Registers are used to trigger the MA operation start or to exchange data required for computations. However, for neural network computation acceleration, a bulk of data transfer is required due to the large amount of weights, activations, and output values. Conventional accelerator users in on-cloud AI rarely need to consider physical memory addresses when transferring large amounts of data between the device and host, as the operating system's virtual memory technique efficiently manages these transfers. However, MCU has memory constraints compared to PC and server. Therefore, memory size constraints are important when TinyML software uses SFR to transfer data required for computational processes from memory to MA peripheral devices. If all the data used by the MA were mapped as register addresses in memory, it would exceed the physical memory address range of the MCU. Therefore, it is inevitable to use DMA controllers to access the data that MA needs for neural network computations, and vice versa.

Figure 5 represents the data transfer modeling of MA connected to the peripheral bus. When DMA transfers bulk data without CPU intervention, iterative transfer is required because of the size of data that can be handled at once is fixed. Simultaneously, the source or destination address of data can be fixed or auto incremented. Because, as mentioned above, MA requires constrained register sector space, the DMA accesses with a single peripheral address, and when a peripheral bus access occurs from DMA, the internal buffer address is increased within MA. When the DMA stores data into the MA's input and weight buffers, and loads data from the output buffer, the MA can transfer sequential data over the same peripheral address access. Such a DMA operation model is defined in the MA's CR register. When the MA is enabled by SFR configuration, DMA request is set to fill input and weight buffer. DMAC Req. Clock Timing is the clock parameter until DMA respond data to the peripheral bus in this request. Memory Read Clock, Memory Write Clock are the clock parameters required to access memory in continuous DMA operations. The DMA

configuration parameters are used for testbench generation that models the data transfer process in MA and peripheral bus protocols. When a data transfer request is generated from the MA, the data used for neural calculation is supplied to the MA or transferred to the emulator, depending on the behavior of the configured DMA controller.

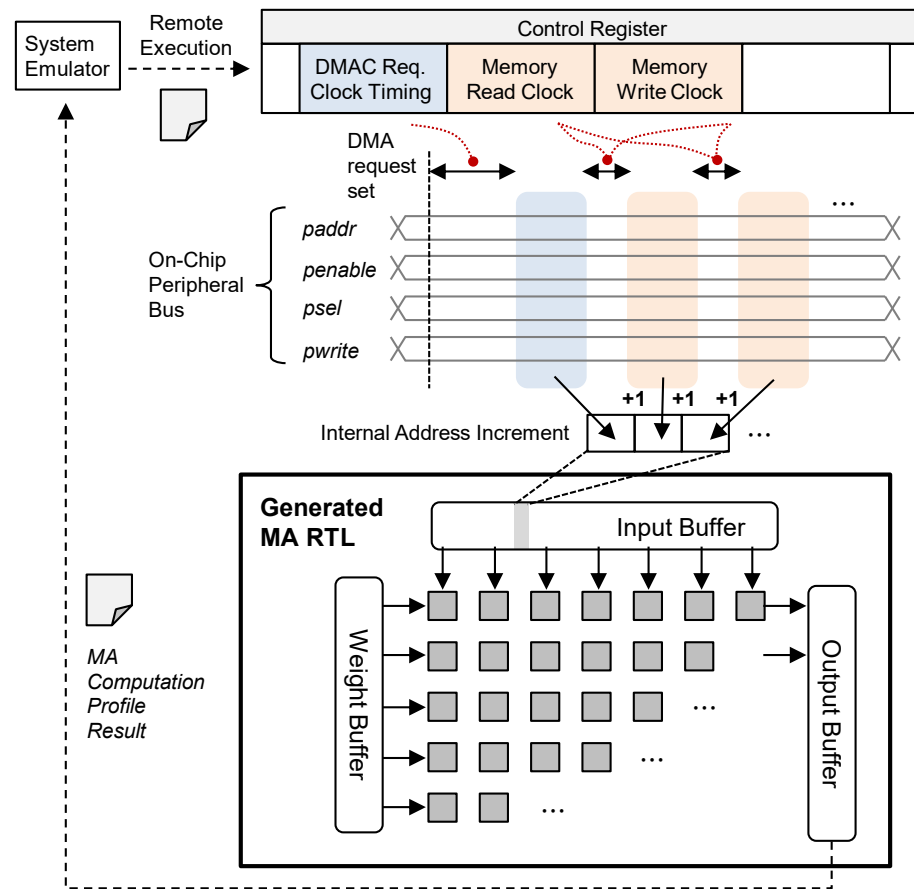


Figure 5. MA peripheral and DMA controller model interaction in MAIL. The DMA controller model is generated as a testbench to instantiate the MA RTL.

4. Evaluation

We select two TinyML cases to evaluate the MAIL framework, consisting of fundamental FC and CNN computations. The first case is a speech recognition application using the TinyConv model architecture [28]. It consists of one CNN and one FC layer and outputs the classification result using a spectrogram. The second case is a gesture recognition (Gesture) model using a 3-axis accelerometer [29]. The detailed structure of each model is illustrated in Figure 6. The max pooling operation is used only in the second Gesture model in Figure 6b. It consists of two pairs of CNN and FC layers and includes a maxpooling layer. The two workloads used in the evaluation are representative case models used in TFLM. These neural network models were selected as evaluation workloads because they are considered representative cases of TinyML. We performed int8 linear quantization to deploy the model on MCU and simplify the MA RTL computation. In addition, all activations use int16 format. We deployed the model to STM32F407 (Cortex-M4, 128 kB SRAM/1 MB Flash) for MAIL and physical MCU comparison. The STM32F407 board uses the STM32F407VGT6 MCU. The MCU is designed based on the Arm Cortex-M4 (CM4), which is a mid-performance Cortex lineup between the CM0 and CM7. We selected the CM4 as the target for evaluation because it can achieve a balance between performance and power efficiency, making it suitable for embedded systems. The STM32F407 clock

configuration was chosen to be 168 MHz and used for latency normalization. We utilized a Cadence NCVerilog as the RTL simulator within MAIL for the experiment.

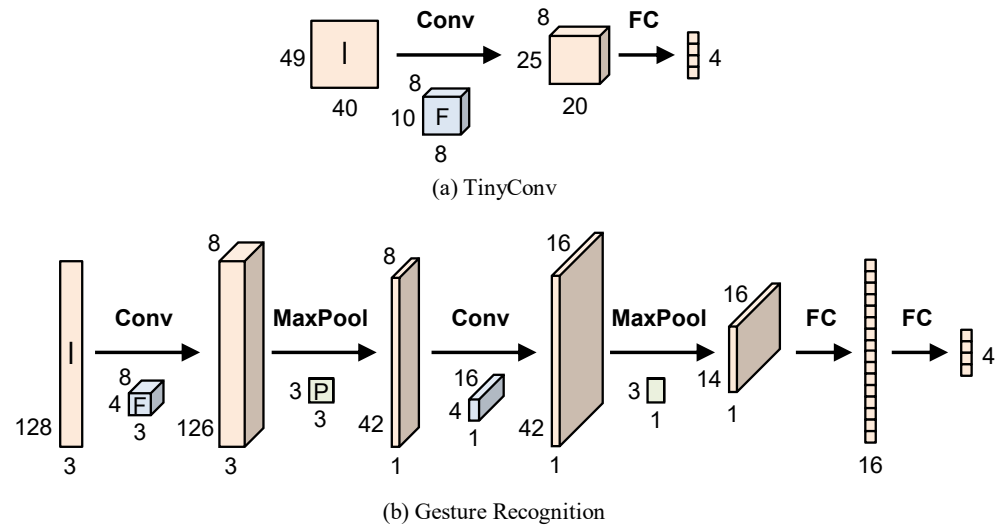


Figure 6. Model architecture of the TinyML applications used in the evaluation: (a) TinyConv for speech recognition, (b) gesture recognition.

The TinyML inference time includes not only the FC and CNN layers, but also the pooling and softmax operations. Table 1 compared the proportions of the CNN, FC operation and the remaining operations in the evaluation cases. The results of the elapsed time ratio of the part excluding the FC, CNN in two cases are under 1% and 6.7%, respectively. The proportion of calculations other than CNN and FC in the entire TinyML process can be negligible. Therefore, when we compared the acceleration result using MA in the MAIL framework with the existing TinyML inference library, we only compared the CNN and FC calculation parts.

Table 1. Executed TinyML models on STM32F407 device using TFLM and measured elapsed time. Calculate the proportion of elapsed time for each type of neural network computation.

Model	Speech Recognition		Gesture Recognition	
	Elapsed Time (μ s)	Ratio (%)	Elapsed Time (μ s)	Ratio (%)
conv2d	23,049	99	4882	93.2
fc	302		91	
pooling	-	<1	336	6.7
non-linear	22		23	

Figure 7 compares the cycle latency result obtained from TinyML software executed by baremetal, existing inference library and MAIL by replacing MA within neural computation. The inference library used to run TinyML is X-Cube-AI [30], which is inference library for 32 bits STMicro MCUs, and TFLM [6]. In order to insert MA, we have to decompose the atomic neural calculations of the internal source code, we inserted the MA register enable code into the CNN and FC parts in baremetal instead of the existing libraries. The graphs in Figure 7 plot the cycle latency for each CNN, FC layer for the two cases. DMAC Req. Clock Timing, Memory Read Clock, Memory Write Clock are selected at 5, 3, 3 clock. conv2d, fc denote a CNN, FC layers within the first application, conv2d₁, conv2d₂, fc₁, fc₂ denote a pair of CNN, FC layers within the second application. If the MA is inserted in the CNN layer, it replaces the convolution of the single-channel input and the filter, and if it is inserted in the FC layer, it replaces the input vector and weight matrix multiplication.

As a result of comparing the clock latency in each layer, the CNN layer decreased the clock latency up to 94.25%. These MAIL framework results help determine the MAs configuration used in TinyML. Table 2 shows the combined cycle latency across layers compared to other existing libraries. Despite the cycle latency overhead in the FC layer, the reduction in the CNN layer was dominant, decreasing by -84.32% and -61.32% in the first and second cases, respectively.

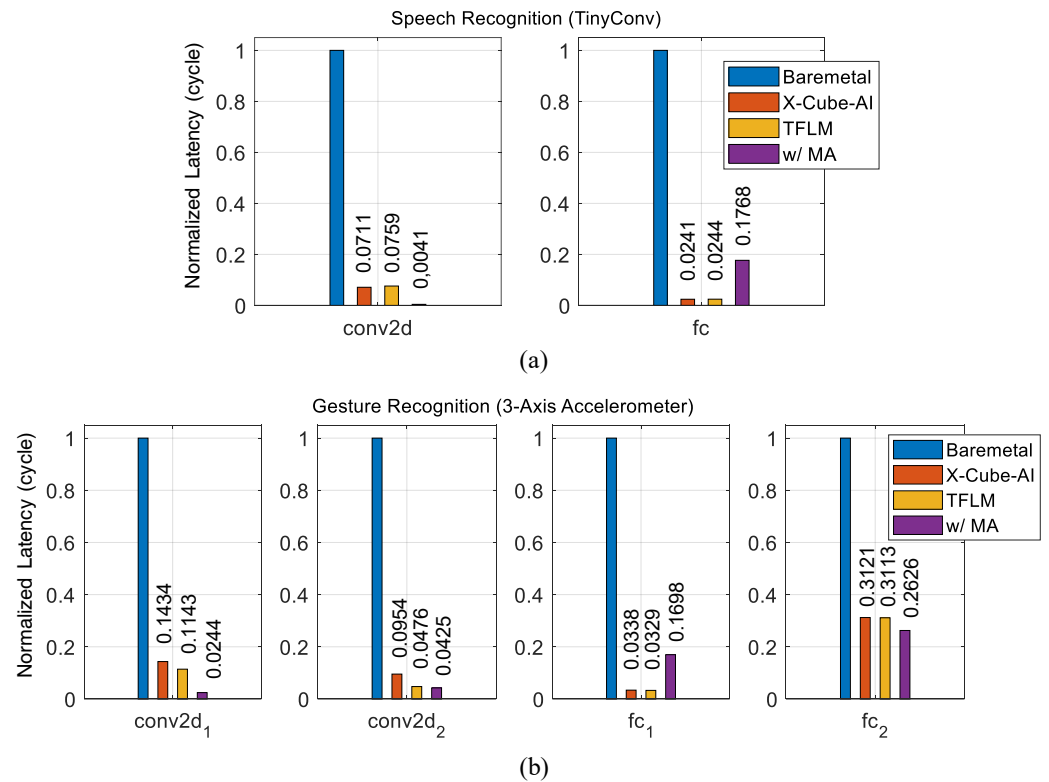


Figure 7. MAIL framework cycle latency measurement for each neural architecture layer of the MA inserted TinyML applications: (a) speech recognition; (b) gesture recognition.

Table 2. Measurement of the neural network total cycle latency when using baremetal firmware, existing inference libraries implemented for TinyML application, and MA inserted TinyML firmware. Total cycle latency reduction was compared to the smallest value, e.g., X-Cube-AI selected in speech recognition application to comparison.

Model	Total Latency (Cycle)	
	Speech Recognition	Gesture Recognition
Baremetal	53,099,517	9,298,534
X-Cube-AI	3,677,731	1,151,904
TFLM	3,959,653	895,776
MA (ours)	576,399 (-84.32%)	346,534 (-61.32%)

In the FC layer with input and output nodes, the delay due to iterative MA enable varies according to the area size that supports the input vector and weight matrix product. Hardware accelerators inherently exhibit a trade-off between performance and resource utilization. Therefore, when evaluating MA, it is essential to consider both performance, represented by delay, and resource utilization, such as hardware area, simultaneously. To address this, we decided to use the area-delay product (ADP) metric, calculated as the product of hardware area and delay. The designer utilizing MAIL can rapidly conduct design exploration based on the prototype’s ADP. Figure 8 shows the ADP results with the

area and cycle latency of the generated MA RTL for the MA coverage size sweep of the FC layer computation. We used Synopsys Design Compiler as the synthesis tool to measure the area of the MA RTL. The first case has 4000×4 size FC layer fc , and we generated a size of MA that divides input nodes into 1, 5, 10, 50, 100, 250, 500 times. When these MA replaced into the TinyML software, we measured the clock latency at MAIL due to repeated calls, and calculated the ADP between the synthesized area from generated MA RTL. Second case has 224×16 size FC layer fc_1 . We varied the divide values to 1, 7, 8, 14, 28 and measured ADP by MA. The results show that iteratively using smaller-sized MAs to accumulate the output achieves better ADP than increasing the area based on the acceptable size of the input vector and weight matrix.

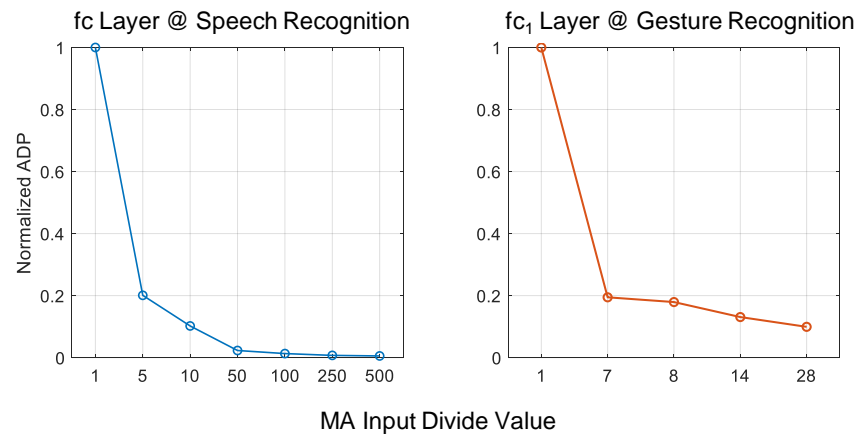


Figure 8. ADP of various MA insertion results by computation scenario of TinyML application FC layer. Iterative calls from divided MAs achieve better ADP.

Table 3 compares the state-of-the-art accelerator prototyping frameworks with the proposed MAIL framework. Ref. [21] targets RISC-V SoC and controls accelerators via MMIO or custom ISA, while [22] is designed for functional profiling of standalone CNN accelerators. Unlike other frameworks that provide only one approach—either RTL simulation or function modeling—based on their abstraction level, MAIL offers a mixed level of abstraction. Unlike the other two frameworks, MAIL targets MCUs and therefore does not utilize DRAM. The workload in [21] replaces neural network models with matrix multiplication due to the substantial time required for full RTL simulation. In the evaluation, all cases used a 16×16 systolic array. The elapsed time measured for each case includes the duration from the framework's initiation to the completion of accelerator operations (either software-controlled or standalone), profiling of the results, and the termination of the framework. The evaluation results demonstrate that MAIL can perform cycle-accurate RTL simulations for hardware accelerator design while significantly reducing simulation time. Consequently, MAIL enables rapid design exploration of accelerators, considering both performance and resource usage.

As we take into account these evaluation results, the MAIL framework makes it easy to iteratively profile hardware and software interactions when using peripheral-level accelerator MAs in various TinyML scenarios. In addition, MAIL enables effect calculation by inserting MA in TinyML software during continuous framework execution flow. Therefore, MAIL mitigates RTL design and embedded software complexity for peripheral-level accelerator evaluate.

Table 3. Comparison of other state-of-the-art accelerator prototyping frameworks with MAIL.

	Gemmini [21]	SCALE-Sim [22]		MAIL (Ours)	
		TinyConv	Gesture	TinyConv	Gesture
Target System	RISC-V SoC	N/A ¹		Arm SoC	
Abstraction-level	Low (RTL)	High (Functional Model)		Mixed	
DRAM Usage	✓	✓		✗	
RTL Simulator	Verilator	N/A ²		Cadence NCVerilog	
Workload	MatMul ³	Conv + FC	Conv + FC	Conv + FC	Conv + FC
Model Architecture	(16, 16) × (16, 16)	Figure 6a	Figure 6b	Figure 6a	Figure 6a
Systolic Array Size	(16, 16)	(16, 16)		(16, 16)	
Elapsed Time (s) ⁴	130.53	2.54	0.89	23.74	20.64

¹ Standalone accelerator model that cannot load operations. ² Not including RTL part. ³ The result represents the average calculated from 10 measurements. ⁴ Matrix Multiplication.

5. Conclusions and Future Work

In this paper, we have defined the accelerator concept, which is connected to the core peripheral bus, MA. In addition, we propose a MAIL framework that can perform an acceleration profile of MA inserted TinyML applications. MAIL is a coupled architecture of system emulators and RTL simulators designed to handle the complex register configuration flows required for embedded software and neural computational acceleration using MA. The MAIL framework offers practical value by enabling cost-effective and comprehensive evaluation prior to integrating accelerators into MCUs designed for edge AI and IoT applications. The MAIL framework addresses hardware area and delay for accelerators connected via MMIO. However, in resource-constrained systems like MCUs, energy consumption is also a critical factor, necessitating evaluation in this dimension. In future work, we plan to develop an energy consumption model for the MAIL framework to provide energy profiling for accelerators utilized in MCUs. We profiled performance and ADP by inserting MA into various neural computations of TinyML applications, and verified the effectiveness of MAIL, which makes TinyML software combined with MA evaluation accessible.

Author Contributions: Entire core architecture designing and performing the numerical analysis, J.K.; corresponding author, D.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Brain Korea 21 4th Generation (BK21 FOUR) Project 4199990113966, in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (10%, 10%, respectively) under Grant NRF-2018R1A6A1A03025109, and Grant NRF-2022R1I1A3069260, in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) funded by Korean Government [Ministry of Science and Information Communication Technology (MSIT)] through the Processing-in-Memory Computing (PIM) Semiconductor Design Research Center (30%) under Grant 2022-0-01170, in part by the Development of Flexible Software-Hardware (SW-HW) Conjunctive Solution for On-Edge Self-Supervised Learning (50%) under Grant RS-2023-00228970.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original data presented in the study are openly available in tflite-micro at <https://github.com/tensorflow/tflite-micro.git> (accessed on 21 November 2024).

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

MA	micro-accelerator
SDK	software development kit
ML	machine learning
MCU	microcontroller unit
SFR	special function register
DMA	direct memory access
FPGA	field programmable gate array
CPU	central processing unit
GPU	graphics processing unit
CNN	convolutional neural network
FC	fully-connected
RTL	register transfer level
PE	processing element
ADP	area-delay product
SVD	system view description

References

- Nurvitadhi, E.; Sheffield, D.; Sim, J.; Mishra, A.; Venkatesh, G.; Marr, D. Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC. In Proceedings of the 2016 International Conference on Field-Programmable Technology (FPT), Xi'an, China, 7–9 December 2016; pp. 77–84. [\[CrossRef\]](#)
- Xiong, Y.; Liu, H.; Gupta, S.; Akin, B.; Bender, G.; Wang, Y.; Kindermans, P.J.; Tan, M.; Singh, V.; Chen, B. MobileDets: Searching for Object Detection Architectures for Mobile Accelerators. In Proceedings of the 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, 20–25 June 2021; pp. 3824–3833. [\[CrossRef\]](#)
- Chen, Y.H.; Yang, T.J.; Emer, J.; Sze, V. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2019**, *9*, 292–308. [\[CrossRef\]](#)
- Saha, S.S.; Sandha, S.S.; Srivastava, M. Machine Learning for Microcontroller-Class Hardware: A Review. *IEEE Sens. J.* **2022**, *22*, 21362–21390. [\[CrossRef\]](#) [\[PubMed\]](#)
- Novac, P.E.; Boukli Hacene, G.; Pegatoquet, A.; Miramond, B.; Gripon, V. Quantization and Deployment of Deep Neural Networks on Microcontrollers. *Sensors* **2021**, *21*, 2984. [\[CrossRef\]](#) [\[PubMed\]](#)
- Google Inc. TensorFlow Lite for Microcontrollers. Available online: <https://github.com/tensorflow/tflite-micro> (accessed on 21 November 2024).
- Lin, J.; Chen, W.M.; Cohn, J.; Gan, C.; Han, S. MCUNet: Tiny Deep Learning on IoT Devices. In Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), Online, 6–12 December 2020.
- Banbury, C.; Zhou, C.; Fedorov, I.; Matas, R.; Thakker, U.; Gope, D.; Janapa Reddi, V.; Mattina, M.; Whatmough, P. MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers. In Proceedings of the Machine Learning and Systems (MLSys), Online, 5–9 April 2021.
- Kwon, H.; Chatarasi, P.; Sarkar, V.; Krishna, T.; Pellauer, M.; Parashar, A. MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings. *IEEE Micro* **2020**, *40*, 20–29. [\[CrossRef\]](#)
- Seok, M.G.; Sarjoughian, H.S.; Park, D. A High-Level Modeling and Simulation Approach Using Test-Driven Cellular Automata for Fast Performance Analysis of RTL NoC Designs. In Proceedings of the 24th Asia and South Pacific Design Automation Conference, ASPDAC'19, Tokyo, Japan, 21–24 January 2019; pp. 382–387. [\[CrossRef\]](#)
- Kwon, J.; Oh, S.; Park, D. Metamorphic Edge Processor Simulation Framework Using Flexible Runtime Partial Replacement of Software-Embedded Verilog RTL Models. In Proceedings of the 2021 IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, Republic of Korea, 22–28 May 2021; pp. 1–5. [\[CrossRef\]](#)
- Boyle, L.; Moosmann, J.; Baumann, N.; Heo, S.; Magno, M. DSORT-MCU: Detecting Small Objects in Real-Time on Microcontroller Units. *IEEE Sens. J.* **2024**, *24*, 40231–40239. [\[CrossRef\]](#)
- Scherer, M.; Macan, L.; Jung, V.J.B.; Wiese, P.; Bompani, L.; Burrello, A.; Conti, F.; Benini, L. DeepDeploy: Enabling Energy-Efficient Deployment of Small Language Models on Heterogeneous Microcontrollers. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2024**, *43*, 4009–4020. [\[CrossRef\]](#)

14. Ng, W.S.; Goh, W.L.; Gao, Y. High Accuracy and Low Latency Mixed Precision Neural Network Acceleration for TinyML Applications on Resource-Constrained FPGAs. In Proceedings of the 2024 IEEE International Symposium on Circuits and Systems (ISCAS), Singapore, 19–22 May 2024. [CrossRef]
15. Jouppi, N.P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; et al. In-Datcenter Performance Analysis of a Tensor Processing Unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA '17, Toronto, ON, Canada, 24–28 June 2017; pp. 1–12. [CrossRef]
16. Scherer, M.; Di Mauro, A.; Rutishauser, G.; Fischer, T.; Benini, L. A 1036 TOp/s/W, 12.2 mW, 2.72 uJ/Inference All Digital TNN Accelerator in 22 nm FDX Technology for TinyML Applications. In Proceedings of the 2022 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS), Tokyo, Japan, 20–22 April 2022; pp. 1–3. [CrossRef]
17. Lin, C.T.; Huang, P.X.; Oh, J.; Wang, D.; Seok, M. iMCU: A 28-nm Digital In-Memory Computing-Based Microcontroller Unit for TinyML. *IEEE J. Solid State Circuits* **2024**, *59*, 2684–2693. [CrossRef]
18. Manor, E.; Greenberg, S. Custom Hardware Inference Accelerator for TensorFlow Lite for Microcontrollers. *IEEE Access* **2022**, *10*, 73484–73493. [CrossRef]
19. Prakash, S.; Callahan, T.; Bushagour, J.; Banbury, C.; Green, A.V.; Warden, P.; Ansell, T.; Reddi, V.J. CFU Playground: Full-Stack Open-Source Framework for Tiny Machine Learning (TinyML) Acceleration on FPGAs. In Proceedings of the 2023 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2023, Raleigh, NC, USA, 23–25 April 2023; pp. 157–167. [CrossRef]
20. Vaverka, F.; Mrazek, V.; Vasicek, Z.; Sekanina, L. TFApprox: Towards a Fast Emulation of DNN Approximate Hardware Accelerators on GPU. In Proceedings of the 2020 Design, Automation Test in Europe Conference Exhibition (DATE), Grenoble, France, 9–13 March 2020; pp. 294–297. [CrossRef]
21. Genc, H.; Kim, S.; Amid, A.; Haj-Ali, A.; Iyer, V.; Prakash, P.; Zhao, J.; Grubb, D.; Liew, H.; Mao, H.; et al. Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration. In Proceedings of the 2021 58th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 5–9 December 2021; pp. 769–774. [CrossRef]
22. Samajdar, A.; Joseph, J.M.; Zhu, Y.; Whatmough, P.; Mattina, M.; Krishna, T. A Systematic Methodology for Characterizing Scalability of DNN Accelerators using SCALE-Sim. In Proceedings of the 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Boston, MA, USA, 23–25 August 2020; pp. 58–68. [CrossRef]
23. Parashar, A.; Raina, P.; Shao, Y.S.; Chen, Y.H.; Ying, V.A.; Mukkara, A.; Venkatesan, R.; Khailany, B.; Keckler, S.W.; Emer, J. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In Proceedings of the 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Madison, WI, USA, 24–26 March 2019; pp. 304–315. [CrossRef]
24. Wu, Y.N.; Emer, J.S.; Sze, V. Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs. In Proceedings of the 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Westminster, CO, USA, 4–7 November 2019; pp. 1–8. [CrossRef]
25. Kwon, H.; Chatarasi, P.; Pellauer, M.; Parashar, A.; Sarkar, V.; Krishna, T. Understanding reuse, performance, and hardware cost of DNN dataflows: A data-centric approach. In Proceedings of the MICRO '52: 52nd Annual IEEE/ACM International Symposium on Microarchitecture, Columbus, OH, USA, 12–16 October 2019; pp. 754–768. [CrossRef]
26. [xPack QEMU Arm]. Available online: <https://xpack.github.io/qemu-arm/> (accessed on 21 November 2024).
27. Wang, S.; Zhou, D.; Han, X.; Yoshimura, T. Chain-NN: An energy-efficient 1D chain architecture for accelerating deep convolutional neural networks. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, Lausanne, Switzerland, 27–31 March 2017; pp. 1032–1037. [CrossRef]
28. TensorFlow Lite for Microcontrollers (Micro Speech Example). Available online: https://github.com/tensorflow/tflite-micro/tree/1b68490dcfd087c308a1f8d14012ab1cb39c9ac0/tensorflow/lite/micro/examples/magic_wand (accessed on 21 November 2024).
29. TensorFlow Lite for Microcontrollers (Magic Wand Example). Available online: https://github.com/tensorflow/tflite-micro/tree/main/tensorflow/lite/micro/examples/micro_speech (accessed on 28 March 2023).
30. STMicroelectronics. AI Expansion Pack for STM32CubeMX. Available online: <https://www.st.com/en/embedded-software/x-cube-ai.html> (accessed on 21 November 2024).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.