frontiers | Frontiers in Electronics

TYPE Original Research
PUBLISHED 11 August 2022
DOI 10.3389/felec.2022.898273



OPEN ACCESS

EDITED BY
Deming Chen,
University of Illinois at Urbana-Champaign, United States

REVIEWED BY
Jae-sun Seo,
Arizona State University, United States
Yu Cao,
Arizona State University, United States
Shanshi Huang,
Georgia Institute of Technology, Atlanta, United States

*CORRESPONDENCE
Vaibhav Verma,
vv8dn@virginia.edu

SPECIALTY SECTION
This article was submitted to Integrated Circuits and VLSI, a section of the journal Frontiers in Electronics

RECEIVED 17 March 2022
ACCEPTED 05 July 2022
PUBLISHED 11 August 2022

CITATION
Verma V and Stan MR (2022), AI-PiM—Extending the RISC-V processor with Processing-in-Memory functional units for AI inference at the edge of IoT. *Front. Electron.* 3:898273.
doi: 10.3389/felec.2022.898273



# AI-PiM—Extending the RISC-V processor with Processing-in-Memory functional units for AI inference at the edge of IoT

Vaibhav Verma* and Mircea R. Stan

Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA, United States

The recent advances in Artificial Intelligence (AI) achieving "better-than-human" accuracy in a variety of tasks such as image classification and the game of Go have come at the cost of exponential increase in the size of artificial neural networks. This has lead to AI hardware solutions becoming severely memory-bound and scrambling to keep-up with the ever increasing "von Neumann bottleneck". Processing-in-Memory (PiM) architectures offer an excellent solution to ease the von Neumann bottleneck by embedding compute capabilities inside the memory and reducing the data traffic between the memory and the processor. But PiM accelerators break the standard von Neumann programming model by fusing memory and compute operations together which impedes their integration in the standard computing stack. There is an urgent requirement for system-level solutions to take full advantage of PiM accelerators for end-to-end acceleration of AI applications. This article presents AI-PiM as a solution to bridge this research gap. AI-PiM proposes a hardware, ISA and software co-design methodology which allows integration of PiM accelerators in the RISC-V processor pipeline as functional execution units. AI-PiM also extends the RISC-V ISA with custom instructions which directly target the PiM functional units resulting in their tight integration with the processor. This tight integration is especially important for edge AI devices which need to process both AI and non-AI tasks on the same hardware due to area, power, size and cost constraints. AI-PiM ISA extensions expose the PiM hardware functionality to software programmers allowing efficient mapping of applications to the PiM hardware. AI-PiM adds support for custom ISA extensions to the complete software stack including compiler, assembler, linker, simulator and profiler to ensure programmability and evaluation with popular AI domain-specific languages and frameworks like TensorFlow, PyTorch, MXNet, Keras etc. AI-PiM improves the performance for vector-matrix multiplication (VMM) kernel by 17.63x and provides a mean speed-up of 2.74x for MLPerf Tiny benchmark compared to RV64IMC RISC-V baseline. AI-PiM also speeds-up MLPerf Tiny benchmark inference cycles by 2.45x (average) compared to state-of-the-art Arm Cortex-A72 processor.

KEYWORDS

**processing-in-memory, PIM, artificial intelligence hardware, AIHW, TinyML, RISC-V, custom instruction extensions, IoT edge**

## 1 Introduction

There has been an incessant increase in the demand for artificial intelligence (AI) and machine learning (ML) based systems ever since ML models like He et al. (2016) achieved better-than-human accuracy ($\approx 95\%$) in the ImageNet Large Scale Visual Recognition Challenge (Russakovsky et al. (2015)) and Deep Neural Networks (DNN) like AlphaGo (Silver et al. (2016)) defeated the human Go champions. These large Artificial Neural Networks (ANNs) were originally designed to be hosted on big compute clusters consisting of server CPUs and GPUs. But in the last few years, AI applications have been expanding away from the cloud and towards the edge of Internet of Things (IoT). Low-cost and low-power microprocessors and microcontroller units (MCU) are being targeted to process edge AI applications in order to achieve the goal of truly intelligent IoT and this class of computing has been referred to as tinyML (Warden and Situnayake (2019)). The goal of tinyML devices and edge AI in general is to process the data close to data generation sources like IoT sensors and reduce the amount of data that is transferred to the cloud. Edge AI offers enhanced security and privacy by processing the user data in individual edge consumer devices. This local processing of data also improves the quality of service (QoS) in areas with no network coverage, latency of response and immensely reduces the cost and bandwidth requirements for communicating large amounts of data from edge devices to the cloud.

RISC-V (Waterman et al. (2014)) and Arm are the two most widely used Instruction Set Architectures (ISA) for tinyML and edge hardware devices. But open-source RISC-V ISA offers the added advantage of lowering the cost of edge IoT hardware by removing the license royalty costs associated with other edge processor architectures like Arm. Many AI accelerators have also been proposed in literature (Chen et al. (2016); Jouppi et al. (2017); Yazdanbakhsh et al. (2021); Anderson et al. (2021)) to meet the demands of compute and memory-intensive AI workloads like Deep Neural Networks (DNN). Processing-in-memory (PiM) based accelerators are one such class of AI accelerators which are particularly well-suited to alleviate the memory-wall problem (Moyer (1991); Wulf and McKee (1995)) (also known as von Neumann bottleneck) by minimizing the movement of data between the processor and the memory while accelerating the computation of AI workloads at the edge. PiM accelerators can be based on different memory technologies like SRAM (Eckert et al. (2018); Dong et al. (2020); Zhang et al. (2017)), DRAM (Roy et al. (2021); Seshadri et al. (2017); Hajinazar et al. (2021)) or emerging memories like RRAM (Chi et al. (2016); Li et al. (2020); Chou et al. (2019)) and can be of different sizes depending on the underlying memory technology. State-of-the-art PiM AI accelerators fuse memory and compute functionality together and hence deviate from the standard von Neumann architecture. This makes PiM accelerators hard to program and difficult to integrate in the standard computing stack since they do not follow the traditional programming model. These accelerators require special compilers to orchestrate the data movement to and from the accelerator (Li et al. (2020); Jia et al. (2022)) and each new design of the accelerator necessitates the design of a new compiler. Standard C/C++ and deep learning compilers struggle to map computations effectively onto PiM accelerators without an intermediate special PiM compiler. Hence, to fully utilize the advantages of PiM accelerators for AI applications at the edge of IoT, a standard design methodology is required to enable integration of PiM accelerators in the traditional computing stack.

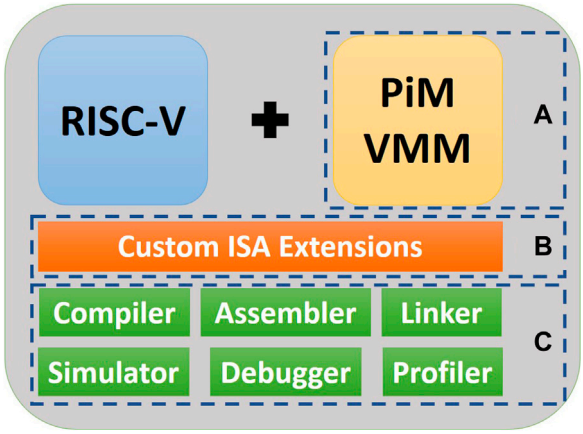


FIGURE 1
Top-level overview of AI-PiM hardware, ISA and software co-design methodology. **(A)** RISC-V hardware is extended with the addition of PiM based Vector-Matrix Multiplication (VMM) functional unit. **(B)** Custom instruction extensions are added to the RISC-V ISA to expose the VMM functionality of the PiM hardware to the programmers. **(C)** Complete Software Development Kit (SDK) including compiler, simulator etc. is then extended to communicate the hardware and ISA enhancements to the software stack.

In this article, we present AI-PiM as a hardware, ISA and software co-design solution to bridge this research gap. AI-PiM extends the RISC-V processor pipeline via integration of PiM accelerators as fine-grained functional units. AI-PiM also extends the RISC-V ISA with custom instructions which directly target PiM functional units to make these accelerators transparent to the software stack and make it easier to map various computation kernels to PiM functional units. Figure 1 shows the top-level overview of AI-PiM. Section 3.2 explains the tight integration of PiM accelerators in AI-PiM methodology in detail. Sections 3.3 and Section 3.4 explain the complete end-to-end hardware, ISA and software co-design methodology of AI-PiM.

AI-PiM follows a similar developmental evolution path as the floating-point units which were originally designed for farming out a huge chunk of scientific computation to floating-point co-processors similar to large batch size AI workloads carved out for

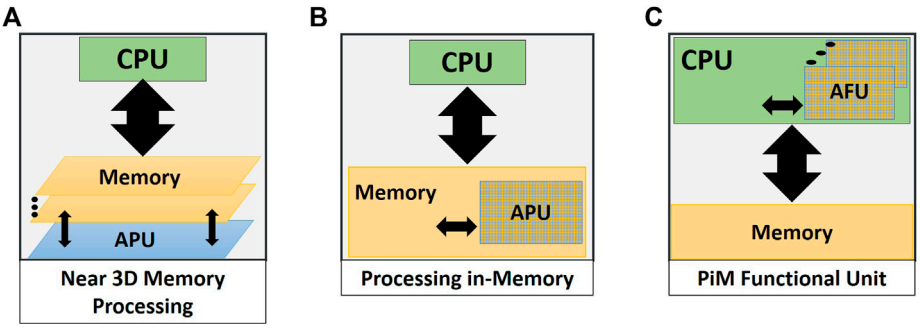


FIGURE 2
Different processing-in-memory hardware architectures for accelerating AI applications. **(A)** Near 3D memory processing solutions embed AI processing functionality into the base logic die of 3D stacked memory. **(B)** Processing-in-memory solutions convert a part of the memory hierarchy (SRAM, DRAM or NVM) into a dedicated AI Processing Unit (APU). **(C)** AI-PiM tightly integrates optimum sized PiM accelerators as AI Functional Units (AFU) inside the processor pipeline.

current PiM accelerators. But as floating-point computations became more common, floating-point accelerators became a part of the processor hardware as floating-point units (FPU) along with floating-point instructions becoming a part of the processor ISA. We believe PiM accelerators for AI acceleration will follow a similar path and AI-PiM presents a solution for this forward-looking problem. The main contributions of this article are:

- A standardized hardware, ISA and software co-design methodology to integrate PiM accelerators in the main processor pipeline.
- Treatment of PiM accelerators as a first-class citizen in the RISC-V processor pipeline unlike prior works which treat PiM hardware as a co-processor or accelerator integrated over a system bus.
- Design-space exploration of PiM functional units to determine optimum size of PiM Vector-Matrix Multiply (VMM) unit for IoT edge AI workloads.
- System level simulation results for accelerating complete neural network models for tinyML applications and comparison with existing edge AI hardware solutions.

## 2 Background

The "memory wall" problem in computer architecture (Wulf and McKee (1995); Moyer (1991)) exposes the limitations of traditional von Neumann architecture with separate hardware for compute and memory. This separation between logic processing compute blocks and the memory blocks used for storage of data and programs leads to system performance being limited by the memory bus bandwidth in memory-intensive workloads like AI. Several prior proposals have been made in the literature to bring the memory and compute units closer to reduce or remove the memory wall. The IRAM (Patterson et al. (1997)) architecture proposed fabricating a compute logic processor on the DRAM main memory chip. This was an early proposal for the PiM model but was not widely accepted due to fabrication technology limitations at the time. But after the introduction of 3D stacked memory technology like Jun et al. (2017) and Jeddeloh and Keeth (2012), many prior works like Gao et al. (2017) embedded AI Processing Units (APU) in the base logic die of a 3D stacked main memory. This technique shown in Figure 2A lowers the memory wall by introducing near-memory processing and minimizing the memory bandwidth pressure.

This was followed by research focused on embedding processing capabilities into the DRAM main memory to create Processing-in-Memory (PiM) architectures like Seshadri et al. (2017), Roy et al. (2021) and Hajinazar et al. (2021). But in the past few years, PiM architectures have also been proposed using memory technologies other than DRAM. SRAM based PiM architectures (Eckert et al. (2018); Zhang et al. (2017); Jia et al. (2022); Dong et al. (2020)) offer the advantage of easier fabrication than DRAM based solutions since SRAM is usually fabricated in the same process technology as the compute blocks. Many non-volatile memory (NVM) like RRAM based PiM solutions have also proposed recently (Shafiee et al. (2016); Chi et al. (2016); Chou et al. (2019); Li et al. (2020)). NVM based PiM solutions offer the advantage of non-volatility of data which is not possible in SRAM or DRAM based PiM architectures. Figure 2B shows the basic architecture of AI accelerators based on PiM hardware where a part of the memory hierarchy consisting of SRAM, DRAM or NVM is converted to a dedicated AI Processing Unit. Some commercial prototypes based on PiM architecture have also started to become available (Devaux (2019); Lee et al. (2021)).

Analog-to-digital and digital-to-analog converters (ADC and DAC) account for the biggest area overhead in many PiM implementations and also dominate the power consumption.

TABLE 1 Quantitative description of MLPerf Tiny benchmark (Banbury et al. (2021)) models depicting the scale of neural network models used in tinyML applications.

| Model | Number of layers | Number of parameters (K) | TFLite model size (KB) | Task | Dataset |
|---|---|---|---|---|---|
| MobileNetV1 (0.25x) | 14 | 221.8 | 325 | Visual Wake Words | Visual Wake Words Dataset |
| ResNet-V1 | 8 | 78.6 | 96 | Image Classification | CIFAR-10 |
| DSCNN | 4 | 24.9 | 52.5 | Keyword Spotting | Google Speech Commands |
| FC AutoEncoder | 9 | 270 | 270 | Anomaly Detection | ToyADMOS (ToyCar) |

There has been prior research into purely digital PiM (Imani et al. (2019)) solutions which let go of the ADC/DAC in order to simplify the PiM design and keep the area overheads low. Such solutions allow for smaller PiM sizes without having to worry about amortizing the ADC/DAC area overhead using bigger PiM arrays. Although DRAM based PiM architectures utilize large (Mb) memory sizes, SRAM and NVM based PiM solutions utilize smaller (Kb) memory arrays. Dong et al. (2020) presents an SRAM based PiM solution based on foundry provided 8T (8 transistor) bit-cell with an array size of 64 × 64 bits. Such smaller sized PiM solutions are an ideal fit low-cost edge AI applications due to limited area cost required to implement these solutions.

AI-PiM focuses on tightly integrating such small sized PiM accelerators within the RISC-V processor pipeline as functional execution units. This allows acceleration of common AI kernels like vector-matrix multiplication within the CPU pipeline. This design philosophy of utilizing PiM based AI functional units (AFU) inside the processor is shown in Figure 2C. This approach is different than prior solutions (Figures 2A,B) where relatively large PiM based AI processing units (APU) communicate with the CPU over a system bus. In AI-PiM approach shown in Figure 2C, smaller sized PiM functional units are embedded inside the processor to allow for finer-grained offloading of AI kernels. This fine-grained offloading of AI kernels is beneficial in edge AI and tinyML applications which consist of a mix of both AI and non-AI tasks which need to be seamlessly executed on the same processor hardware. Moreover, tinyML workloads (Banbury et al. (2021)) shown in Table 1 consist of small neural network models with a few kilo parameters running with batch size 1 which precludes farming out a huge chunk of the AI workload to be offloaded to bigger APUs shown in Figures 2A,B. AI-PiM utilizes PiM functional units to both store weights and compute vector-matrix multiplication operations in a weight-stationary fashion (Chen et al. (2016)). This approach to utilize smaller sized PiM AFUs has been proposed to balance the compute parallelism offered by PiM AFU with the huge area cost required for bigger PiM accelerators. Smaller sized AFUs also allow to balance the memory load/store operations with the compute operations since a bigger AFU would mandate a large number of store operations to change the weights from one layer of the neural network to the other for a single highly parallel compute operation.

Section 3.2 details how this tight integration of PiM AFU is done within the RISC-V processor pipeline. Section 3.1 discusses the support for PiM AFUs at the ISA level using custom instruction extensions to the RISC-V ISA.

# 3 Materials and methods

## 3.1 ISA extensions for PiM

AI-PiM extends the RISC-V ISA to support the PiM functional units within the RISC-V processor pipeline and expose the functionality of the PiM hardware to higher layers of abstraction in the software stack. Custom ISA instructions allow to effectively utilize the PiM functional units and make it easier to map computations to PiM hardware by conveying the information about PiM functionality to the compiler. Compiler utilizes this knowledge about exact hardware behavior of PiM to break computation kernels into appropriate sizes and map those kernels to the hardware preserving the shape and size of PiM operands.

Three new sets of instructions that have been included as PiM extensions in the RISC-V ISA with their binary encoding details shown in Figure 3. Figure 3A shows the vector-matrix multiply (**vmm**) instruction which captures the basic compute functionality of the PiM hardware. This instruction encodes the input activation in the *rs1* source operand. Two output registers *rdl* and *rdh* are used as low and high registers for the accumulated output. The *vmm* instruction supports multiple sizes for vector-matrix multiply PiM hardware units along with multiple input and accumulation bit widths. The instruction has been tested for supporting 4 bit and 8 bit input and weight (stored in memory) operands along with 8, 16 and 32 bit accumulations for the output based on different flavors of PiM hardware. If required to return more or longer output activations than what can fit in *rdl* and *rdh* registers, dedicated return registers can also be encoded along with the instruction. But the dedicated return register encoding is just a workaround for the time being till the support for either scratchpad memory or output buffer for the *vmm* instruction for specific cases is enabled. This support is still in experimental phase and would be described in detail in a future publication.

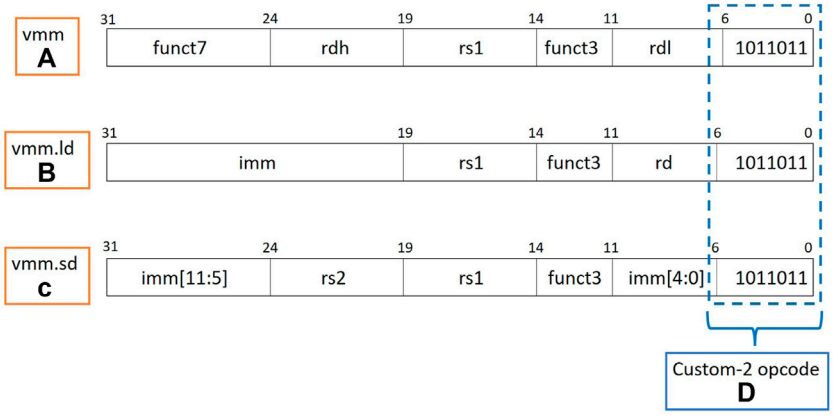


FIGURE 3
**(A)** Binary encoding for PiM vector-matrix multiply (vmm) custom instruction. **(B)** Binary encoding for the custom load instruction for PiM accelerator memory. **(C)** Binary encoding for the custom store instruction for PiM accelerator memory. **(D)** All the new instructions are encoded in the custom-2 opcode space of RISC-V ISA to maintain compatibility with the base RISC-V ISA.

Figure 3B shows the binary encoding for the custom load instruction (**vmm.ld**). This instruction is utilized for loading the output operand from the PiM AFU memory to the processor registers. For a dual cycle (execute and memory stages) PiM AFU, the load instruction is fused inside the *vmm* instruction for return of the output operand in the memory stage to be written back in the writeback stage to the *rdl* and *rdh* registers based on the size of the output operand. For multicycle PiM AFU, *vmm.ld* instruction is used to load output from the PiM AFU memory to the processor registers. Figure 3C describes the binary encoding of custom store instruction (**vmm.sd**) used to store the weights into the PiM AFU memory for weight stationary execution. *rs1* source register and *imm* immediate offset are used to calculate the address of the location in the PiM AFU memory where the value from the *rs2* source register is stored.

All the extension instructions follow the conventions of RISC-V instruction encoding philosophies like three register encoding for compute operations and immediate, source and destination register encoding restrictions for the load and store memory operations. The funct3 and funct7 bits of the instructions are used to relay information about different PiM sizes and other characteristics while keeping the rest of the instruction encoding constant to reduce the instruction decoding complexity. As shown in Figure 3D, all new PiM instruction extensions have been encoded in the custom-2 opcode space of RISC-V ISA which allows AI-PiM to be completely compatible with base RISC-V ISA.

Figure 4 shows the pseudo C code and pseudo assembly code for 1 × 8 vector multiplication to the 8 × 8 matrix with 8 bit elements each to generate 1 × 8 output vector with 16 bit accumulated values split across two 64 bit registers (*rdl* and *rdh*). This code shows that 8 × 8 matrix is written row-by-row by packing 1 × 8 8 bit rows into a single 64 bit input operand for the *vmm.sd* instruction. Once all the rows of the 8 × 8 matrix are stored in memory, the compute *vmm* instruction can be issued to perform a dual cycle vector-matrix multiply operation. The PiM receives the 1 × 8 input vector with 8 bit elements as a packed 64 bit operand (treated as a column vector) and multiplies it in parallel with matrix column values stored on each bitline to generate 1 × 8 16 bit accumulated output vector. The output vector is received in the memory stage and written to *rdl* and *rdh* destination registers as the low and the high part of the output in the writeback stage. This PiM based vector-matrix multiplication instruction is used to accelerate a variety of AI kernels like Conv2D, depthwise separable Conv2D, fully connected dense layers and other kernels which map naturally to the vmm primitive.

AI-PiM development methodology values that as we adapt the RISC-V ISA with custom instructions and augment the RISC-V hardware with PiM AFU, it is of high importance to also develop the software infrastructure to efficiently target AI applications this new AI-PiM processor architecture. These software development efforts for AI-PiM are detailed in Section 3.3.

Special care has been taken to keep the custom PiM instruction extensions agnostic to the type of underlying PiM hardware technology (RRAM/STT-MRAM/SRAM etc.) to create a clear separation between the ISA and the hardware implementation. The type of PiM and its performance characteristics are encoded separately in the processor model in a cycle accurate fashion as described in Section 3.2. This clear separation allows AI-PiM the flexibility to swap PiM accelerators while keeping the same ISA and the software stack when better PiM accelerators are available. Section 4 details the performance improvements and the area overheads of supporting these custom PiM instruction extensions to the RISC-V ISA.

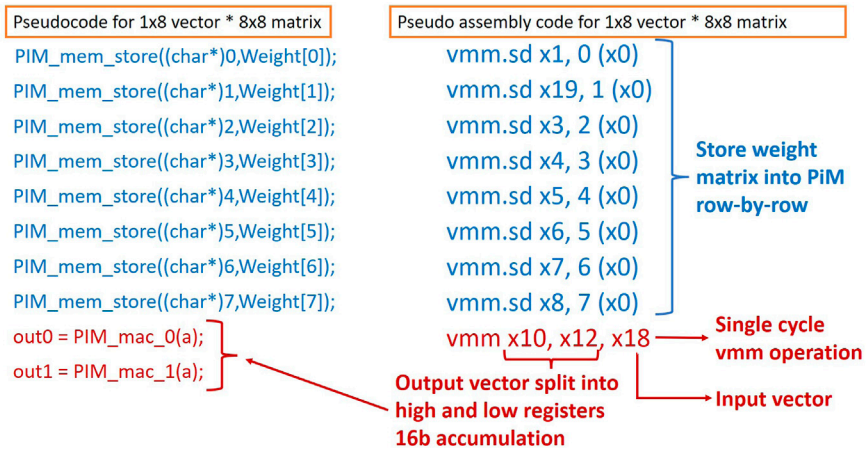


FIGURE 4
Pseudo code for 1 × 8 vector (8b elements) multiplication with an 8 × 8 matrix (8b elements) to produce 1 × 8 vector output with 16b accumulation and the corresponding pseudo assembly code with instruction extensions.

## 3.2 Tightly integrated PiM functional unit

This section provides the details about how PiM functional unit is tightly integrated inside RISC-V processor pipeline at the microarchitecture level. As discussed previously in Section 2, finer-grained offloading of AI kernels to the PiM functional units while executing IoT edge workloads consisting of a mix AI and non-AI tasks is a major motivation for this tight integration. Such tight integration of PiM functional units within an edge processor allows to save area and hardware cost of a decoupled AI accelerator by utilizing the same processor to run both AI and non-AI tasks. This approach also helps to save power required for communication of data between processor and a standalone AI accelerator used as a coprocessor. The tight integration approach is highly beneficial when PiM functional units can only accelerate a part of the AI workload e.g. vector-matrix multiplications and the host processor needs to perform pre- and post-processing of data for the PiM and execute other layers in the neural network like pooling and activation layers. Tight integration also helps to reuse resources from the processor pipeline and reduce the complexity required to design bus interface and separate ISA for a decoupled PiM accelerator. AI-PiM enables PiM functional units to accept inputs from and write the outputs back to the RISC-V processor pipeline stages and registers simplifying the interface design. Additionally, it is easier to break convolutional kernels and map them onto small PiM VMM functional units compared to bigger VMM units which require efficient mapping of workloads to keep a bigger PiM accelerator fully utilized. The smaller PiM functional units also help to balance the latency of PiM computation units with the processor pipeline delays to keep the critical execution path of the processor short. This helps to accelerate AI instructions described in Section 3.1 seamlessly on the same processor which runs non-AI part of the applications. This helps us to reduce the cost and complexity of AI hardware at the edge of IoT since AI-PiM is a complete system-level solution rather than an accelerator for AI tasks which depends on a control processor. This tight integration of AFUs enables AI-PiM to execute complete applications and not just the AI/ML kernels like prior PiM solutions.

Figure 5 shows the microarchitectural view of the modified RISC-V processor pipeline extended with PiM AI functional unit (AFU). AI-PiM implements a basic five stage pipeline for the RISC-V processor as shown in Figure 5. The custom PiM extension instructions are fetched and decoded in the same way as any other standard RISC-V instruction. But rather than using standard arithmetic and logic unit (ALU) in the execute stage, custom instructions send their operands to the PiM AFU. *vmm* instruction is a compute instruction and reads the input operand from the general purpose registers (GPR) and dispatches the same as the input to the PiM AFU. The PiM AFU computes vector-matrix multiplication of the input vector with the weight matrix stored in the PiM AFU memory and returns the output vector at the end of the memory stage which is then written back to the GPR in the writeback stage. The custom memory instructions (*vmm.ld* and *vmm.sd*) are also fetched and decoded like standard RISC-V memory instructions but rather than performing load or store operations on the RISC-V data memory, these instructions perform custom load and store operations on the PiM AFU memory in the memory stage. Such a tight integration allows AI-PiM to reuse majority of the RISC-V pipeline hardware with overheads only for the PiM functional units.

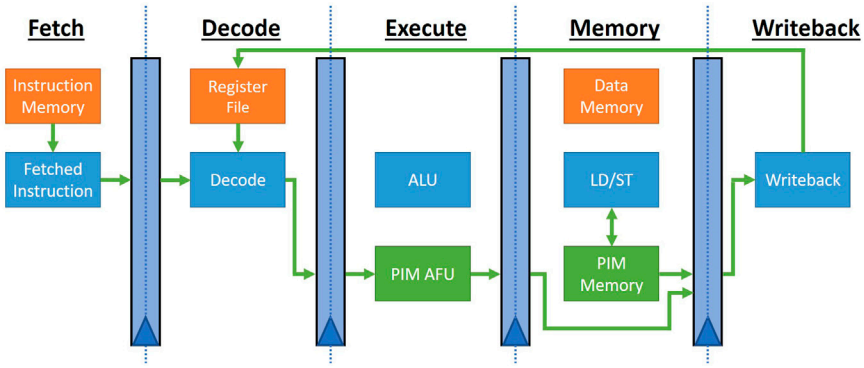


FIGURE 5
RISC-V processor pipeline data flow for custom PiM instructions targeting tightly integrated PiM functional units.

Many design decisions are important when PiM functional unit is developed to be integrated in the RISC-V processor pipeline. The first question is about the size of the PiM functional unit. It is required to balance the latency and frequency of PiM compute operations with the latency for weight updates since a larger sized AFU requires less frequent weight updates but takes longer for each update cycle. Additionally, the maturity and endurance of underlying memory technology also imposes restrictions on the physical size of the AFU. Once the AFU size is defined, the next question is how many such AFUs can be fit given an area budget and pipeline resources and complexity. AI-PiM enables researchers to find answers to all these questions as part of the design-space exploration of PiM AFUs and the impact of each hardware design on the end IoT edge AI application as described in Section 4.4. The current version of the PiM AFU has been behaviorally modeled in a cycle accurate manner as 8T SRAM (Dong et al. (2020)) based in-memory vector-matrix multiplication unit. The PiM VMM functional unit has been tested and verified at the system level to support both 4 bit and 8 bit integer formats for weights and input activations with output accumulation in 16 and 32 bit integer precision based on different quantization levels supported in the software versions of pretrained neural network models. The PiM VMM functional unit utilizes packed inputs and outputs as 64 bit operands for compatibility with 64 bit version of the RISC-V processor. Section 3.3 describes how the PiM AFU is exposed to the compiler and other layers of the software stack via custom ISA extensions described in Section 3.1.

## 3.3 Software framework development

AI-PiM extends the RISC-V processor hardware with PiM functional units as described in Section 3.2 and the RISC-V ISA with custom instruction extensions to support inclusion of PiM functional units in the hardware and software stacks as described in Section 3.1. But designing a custom processor is not very advantageous without the corresponding software support required to target real-world workloads to this processor. Hence, AI-PiM differentiates from existing PiM solutions by co-developing the complete software stack along with ISA and hardware PiM contributions. Another highlight of AI-PiM solution is that it preserves programmability in popular domain specific languages and frameworks for AI applications like PyTorch (Paszke et al. (2019)), TensorFlow (Abadi et al. (2016)), ONNX, MXNet (Chen et al. (2015)) and Keras by allowing existing neural network descriptions from any of these frameworks to be compiled with custom instructions and simulated for hardware with PiM functional units. The complete software, ISA and hardware co-design methodology and development flow has been shown in Figure 6.

AI-PiM software development framework is bifurcated into two different phases. At the frontend, modified version of the TVM (Chen et al. (2018)) open-source deep learning compiler is utilized as shown in Figure 6. TVM supports input neural network descriptions in different domain specific languages (DSL) and frameworks like TensorFlow, PyTorch, MXNet, Keras, CoreML, DarkNet and ONNX. Neural network topologies are distilled into individual operators like Conv2D, DepthwiseConv2D, pooling, transpose etc. using TVM frontends for each DSL or framework. AI-PiM extends the current TVM operators by adding operator compute definitions and operator lowering schedules which map efficiently onto PiM functional unit sizes in hardware. This support has been added to all operators which can be further lowered onto vector-matrix multiplication (VMM) kernels supported by PiM functional units in AI-PiM hardware. The optimized operator compute definitions and schedules are then lowered into Tensor Intermediate Representation (TIR), which is the low-level IR supported in TVM. AI-PiM utilizes the “tensorize” intrinsic in TVM schedule to map custom VMM kernels from individual

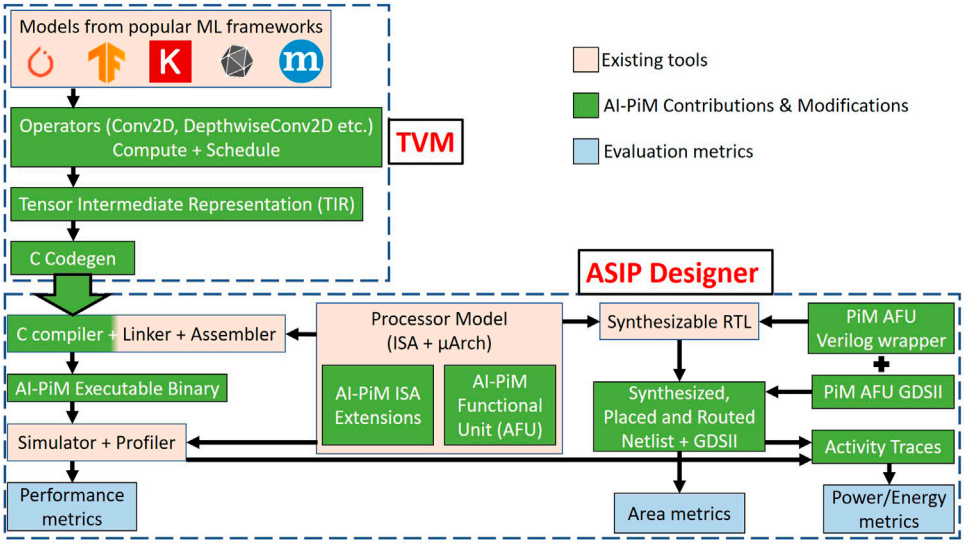


FIGURE 6
AI-PiM hardware, ISA and software co-design methodology consisting of TVM deep learning compiler (Chen et al. (2018)) as the frontend compiler to map models from popular machine learning frameworks to C and Synopsys ASIP Designer (Synopsys (2022)) as the backend processor design tool to generate complete SDK and synthesizable RTL. AI-PiM contributions and modifications to existing tools to support PiM functional units across the hardware and software stacks have been highlighted in green.

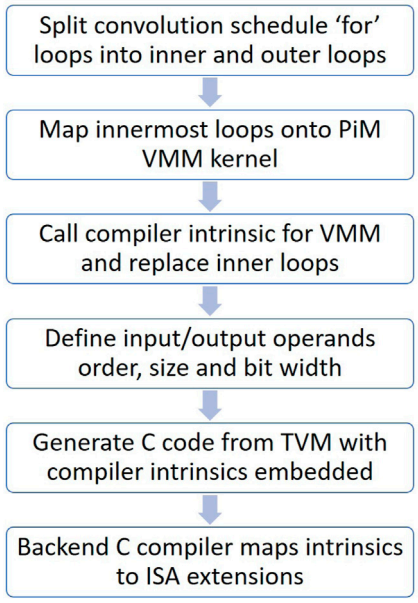


FIGURE 7
Process for mapping high-level operators like Conv2D to ISA extensions using the AI-PiM compiler intrinsics in frontend and backend compilers.

operators to the PiM functional units at the TIR level. The final TIR code consisting of regular IR and custom tensor IR for PiM units is then passed through the C code generation (codegen) phase to generate specialized C code for the input neural networks bundled with lightweight TVM C runtime. AI-PiM has extended the TVM C codegen stack to support C code generation for standard RISC-V ISA along with support for custom compiler intrinsics which map directly onto the added ISA extension assembly instructions for AI-PiM as shown in Figure 7. This extension of TVM C codegen allows AI-PiM to generate C code for input neural networks from different DSLs and frameworks with custom compiler intrinsics for ISA extensions embedded into the generated C source code. This is an optimized way of generating C code consisting of standard C functions and customized intrinsics for ISA extensions since this approach does not require to run any specialized scripts to convert TVM generated generic C code to C code with support for PiM instructions or manually hand code system libraries or assembly instructions to support PiM ISA extensions in AI-PiM. TVM compiler modifications at different levels enable AI-PiM to generate customized C code with support for PiM functional units for neural network models from almost all the standard AI frameworks. This allows AI-PiM to support processing-in-memory based hardware functional units in the processor and expose this functionality to the highest levels of the software abstraction all the way up to AI specific DSLs and frameworks.

Figure 7 provides further details into step-by-step process of lowering specialized schedule of TVM operators down to AI-PiM ISA extension assembly instructions. TVM compute operators like Conv2D, DepthwiseConv2D, Dense etc. are augmented with special lowering schedules for AI-PiM where the main kernel

consisting of multiple "for" loops is split into inner and outer loops. The innermost two loops are sized to map efficiently onto the PiM hardware functional units. After this splitting at the operator level, inner loops are replaced with compiler intrinsics at the TIR level. Special care is given to properly define the input and output operands in correct order, size and bit width to ensure correct mapping of operands from operator to TIR to TIR with compiler intrinsics. This code is then lowered further where regular TIR is lowered to standard C code and compiler intrinsics are retained within the C code. This version of the C code is then generated from TVM and passed onto the backend C compiler. The backend compiler is modified to map compiler intrinsics to custom instructions and this allows to generate assembly code consisting of standard RISC-V instructions along with custom assembly instructions.

Synopsys ASIP Designer (Synopsys (2022)) acts as the backend of the AI-PiM hardware, ISA and software co-design methodology as shown in Figure 6. The backend definition starts from the processor model shown in Figure 6. The processor model consists of the instructions supported in the ISA along with the microarchitecture definition of the processor describing register transfers in each stage of the processor pipeline for each ISA instruction. This processor description is written in a tool specific language called nML (Fauth et al. (1995)). AI-PiM extends the current RISC-V processor model with AI-PiM functional units integrated inside the processor pipeline (Section 3.2) and ISA extensions targeting these functional units (Section 3.1). This extended processor model is used by ASIP Designer to generate a C compiler for the extended RISC-V processor. AI-PiM further modifies the generated compiler to define the compiler intrinsics for ISA extension instructions which are then exposed to TVM frontend. AI-PiM creates a bridge between enhanced versions of TVM and ASIP Designer where C code generated from TVM along with custom compiler intrinsics can be directly compiled by ASIP Designer generated C compiler which lowers the intrinsics to appropriate assembly instructions which map directly onto the PiM functional units. ASIP Designer utilizes the extended RISC-V processor model to also generate complete Software Development Kit (SDK) including assembler, linker, debugger, instruction set simulator and profiler. Executable binary is then generated for AI-PiM processor consisting of standard RISC-V instructions and custom ISA extensions to support PiM functional units. This executable binary is simulated using a cycle-accurate instruction set simulator to generate performance metrics and dynamic activity traces for power estimation.

## 3.4 Top-level hardware design

AI-PiM design methodology is based on hardware, ISA and software co-design approach where PiM hardware is co-developed along with corresponding ISA extensions and software framework. The design methodology to integrate PiM hardware functional units into top-level hardware design is shown in Figure 6. The processor model describing the ISA and microarchitecture of AI-PiM processor is used to generate a synthesizable RTL along with the SDK generation. This ensures that the software and hardware are always in sync with each other since they are generated from the same processor description. PiM functional units follow a custom hardware development due to the analog and mixed-signal nature of PiM blocks. The top level Verilog wrapper for the custom PiM blocks is generated to capture the input and output signals and to connect these signals to the top-level processor hardware. This Verilog wrapper replaces the synthesizable RTL generated by the processor design tool for PiM functional units based of their behavioral description in the processor model. This synthesizable RTL along with the PiM AFU Verilog wrapper is then synthesized and placed and routed (P&R) to generate the final netlist and GDSII layout for the AI-PiM processor. PiM functional unit modules are marked as black boxes during the synthesis and P&R steps and custom designed layout for the PiM blocks is placed in the black box locations and routed to generate accurate area metrics for the AI-PiM design.

# 4 Results

## 4.1 Evaluation methodology

Detailed experiments have been performed to evaluate the performance of AI-PiM processor on AI kernels and benchmarks. All the performance evaluation results for AI-PiM and RISC-V baseline are generated using a cycle-accurate simulator and the area results have been obtained through synthesis with Synopsys DC Compiler on 14 nm FinFET technology from GlobalFoundries. All the experiments have been performed on quantized neural networks and utilize 8-bit weights and input activations and 32-bit accumulations for the VMM operation. MLPerf Tiny benchmark (Banbury et al. (2021)) along with ResNet-50 neural network model and general matrix-vector multiplication kernel have been used to quantify the performance at a small AI kernel and complete neural network levels. MLPerf Tiny benchmark is representative of real-world workloads for the AI applications at the extreme edge of IoT. A standard 5-stage in-order RISC-V processor implementation supporting RV64IMC RISC-V ISA is chosen as the baseline. AI-PiM performance has also been compared to Arm Cortex-A72 processor from the Raspberry Pi 4. RISC-V processor with support for the standard integer (I), multiplication (M) and compressed (C) instruction extensions (Waterman et al. (2014)) and Raspberry Pi 4 are common choices for tinyML (Warden and Situnayake (2019)) systems as depicted by the hardware choices for the systems in the results section of the MLPerf Tiny benchmark

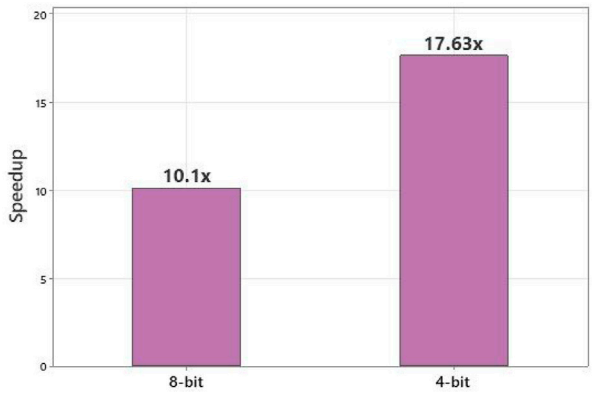


FIGURE 8
Performance improvement of AI-PiM for GEMV kernel compared to RV64IMC baseline. Results for 64 × 64 bit PiM functional unit used in two different configurations are shown - with 8-bit inputs (1 × 8 vector * 8 × 8 matrix) and 4-bit inputs (1 × 16 vector * 16 × 16 matrix).

(MLCommons (2021)). Arm Cortex-A72 performance metrics are generated by running compiled neural networks on Raspberry Pi 400 using the perf performance analysis tool. If the size is not explicitly mentioned then the AI-PiM results correspond to 64 × 64 bit 8T SRAM based in-memory vector-matrix multiply unit. Section 4.4 discusses the effects of different sizes for the PiM VMM AFU.

## 4.2 GEMV kernel performance

Figure 8 shows the performance improvement of using AI-PiM with 64 × 64 bit in-SRAM functional unit in two different modes compared to RV64IMC RISC-V baseline processor. Mode 1 utilizes 8-bit input operands and accumulates in 16-bit to perform 1 × 8 vector multiplication with 8 × 8 matrix. Mode 2 works on 4-bit input operands and accumulates the output in 8-bit to perform 1 × 16 vector multiplication with 16 × 16 matrix. Figure 8 shows the speed improvements of AI-PiM over a small VMM kernel. VMM forms the basis of most neural network inference computations and this result shows the effectiveness of AI-PiM in accelerating edge AI workloads.

## 4.3 MLPerf tiny benchmark performance

AI-PiM is a system-level solution capable of accelerating complete neural network models rather than an accelerator-only solution focusing on accelerating select AI kernels. This capability enables simulation of the complete MLPerf Tiny inference benchmark. The results from this experiment are shown in Figure 9. The number of processor cycles required to compute a single inference on each neural network model from the benchmark have been computed and compared against the RISC-V baseline processor and also against Arm Cortex-A72 processor from Raspberry Pi 4.

Figure 10 further details the performance speedup offered by AI-PiM over current edge AI processors. AI-PiM provides and average speedup of 2.74x compared to RISC-V baseline processor and 2.45x compared to Arm Cortex-A72 processor for the MLPerf Tiny benchmark. It should be noted that the 10.1x speedup shown in Figure 8 with 8-bit input operands for a small GEMV kernel does not translate to similar improvements at the complete neural network level as shown for different neural network models in Figure 10. The reason is that PiM functional units accelerate only convolutional and fully connected layers of the neural networks and other layers are not accelerated by PiM functional units. Additionally, TVM introduces a lot of memory-management wrapper code around the neural network layers which is not accelerated by the PiM functional units and the custom ISA extensions. Hence, we see a reduction in speedup when moving from a hand crafted GEMV kernel written in C to complete neural network model written in TensorFlow Lite for Microcontrollers (TFLM) and converted to C code via frontend ML compiler like TVM. This shows the importance of complete workload level performance analysis enabled by AI-PiM compared to other PiM solutions which focus only on kernel level performance analysis.

The neural network models in the MLPerf Tiny benchmark cover a wide variety of models including standard convolutional neural networks (CNN) like ResNet, models with depthwise separable convolutions and deep autoencoder models. The benchmark also spans across different applications used for edge AI and tinyML application domains like audio and video keyword spotting, image classification and anomaly detection. AI-PiM achieves a consistent speedup over the current edge AI processors for all models in the benchmark showing the effectiveness of designing AI-PiM as a general AI acceleration solution rather than specializing AI-PiM for any particular type of neural network model or application.

## 4.4 Design-space exploration

The results shown thus far have utilized 64 × 64 bit in-SRAM VMM functional unit. But AI-PiM hardware, ISA and software co-design allows extensive design-space exploration for PiM functional units. One of the knobs in this design-space exploration is the size of the PiM functional units. AI-PiM allows to sweep the possible sizes of PiM AFUs and generate performance and area metrics based on cycle accurate performance measurements on real workloads and synthesis or place and route based area estimations for each size of the PiM AFU in this design-space. Rather than relying on empirical decisions about what should be the size of the PiM AFU, AI-

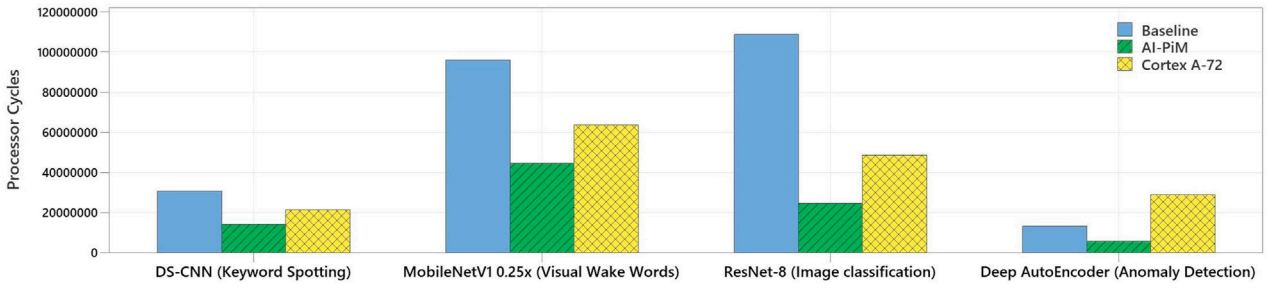


FIGURE 9
Comparison of AI-PiM performance with RV64IMC RISC-V baseline and Arm Cortex-A72 processor on MLPerf Tiny benchmark in terms of number of processor cycles.

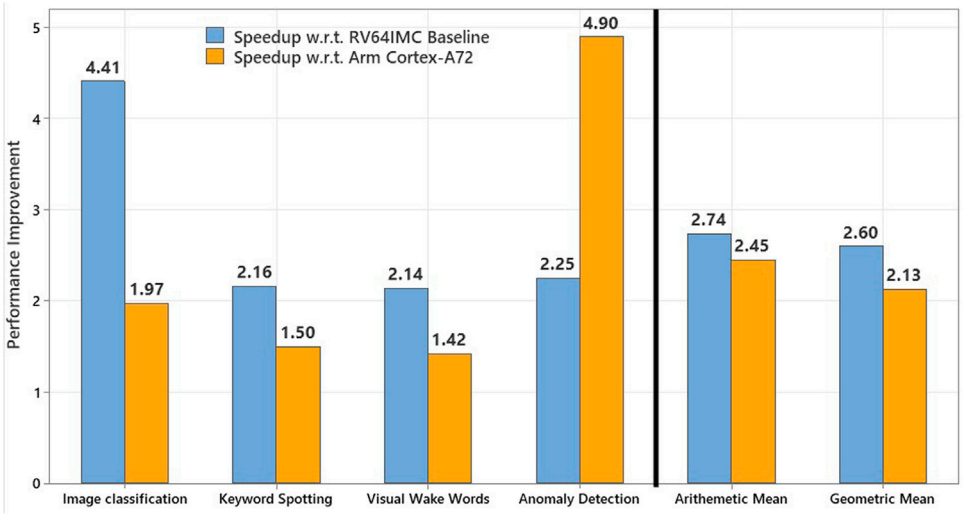


FIGURE 10
Performance speedup as a result of AI-PiM compared to RV64IMC RISC-V baseline and Arm Cortex-A72 processor on MLPerf Tiny benchmark.

PiM allows comprehensive data based analysis of the impact of each possible AFU size on the performance of the end AI applications and area of end hardware implementation. In this section, we show an example of the design space exploration by considering the performance and area impacts of PiM VMM sizes ranging from small 16 × 16 bit functional units to 64 × 64 bit functional units. Figure 11 shows the performance improvement offered by AI-PiM design with different PiM AFU sizes for the ResNet-8 network from the MLPerf Tiny benchmark suite. Results shown in Figure 11 match the intuitive understanding that the bigger the size of the accelerator, higher is the parallelism offered by the VMM unit and hence, higher the actual speedup of the design.

But a quick look at Figure 12 clearly shows that higher accelerator performance comes at a higher area overhead. The area overhead is measured as a ratio of the extra area required for AI-PiM with different PiM functional unit sizes to the area of the baseline RV64IMC processor. In such a scenario, AI-PiM offers an effective design-space exploration methodology where users can do a comprehensive trade-off analysis at the design time between the performance and area overhead of each size of the PiM functional unit by comparing the performance not on just small kernels but on complete neural network models which will serve as the final application workloads.

Edge AI devices are often cost, size and power limited which requires extracting maximum performance at a minimum area cost. This philosophy is used to devise a simple figure of merit (FoM) by dividing the percentage performance improvement offered by individual PiM VMM AFU sizes by percentage area overhead for each design. This allows to maximize the

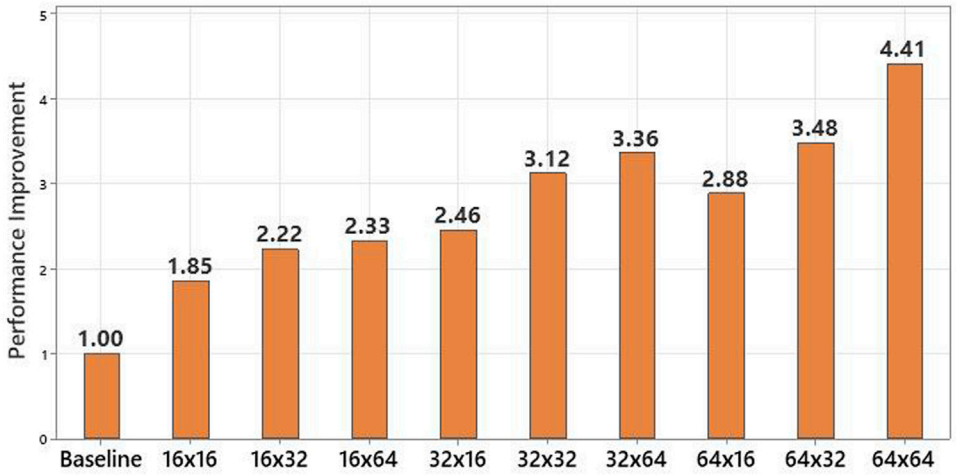


FIGURE 11
Performance speedup on ResNet-8 model from MLPerf Tiny benchmark in terms of number of processor cycles compared to RV64IMC 5-stage RISC-V processor baseline.

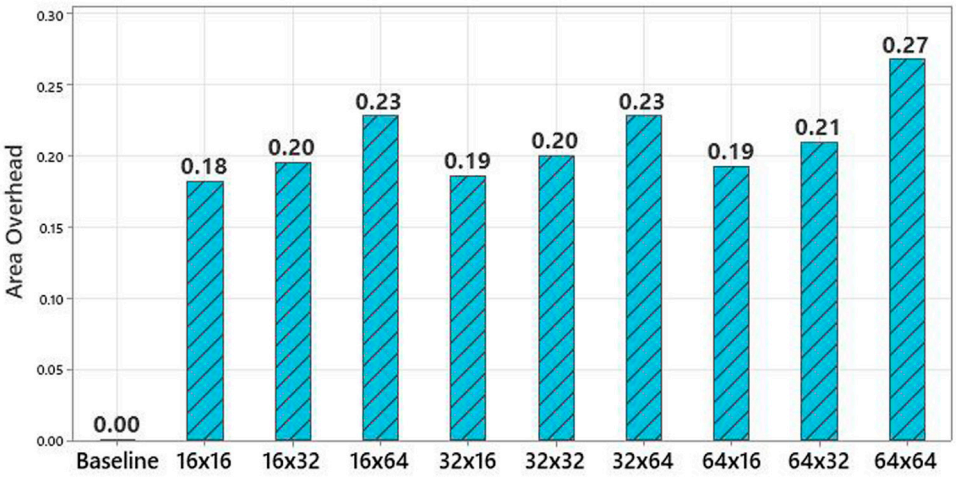


FIGURE 12
Area overhead of different PiM AFU sizes compared to RV64IMC 5-stage RISC-V processor baseline.

performance per unit area cost. Figure 13 clearly shows that designing using smallest PiM AFU in terms saving area or designing with biggest PiM AFU in terms of performance maximization does not offer the highest advantage. Rather AFU sizes in the middle of the range such as 32 × 32, 64 × 16 and 64 × 32 provide the highest performance improvement per percentage area overhead. Such figures of merits are necessary when designing tinyML and edge hardware devices where a very high importance is given to keeping the area cost of the design to the minimum.

## 4.5 Comparison to standalone AI accelerator

AI-PiM also enables power and energy estimations using simulated activity files back-annotated to either synthesis or post-place-and-route netlists. This allows for accurate, fast and extensive power, performance, area and energy (PPAE) analysis for each hardware, ISA and software framework design point. The power results in this section have been generated using the activity files from simulation of the audio keyword spotting

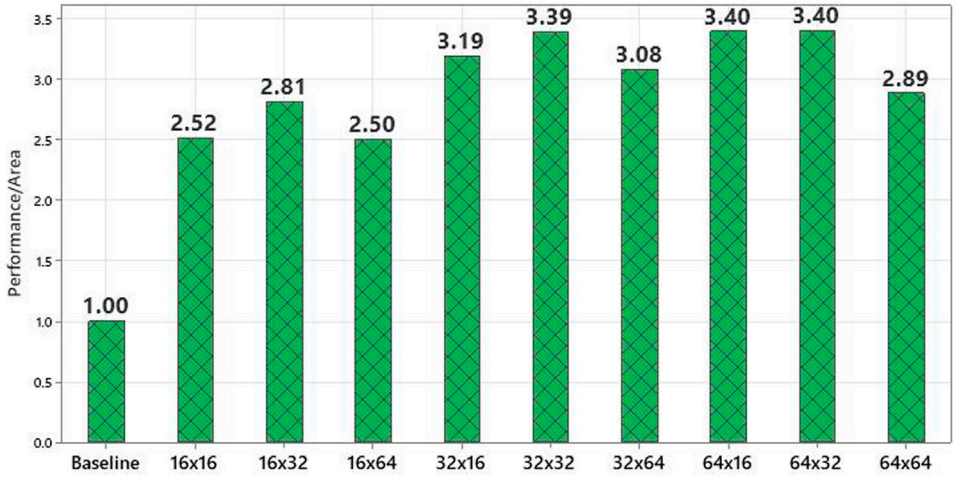


FIGURE 13
Improvement in Figure of Merit for AI-PiM (percentage performance gain/percentage area overhead) compared to RV64IMC 5-stage RISC-V processor baseline.

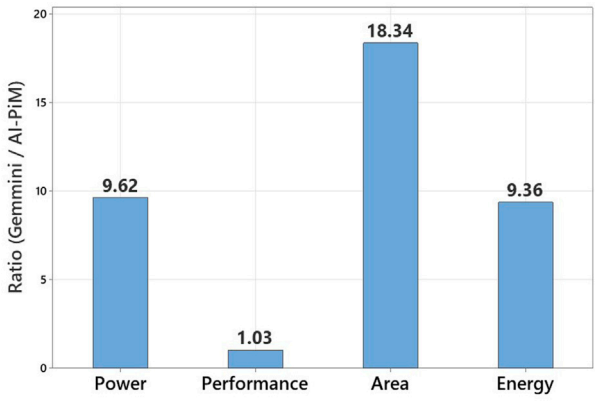


FIGURE 14
Complete power, performance, area and energy (PPAE) comparison of AI-PiM with the state-of-the-art Gemmini standalone AI accelerator.

benchmark featuring DS-CNN neural network model from MLPerf Tiny benchmark. These activity files are then back-annotated to the RTL for AI-PiM RISC-V processor and synthesized for 14 nm FinFET technology. Power estimates for the PiM AFU are added to the processor average power metrics. All the power metrics show the average power consumption for the DS-CNN benchmark. Energy spent on the entire benchmark model inference is calculated using the average power, performance in terms of number of cycles required to run the benchmark for each design and the frequency of the design.

Figure 14 shows PPAE comparison of AI-PiM with the equivalent Gemmini accelerator (Genc et al. (2021); Gonzalez and Hong (2020)) with 8 × 8 systolic array. Figure 14 shows that loosely coupled Gemmini systolic array accelerator takes 9.62 times the power, 18.34 times the area and 9.36 higher energy to offer just 3% performance improvement over AI-PiM for the ResNet-50 neural network model. It should be noted that Gemmini uses hand crafted C kernels to report the ResNet-50 performance while AI-PiM uses TensorFlow Lite compiled version of the benchmark using TVM. The hand crafted C kernels offer high performance efficiency but AI-PiM is able to achieve equivalent performance with the model compiled from a high level domain-specific language. This further proves the advantage of hardware, ISA and software co-design methodology of AI-PiM and shows the advantage of tight integration of AFU compared to loose integration of AI accelerator as performed in the Gemmini accelerator.

## 5 Discussion

AI algorithms are evolving at a much faster pace than AI hardware development. This necessitates building scalable and flexible hardware solutions for processing AI applications all the way from the cloud to the extreme edge of IoT. AI-PiM offers this flexibility by extending the RISC-V processor with custom instructions and hardware for integrating PiM accelerators within the processor. Although the current article is focused on SRAM based PiM functional unit, AI-PiM offers the flexibility to integrate other CMOS compatible PiM technologies like RRAM based PiM units also within the processor pipeline. Most importantly, AI-PiM develops a standardized design methodology for integrating PiM accelerators in the standard computing stack at the microarchitecture, ISA and software

levels. This allows to build an extended RISC-V based edge processor which is capable of exploiting the parallelism advantages of PiM accelerators and couple it with the programmable nature of a standard processor to build a hardware solution which can process both AI and non-AI edge applications. Accurate power estimation is an important aspect of edge processor design along with performance and area estimation. AI-PiM methodology provides an agile path to perform such power, performance and area (PPA) estimation. Power metrics for AI-PiM have not been reported in this article but will be the subject of future work. Authors are also extending the current AI-PiM design to support multiple and heterogeneous PiM functional units to offer an easy path for integrating these non von Neumann accelerators in the standard computing model.

## 6 Conclusion

In this article, we presented AI-PiM architecture for tightly integrating PiM accelerators as functional execution units inside the RISC-V processor using hardware, ISA and software co-design methodology. AI-PiM extends the RISC-V processor hardware with PiM functional units, RISC-V ISA with custom instruction extensions and develops the software framework to transparently expose PiM accelerators to the software stack. Agile design methodology of AI-PiM allows users to explore a comprehensive design-space at the hardware and ISA levels. AI-PiM enables processing of both AI and non-AI applications on single edge processor and provides an average speedup of 2.74x over the RV64IMC RISC-V processor baseline and 2.45x speedup over Arm Cortex-A72 processor for MLPerf Tiny benchmark.

## Data availability statement

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

## Author contributions

VV was involved with the conceptualization of the idea, conducting experiments and writing the manuscript. MS was involved with the conceptualization of the idea and writing and reviewing the manuscript. This manuscript has been approved by all the authors.

## Funding

This work has been funded by Semiconductor Research Corporation (SRC) under GRC AIHW program task number 2945.001. This work was supported in part by CRISP, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program, sponsored by MARCO and DARPA.

## Acknowledgments

The authors would like to thank SRC for funding this research and to Xida Ren from University of Virginia for providing access to Raspberry Pi during the pandemic. The authors also acknowledge lively discussions that have shaped the work with: Kevin Skadron (UVa), Ashish Venkat (UVa), Mike Caraman (NXP), Mahesh Chandra (NXP), Ramesh Chauhan (Qualcomm), Vivek De (Intel), Paul Somnath (Intel), Deepak Desalukunte (Intel), Muhammad Khelah (INtel), Krishnan Kailas (IBM), Philip Jacob (IBM), Karthik Swaminathan (IBM), Xin Zhang (IBM), Matt Ziegler (IBM), Mihir Mody (TI).

## Conflict of Interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest. The funder was not involved in the study design, collection, analysis, interpretation of data, the writing of this article or the decision to submit it for publication.

## Publisher's note



## References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., et al. (2016). "TensorFlow: A system for large-scale machine learning," in 12th USENIX Symposium on Operating Systems Design and Implementation, 265–283.

Anderson, M., Chen, B., Chen, S., Deng, S., Fix, J., Gschwind, M., et al. (2021). *First-generation inference accelerator deployment at facebook*.

Banbury, C. R., Reddi, V. J., Lam, M., Fu, W., Fazel, A., Holleman, J., et al. (2021). *Benchmarking TinyML systems: Challenges and direction*.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., et al. (2015). Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., et al. (2018). "Tvm: An automated end-to-end optimizing compiler for deep learning," in 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). (Carlsbad, CA: USENIX Association), 578–594.

Chen, Y.-H., Krishna, T., Emer, J., and Sze, V. (2016). "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in IEEE International Solid-State Circuits Conference, ISSCC 2016, 262–263. Digest of Technical Papers.

Chi, P., Li, S., Xu, C., Zhang, T., Zhao, J., Liu, Y., et al. (2016). "Prime: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 27–39. doi:10.1109/ISCA.2016.13

Chou, T., Tang, W., Botimer, J., and Zhang, Z. (2019). "Cascade: Connecting RRAMs to extend analog dataflow in an end-to-end in-memory processing paradigm," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, New York, NY, USA (Association for Computing Machinery), 114–125. doi:10.1145/3352460.3358328

Devaux, F. (2019). "The true Processing in Memory accelerator," in 2019 IEEE Hot Chips 31 Symposium (HCS), 1–24. doi:10.1109/HOTCHIPS.2019.8875680

Dong, Q., Sinangil, M. E., Erbagci, B., Sun, D., Khwa, W.-S., Liao, H.-J., et al. (2020). "15.3 A 351 tops/W and 372.4 gops compute-in-memory SRAM macro in 7nm FinFET CMOS for machine-learning applications," in 2020 IEEE International Solid- State Circuits Conference - (ISSCC), 242–244. doi:10.1109/ISSCC19947.2020.9062985

Eckert, C., Wang, X., Wang, J., Subramaniyan, A., Iyer, R., Sylvester, D., et al. (2018). "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), 383–396. doi:10.1109/ISCA.2018.00040

Fauth, A., Van Praet, J., and Freericks, M. (1995). "Describing instruction set processors using nML," in Proceedings the European Design and Test Conference. ED TC 1995, 503–507.

Gao, M., Pu, J., Yang, X., Horowitz, M., and Kozyrakis, C. (2017). "Tetris: Scalable and efficient neural network acceleration with 3D memory," in Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (New York, NY, USA: Association for Computing Machinery), 751–764. doi:10.1145/3037697.3037702

Genc, H., Kim, S., Amid, A., Haj-Ali, A., Iyer, V., Prakash, P., et al. (2021). "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in 2021 58th ACM/IEEE Design Automation Conference (DAC), 769–774. doi:10.1109/DAC18074.2021.9586216

Gonzalez, A., and Hong, C. (2020). *A chipyard comparison of NVDLA and Gemmini*.

Hajinazar, N., Oliveira, G. F., Gregorio, S., Ferreira, J. a. D., Ghiasi, N. M., Patel, M., et al. (2021). "Simdram: A framework for bit-serial simd processing using DRAM," in Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, New York, NY, USA (Association for Computing Machinery), 329–345.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778. doi:10.1109/CVPR.2016.90

Imani, M., Gupta, S., Kim, Y., and Rosing, T. (2019). "FloatPIM: In-Memory acceleration of deep neural network training with high precision," in 2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA), 802–815.

Jeddeloh, J., and Keeth, B. (2012). "Hybrid memory cube new DRAM architecture increases density and performance," in 2012 Symposium on VLSI Technology (VLSIT), 87–88. doi:10.1109/VLSIT.2012.6242474

Jia, H., Ozatay, M., Tang, Y., Valavi, H., Pathak, R., Lee, J., et al. (2022). Scalable and programmable neural network inference accelerator based on in-memory computing. *IEEE J. Solid-State Circuits* 57, 198–211. doi:10.1109/JSSC.2021.3119018

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., et al. (2017). "In-datacenter performance analysis of a tensor processing unit," in Proceedings of the 44th Annual International Symposium on Computer Architecture, New York, NY, USA (Association for Computing Machinery, ISCA), 1–12. doi:10.1145/3079856.3080246

Jun, H., Cho, J., Lee, K., Son, H.-Y., Kim, K., Jin, H., et al. (2017). "HBM (high bandwidth memory) DRAM technology and architecture," in 2017 IEEE International Memory Workshop (IMW), 1–4. doi:10.1109/IMW.2017.7939084

Lee, S., Kang, S.-h., Lee, J., Kim, H., Lee, E., Seo, S., et al. (2021). "Hardware architecture and software stack for PIM based on commercial DRAM technology : Industrial product," in 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), 43–56. doi:10.1109/ISCA52012.2021.00013

Li, W., Xu, P., Zhao, Y., Li, H., Xie, Y., and Lin, Y. (2020). "Timely: Pushing data movements and interfaces in PIM accelerators towards local and in time domain," in Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (IEEE Press), 832–845.

MLCommons (2021). *MLPerf Tiny results*. Availableat: https://mlcommons.org/en/inference-tiny-05/.

Moyer, S. A. (1991). Performance of the iPSC/860 node architecture *(Citeseer)*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). *PyTorch: An imperative style*. High-Performance Deep Learning Library.

Patterson, D., Anderson, T., Cardwell, N., Fromm, R., Keeton, K., Kozyrakis, C., et al. (1997). A case for intelligent RAM. *IEEE Micro* 17, 34–44. doi:10.1109/40.592312

Roy, S., Ali, M., and Raghunathan, A. (2021). PIM-DRAM: Accelerating machine learning workloads using processing in commodity DRAM. *IEEE J. Emerg. Sel. Top. Circuits Syst.* 11, 701–710. doi:10.1109/JETCAS.2021.3127517

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., et al. (2015). ImageNet large scale visual recognition Challenge. *Int. J. Comput. Vis.* 115, 211–252. doi:10.1007/s11263-015-0816-y

Seshadri, V., Lee, D., Mullins, T., Hassan, H., Boroumand, A., Kim, J., et al. (2017). "Ambit: In-Memory accelerator for bulk bitwise operations using commodity DRAM technology," in 2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 273–287.

Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J. P., Hu, M., et al. (2016). "Isaac: A convolutional neural network accelerator with *in-situ* analog arithmetic in crossbars," in Proceedings of the 43rd International Symposium on Computer Architecture (IEEE Press), 14–26. doi:10.1109/ISCA.2016.12

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature* 529, 484–489. doi:10.1038/nature16961

Synopsys (2022). ASIP designer. Availableat: https://www.synopsys.com/dw/ipdir.php?ds=asip-designer.

Warden, P., and Situnayake, D. (2019). *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media, Inc..

Waterman, A., Lee, Y., Patterson, D. A., and Asanović, K. (2014). *The RISC-V instruction set manual, volume I: User-level ISA, version 2.0*. Tech. Rep. UCB/EECS-2014-54. Berkeley: EECS Department, University of California.

Wulf, W. A., and McKee, S. A. (1995). Hitting the memory wall: Implications of the obvious. *SIGARCH Comput. Archit. News* 23, 20–24. doi:10.1145/216585.216588

Yazdanbakhsh, A., Seshadri, K. K., Akin, B., Laudon, J., and Narayanaswami, R. (2021). An evaluation of edge TPU accelerators for convolutional neural networks. ArXiv abs/2102.10423

Zhang, J., Wang, Z., and Verma, N. (2017). In-memory computation of a machine-learning classifier in a standard 6T SRAM array. *IEEE J. Solid-State Circuits* 52, 915–924. doi:10.1109/JSSC.2016.2642198