In-Pipeline Integration of Digital In-Memory-Computing into RISC-V Vector Architecture to Accelerate Deep Learning

Tommaso Spagnolo* Cristina Silvano*
Riccardo Massa† Filippo Grillotti† Thomas Boesch‡ Giuseppe Desoli†
*Politecnico di Milano, Milan, Italy
Email: tommaso.spagnolo@polimi.it, cristina.silvano@polimi.it
†STMicroelectronics, Milan, Italy
Email: riccardo.massa@st.com, filippo.grillotti@st.com, giuseppe.desoli@st.com
‡STMicroelectronics, Geneva, Switzerland
Email: thomas.boesch@st.com

***Abstract*—Expanding Deep Learning applications toward edge computing demands architectures capable of delivering high computational performance and efficiency while adhering to tight power and memory constraints. Digital In-Memory Computing (DIMC) addresses this need by moving part of the computation directly within memory arrays, significantly reducing data movement and improving energy efficiency. This paper introduces a novel architecture that extends the Vector RISC-V Instruction Set Architecture (ISA) to integrate a tightly coupled DIMC unit directly into the execution stage of the pipeline, to accelerate Deep Learning inference at the edge. Specifically, the proposed approach adds four custom instructions dedicated to data loading, computation, and write-back, enabling flexible and optimal control of the inference execution on the target architecture. Experimental results demonstrate high utilization of the DIMC tile in Vector RISC-V and sustained throughput across the ResNet-50 model, achieving a peak performance of 137 GOP/s. The proposed architecture achieves a speedup of 217× over the baseline core and 50× area-normalized speedup even when operating near the hardware resource limits. The experimental results confirm the high potential of the proposed architecture as a scalable and efficient solution to accelerate Deep Learning inference on the edge.**

***Index Terms*—AI Accelerator, RISC-V Architecture, Instruction Set Extension, Digital In-Memory Computing, Vector Processors**

I. Introduction

Artificial Intelligence continues to reshape technology and society, driven by the rapid evolution of increasingly complex AI models. Since the advent of modern Deep Learning with AlexNet in 2012 [1], AI model complexity, typically measured by parameter count, has roughly doubled each year [2]. Conversely, hardware performance has improved at a substantially slower pace, achieving only around 30% annual growth in terms of FP32 TFLOP/s [3]. This widening gap has made it clear that traditional computing architectures can no longer rely solely on incremental improvements; instead, architectural innovation is required to effectively manage the computational demands of emerging AI workloads. Furthermore, contemporary AI models have become increasingly diverse, encompassing various neural network types, evolving inter-layer connectivity, distinct data-flow patterns, diverse activation functions, and multiple numerical formats (e.g., quantization levels). These variations place significant emphasis on computational flexibility, as fixed-function hardware accelerators, though efficient, often lack the adaptability required to accommodate such rapidly changing and heterogeneous workloads.

The widespread adoption of AI has highlighted significant drawbacks in centralized cloud computing, especially regarding latency, energy efficiency, and data security [4]. These challenges have driven a shift toward edge computing, where AI inference occurs locally on devices, providing lower latency, improved security, and reduced dependency on network connectivity. However, edge deployment imposes strict constraints on power, memory, and performance, necessitating specialized hardware capable of high computational throughput within tight energy and memory limits.

At the device level, the primary obstacle to efficiently executing advanced AI workloads arises from excessive data movement rather than computation itself. Modern Deep Learning architectures, such as Convolutional Neural Networks (CNNs) and transformers, frequently move large volumes of feature maps, weights, and intermediate results across memory hierarchies, placing immense pressure on memory bandwidth. Traditional digital accelerators often struggle with significant latency and energy overheads, largely driven by continuous data transfer between memory and compute units. Indeed, transferring data from off-chip DRAM typically consumes orders of magnitude more energy than performing actual Multiply-Accumulate (MAC) operations [5], primarily due to interconnect capacitance.

Addressing this critical bottleneck requires improving data locality and reducing the physical distance between memory and computation (see Fig. 1). Near-memory computing has emerged as a promising strategy, which involves placing small, high-speed memories, such as scratchpads, close to compute units. By storing frequently accessed data near the compute elements, these architectures significantly minimize energy consumption and latency, promoting data reuse.

Building on the concept of near-memory computing, In-Memory Computing (IMC) takes this paradigm even further by integrating computational capabilities directly in memory.

IMC architectures are classified into Analog (AIMC) and Digital (DIMC). AIMC uses emerging Non-Volatile Memory (NVM) technologies like ReRAM or PCM to perform analog MACs [6], achieving energy efficiency over 1000 TOPS/W at low precision [7], but suffers from reliability and accuracy issues [8], limiting adoption in precision-critical tasks. DIMC, on the other hand, uses digital SRAM arrays and offers deterministic behavior, high bit-precision, and seamless CMOS integration. Though typically achieving lower energy efficiency (10–300 TOPS/W) than AIMC, DIMC ensures

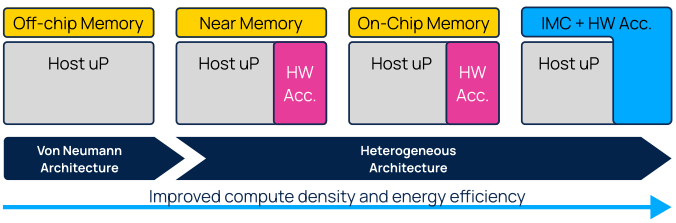

Fig. 1. Architectural approaches to improve data locality: from von Neumann designs to heterogeneous systems with near-memory and in-memory computing.

robustness and accuracy, making it ideal for fully-digital CNN accelerators at the edge [9], [10].

Integrating IMC units into processors involves choosing between tightly-coupled and loosely-coupled configurations. In loosely-coupled designs, IMC tiles act as accelerators accessed via load/store instructions through the processor's I/O bus at specific memory-mapped addresses. Although scalable and simple to integrate, this approach introduces significant communication overhead, causing stalls and performance loss. Conversely, tightly-coupled designs embed IMC units directly into the pipeline or nearby memory, leveraging ISA extensions for direct access. This reduces communication latency to single-digit cycles by avoiding memory hierarchy and interconnect delays, though it adds complexity through extra pipeline ports, hazard logic, tighter timing, and extensive verification. Despite this complexity, tightly-coupled integration provides superior responsiveness, efficiency, and flexibility for adapting to various neural network topologies and dynamic data flows.

The RISC-V ISA, with its open and modular structure, provides an ideal foundation for exploring these integration strategies. Its flexibility and modularity allow designers to introduce custom instructions that support IMC operation, whether tightly integrated or offloaded, while maintaining compatibility with existing toolchains.

Building on this modularity, the RISC-V vector extension [11] introduces native support for data-parallel workloads such as CNN inference, Digital Signal Processor (DSP), and computer vision algorithms. By exploiting Data-Level Parallelism (DLP), vector processors can achieve high throughput while maintaining energy and area efficiency—key advantages for edge AI. Unlike fixed-width Single Instruction-Multiple Data (SIMD) models, the vector extension supports variable-length registers, allowing developers to tailor vector operations to different applications and hardware budgets.

This flexibility and efficiency position RISC-V vector units as a strong alternative to traditional GPUs and multicore processors. Compared to the latter, vector units offer improved power efficiency and simpler programming models, while against GPUs, they deliver better area utilization and lower energy consumption. These features make vector architectures—especially in the RISC-V ecosystem—a compelling choice for building general-purpose yet efficient accelerators that integrate emerging paradigms like IMC.

Within this context, this work makes three main contributions:

First, we present an efficient method for tightly coupling a DIMC unit within the vector processor pipeline. This integration enables the DIMC to be fully exploited by extending workload mapping flexibility at the assembly instruction level and an efficient utilization across a wide range of current and emerging workloads.

Second, we propose a custom ISA extension to the RISC-V vector standard, designed to manage the DIMC integration efficiently. Four new vector instructions are proposed: two for loading data into the DIMC and two for managing the computation start and write-back operations. These instructions maximize the unit's bandwidth and throughput while maintaining compatibility with the RISC-V vector instruction encoding, ensuring ease of reuse in future architectures.

Third, we prove the effectiveness of our integration on an industrial-level RISC-V vector core. Experimental results on ResNet50 show that embedding a single DIMC tile yields up to 137 GOPS and over 200× speedup compared to the baseline, even under tight hardware constraints. This baseline-oriented comparison highlights the substantial incremental benefit enabled by our proposed integration method.

Paper Structure. Section II reviews state-of-the-art RISC-V architectures integrating IMC. Section III details the proposed architectural design. Section IV introduces the ISA extension. Section V presents the experimental setup and performance results. Section VI concludes with a summary and future directions.

II. State of the Art

Recent research has explored the integration of In-Memory Computing (IMC) into RISC-V-based processors to improve the efficiency of Deep Learning inference. While both analog and digital IMC architectures have demonstrated significant potential, their integration within programmable processors, especially vector-capable ones, remains limited. This section reviews representative works that integrate IMC units into RISC-V general-purpose or vector processors, distinguishing between tightly and loosely coupled models. The analysis emphasizes reported performance in terms of peak throughput (GOPS or TOPS) and supported precision, and positions them with respect to our approach.

A. Tightly Coupled IMC in RISC-V Cores

Tightly coupled architectures integrate IMC units directly into the RISC-V pipeline, enabling low-latency communication and instruction-level control. For instance, **AI-PiM** [12] extends the RV64IMC ISA with custom PiM instructions, integrating compute-in-memory units within the processor pipeline. This design achieves speedups of up to 17.63× in matrix-vector multiplication and an average improvement of 2.74× across MLPerf Tiny benchmarks. However, AI-PiM targets scalar pipelines, and therefore lacks the scalability required for highly parallel workloads.

A different approach is taken by **Vecim** [13], which integrates an 8T SRAM-based CIM architecture as a Vector Register File within a RISC-V vector core. Supporting multiple precision formats including INT8, BF16, and FP16, the architecture achieves a peak performance of 31.8 GOPS at 250 MHz. Nevertheless, while Vecim benefits from vector capabilities, its design primarily emphasizes register-file integration rather than functional unit extension, limiting its flexibility in supporting diverse computational patterns.

RDCIM [14] instead presents a fully digital CIM processor tightly coupled with a RISC-V core through extended instructions for fine-grained control. It incorporates techniques such as Adding-on-Memory-Boundary (AOMB) and a Multi-Precision Adaptive Accumulator (MPAA) to reduce power and area overheads, supporting 4/8/12/16-bit precision with high energy efficiency. The design demonstrates 66.3 TOPS/W in 4-bit mode and 16.6 TOPS/W in 8-bit mode. Although RDCIM introduces valuable reconfigurable features, it remains focused on scalar pipelines, limiting its applicability to workloads requiring wide vector-level parallelism.

B. Loosely Coupled IMC in RISC-V Cores

Loosely coupled architectures extend RISC-V cores with IMC engines functioning as co-processors, communicating through buses or custom synchronization schemes. **VPU-CIM** [15], for example, integrates RRAM-based CIM with vector ISA extensions, achieving 33.98 TOPS/W with support for variable-bit precision (1–4 bits). However, since its CIM operations are decoupled from the main pipeline, this approach suffers from additional communication overhead and reduced fine-grained control.

Similarly, **CIMR-V** [16] embeds a 10T SRAM-based digital CIM engine alongside a convolution and pooling pipeline, controlled via dedicated instructions. Fabricated in TSMC 28nm, it reaches 26.2 TOPS at 50 MHz for keyword spotting tasks, demonstrating the potential of IMC for domain-specific acceleration. Yet, the offloading model introduces latency and constrains flexibility in AI workloads.

In summary, prior works demonstrate either feasibility (scalar tight coupling) or energy efficiency (vector loose coupling), but none integrate a DIMC directly into the vector pipeline. Our approach fills this gap by embedding DIMC as a functional unit, enabling tight control, scalable vector processing, and efficient execution of complex dataflows while preserving programmability through the standard vector ISA.

III. Proposed DIMC integration within the RISC-V Vector Pipeline

The DIMC unit used in this work is the one presented in [9] and shown in Fig. 2. It features a memory capacity of 32 KiB, organized into 32 rows of 1024 bits each, typically used to store kernel weights during convolution execution. In addition, a 1024-bit input buffer is available to store the feature data used in the computation. The latter, takes place directly between the input buffer and a selected row of the memory, enabling in-memory execution of MAC operations and minimizing data movement overhead.

Internally, the DIMC architecture is structured into multiple sub-arrays, each consisting of 8T 1R1W bitcells and equipped with independent read word lines (RWLs) and read bitlines (RBLs). These sub-arrays can operate either as a unified array in memory-mapped mode or as independent units in compute mode. During computation, multiple RBLs are accessed in parallel within each sub-array, and the resulting signals are sensed and routed through dedicated IO paths to interleaved MAC slices.

This setup supports 256 parallel signed or unsigned 4-bit MAC operations per cycle, implemented as four parallel sub-arrays of 64 MACs each, all sharing a common accumulation pipeline. The architecture also supports runtime reconfiguration, allowing the same hardware to perform 512 2-bit or 1024 1-bit MAC operations per cycle, offering flexibility in precision. This configurability enables a scalable trade-off between accuracy and efficiency, making the DIMC ideal for edge AI inference. The memory interface supports 256-bit data transfers per cycle for both read and write operations. The computation produces 24-bit partial results, which can optionally be passed through a ReLU activation stage before being written back.

We integrate this DIMC as a dedicated execution lane in a RISC-V core implementing the embedded vector extension profile Zve32x, with architectural parameters VLEN = 64 and ELEN = 32, based on the industrial RVV implementation described in [17] (Fig. 3). The unit receives operands and decoded instructions like any other FU, but is driven by new custom vector instructions introduced in Sec. IV.These instructions orchestrate high-bandwidth data loading, fine-grained configuration, and low-overhead result storage, all while allowing the DIMC lane to run in parallel with standard vector FUs.

Why a vector core matters. Beyond raw MAC throughput, overall performance hinges on how quickly feature maps and kernels can be folded, packed, or transposed into the 256-bit stripes the DIMC consumes. The RISC-V vector ISA already provides the data-manipulation primitives needed, and the VRF exposes sufficient read/write ports to match the DIMC bandwidth. These allow software to reshape irregular, dilated, grouped, or highly quantized tensors entirely within the VRF. Consequently, the vector core acts as a flexible "data-manipulator" that feeds a high-performance DIMC compute engine without extra glue hardware; the only cost is a modest cycle overhead inside the VRF. This synergy vastly broadens the range of convolution workloads the architecture can support.

This work concentrates on a single DIMC tile, with the primary goal of defining efficient integration and control within a vector core. Scaling to multiple tiles naturally follows as future work.

IV. Proposed RISC-V Instruction Set Extension

Integrating a non-standard functional unit like the DIMC into a RISC-V vector core requires dedicated custom instructions. While a simple hardware connection may enable basic operation, fully

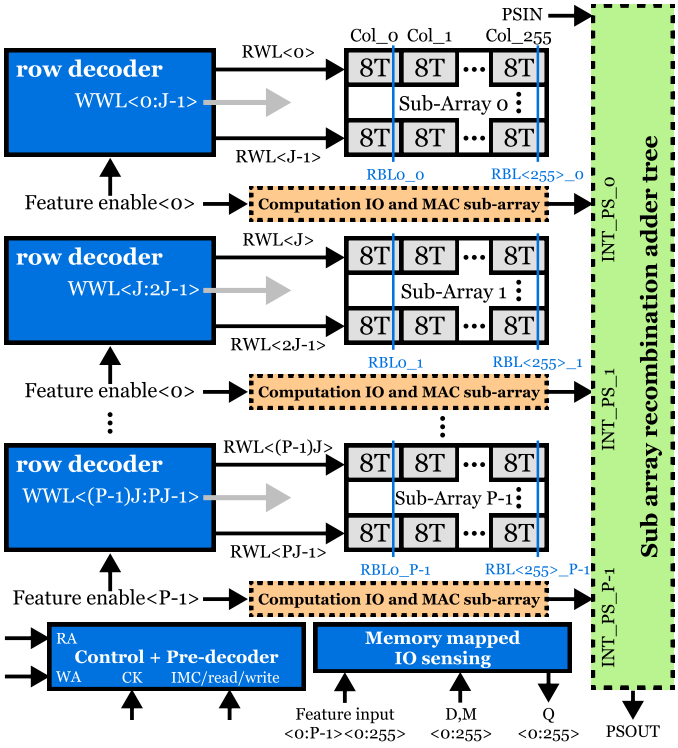

Fig. 2. Architecture of DIMC Tile (P sub-arrays, J rows each) as presented at ISSCC 2023 [9]

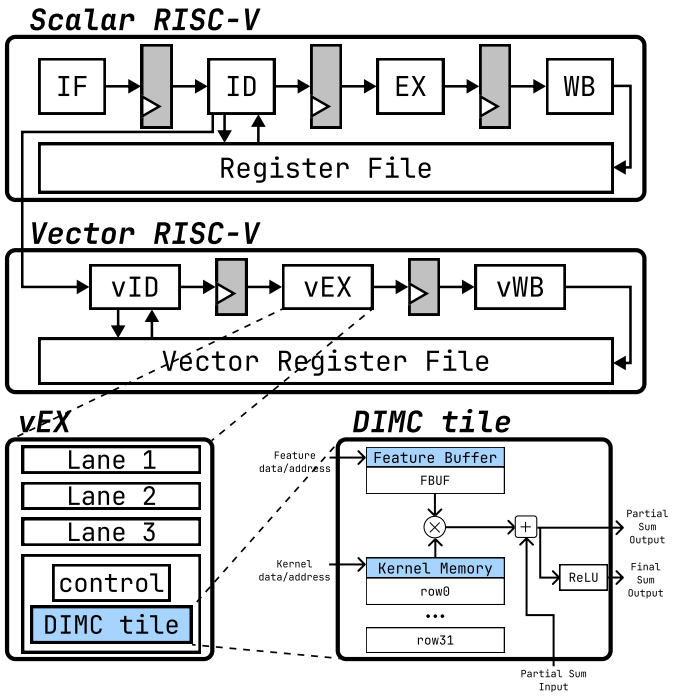

Fig. 3. Architecture overview with the DIMC tile Integrated in the Vector Execution Stage as a Parallel Execution Lane

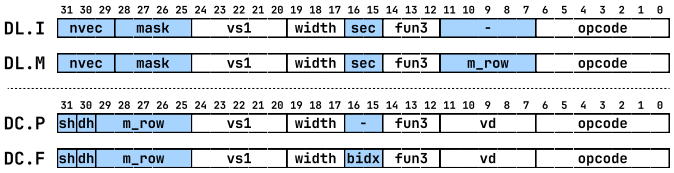

Fig. 4. Bit-level encoding of the proposed custom instructions for DIMC accelerator integration. Customized bit fields are highlighted in blue.

leveraging the DIMC's capabilities requires tight integration and instruction-level customization. This includes fine-grained control of data movement, execution order, and result handling to maximize in-memory convolution efficiency.

Although physically integrated into the Execution Stage of the pipeline, the DIMC unit operates under a specialized execution model that differs from standard vector FUs:

1) Data loading: The DIMC must first be loaded with both weights and input data before computation can begin.
2) Computation: The DIMC performs in-memory MAC operations directly between the input buffer and selected memory rows, and requires explicit control over the target rows, applied masks, and computation precision.
3) Result Write-Back: Partial sums or final outputs of each row are generated sequentially, one per cycle. If not properly synchronized, these results can be lost, making precise timing and control essential.

To support this model, the custom instructions must be carefully designed. To this end, a custom RISC-V vector ISA extension has been developed, comprising four vector-based instructions tailored to optimize the integration of the DIMC unit into the execution pipeline. These instructions enable:

- High bandwidth loading of feature and weights data from the VRF to the DIMC;
- Seamless execution of MAC operations within the DIMC unit, with full control over configuration parameters;
- Optimized handling and storage of partial and final results with minimal memory overhead.

Isolating the DIMC as a functional unit enables parallelism with vector units, avoids access conflicts, reduces memory traffic, and removes coherence issues by routing all exchanges through the VRF.

A. Proposed RISC-V Custom Instructions

To integrate the DIMC accelerator efficiently into the RISC-V vector pipeline, we propose a small set of specialized vector instructions. These instructions are organized into two distinct functional groups based on their roles:

- Data Load Instructions (`DL.I`, `DL.M`) — manage high-bandwidth transfers respectively from the VRF to the Input Buffer and DIMC Memory.
- Compute and Write-Back Instructions (`DC.P`, `DC.F`) — trigger MAC operations inside DIMC, handle the associated configuration signals, and manage the efficient write-back of partial or final results to the VRF.

The bit-level encoding of these custom instructions is detailed in Fig. 4, where customized fields are highlighted in blue.

DIMC Input Buffer Load (DL.I). The `DL.I` instruction transfers between 64 and 256 bits of data from the Vector Register File (VRF) to the DIMC's input buffer. It reads data from `nvec` consecutive VRF registers, starting at `vs1`, using a valid-bit `mask`. The loaded data is written into one of the four 256-bit sectors, specified by `sec`, within the DIMC's 1024-bit input buffer.

DIMC Memory Load (DL.M). The `DL.M` instruction operates similarly to `DL.I`, but includes an additional field, `m_row`, which specifies the DIMC memory row where the data should be loaded.

DIMC Compute & Partial Sum Store (DC.P). The `DC.P` instruction performs an in-memory MAC operation between the data in the DIMC input buffer and the weights stored in the specified memory row (`m_row`). It takes a 24-bit partial sum as input from the half of the `vs1` register selected by `sh`, and stores the resulting 24-bit partial result into the half of the `vd` register specified by `dh`.

DIMC Compute & Final Sum Store (DC.F). The `DC.F` instruction performs the same in-memory MAC operation as `DC.P`, but additionally applies the ReLU activation function and stores the final quantized result in compact form. The result is written into a specific byte of the half of the `vd` register in the VRF, selected by the `dh` (half selector) and `bidx` (byte index) fields.

Partial sums generated by `DPS` are padded from 24 bits to 32 bits for VRF alignment, with possible future optimizations to reduce overhead. Final outputs from `DSS`, quantized to 1-, 2-, or 4-bit precision, are padded to 4 bits and efficiently packed into bytes—two 4-bit results per byte. In the case of an odd number of results, the last half-byte remains unused, defining the maximum padding overhead.

These custom instructions collectively provide a concise yet powerful ISA extension, enabling streamlined integration and operation of the DIMC accelerator within a vector RISC-V core. From an encoding perspective, the four instructions are mapped to the RISC-V custom-0 space, which is explicitly reserved for non-standard extensions. This guarantees that no conflicts arise with current vector extensions, and sufficient encoding space remains available to accommodate further custom instructions in future work.

V. Experimental Results

This section outlines the experimental setup and the approach used to evaluate the performance of the DIMC-enhanced RVV processor against the baseline core. The results of the comparison highlight three key achievements: (1) the high utilization of the DIMC tile enabling near-peak computational throughput, (2) the significant speedup achieved with and without area normalization, and (3) the robust performance sustained even when DIMC hardware capacity limits are exceeded. We focus on comparison with the baseline RVV core because the objective of this work is to demonstrate the incremental benefit that DIMC integration brings to an established industrial-grade architecture. In other words, the emphasis is on quantifying the performance gain achievable by directly embedding DIMC into the vector pipeline, rather than on outperforming unrelated accelerator designs.

A. Experimental Setup

The evaluation was conducted using a cycle-approximate simulation framework developed specifically for the DIMC-enhanced RVV processor. This custom simulator models instruction-level execution with timing granularity sufficient to capture the interplay between the core pipeline and the tightly coupled DIMC unit.

The simulation tool generates execution traces from custom instruction streams derived from deep learning models. Each instruction is assigned a latency based on the hardware pipeline structure and stall conditions. The simulator explicitly models:

- Instruction latencies, including vector loads, stores, and arithmetic operations;
- Pipeline stalls and execution flow control;

- Custom DIMC instruction timing, which reflects the internal datapath latency and tightly coupled access to the registers.

Area estimates for both the baseline and DIMC-enhanced architectures were obtained through RTL synthesis using Cadence tools, targeting a P18 CMOS process node from STMicroelectronics. These values are used to compute area-normalized performance metrics. While energy measurements are not reported in this work, future iterations may include RTL-based power estimates or model-based energy approximations.

The benchmark used for evaluation is ResNet50, from which individual convolutional and fully connected layers were extracted and translated into instruction streams. Each layer is mapped to the DIMC-enhanced RVV processor through a toolchain that:

1) Load kernel weights into the DIMC memory (up to 32 kernels).
2) Load one patch of feature data into the DIMC input buffer.
3) Trigger MAC operations using custom compute instructions.
4) Slide input window across the feature map and repeat 2–3.
5) Reload kernels if needed and continue the iteration.

All MAC operations are executed inside the DIMC, with the RVV core orchestrating data movement and issuing compute instructions via the extended ISA. The simulator itself is not publicly released due to industrial confidentiality constraints.

Assumptions. The following assumptions were made in the simulations to simplify the analysis, clearly define experimental boundaries, and avoid the complexities associated with advanced data mapping strategies and compiler modifications:

- *Vector Instruction Issuing*: Simulations did not consider double-issue vector instruction execution, simplifying modeling at the expense of capturing peak theoretical performance.
- *Memory Access Latency*: Although memory access is not modeled cycle-by-cycle, a fixed-latency external memory is assumed. This is sufficient for our analysis, since all data exchanges with the DIMC are tightly coupled and do not involve DMA.
- *Data Fetching and Reuse*: Simulations involving DIMC assumed that each feature map was loaded directly from memory, even in cases where data reuse could reduce memory accesses. This conservative assumption leads to sub-optimal results.
- *Resolution Limitation*: The baseline RVV architecture supports a minimum data resolution of 8 bits, while the DIMC unit used is limited to a maximum of 4 bits.
- *DIMC Capacity Constraint*: DIMC-supported convolutions were limited to kernels where the total bits per single channel did not exceed 1024 bits.
- *Layer Type Acceleration:* DIMC acceleration specifically targets convolutional and fully connected layers. Operations such as pooling, which rely on the standard RVV ISA, yield similar performance across both baseline and DIMC-enhanced architectures and were thus excluded from simulations results.

Performance Evaluation Metrics. We evaluated the following three performance metrics:

1) OPs (Operations Per Second): The total number of operations executed per second, calculated based on a 500 MHz clock frequency. This metric provides a direct measure of the system's computational throughput.
2) Speedup: The ratio of clock cycles executed by the baseline RVV processor compared to the DIMC-enhanced version, quantifying the relative performance improvement:

$$Speedup = \frac{\text{ClockCycles}_{\text{Baseline RVV}}}{\text{ClockCycles}_{\text{DIMC RVV}}}$$

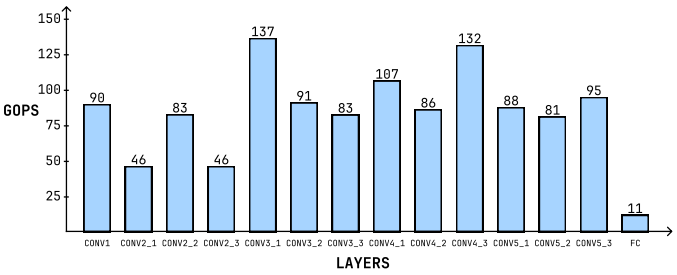

Fig. 5. GOPS Achieved per Layer in ResNet50

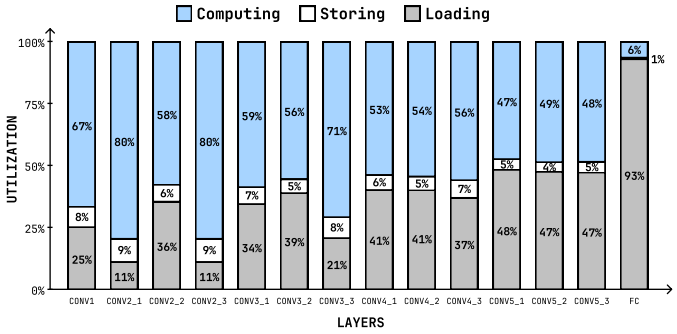

Fig. 6. Operation Distribution (Computing, Loading, Storing) per Layer in ResNet50

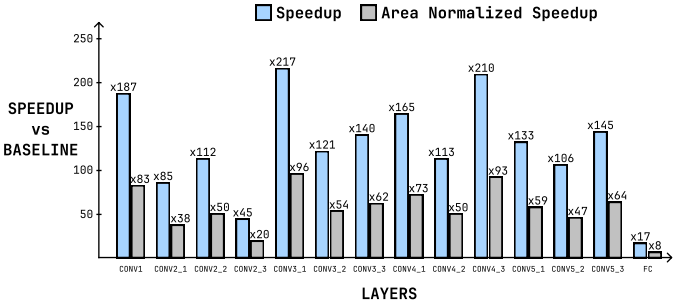

Fig. 7. Speedup and Area-Normalized Speedup per Layer in ResNet50

3) Area-Normalized Speedup: The speedup metric normalized with respect to the baseline RVV area, allowing performance improvements to be evaluated in the context of increased hardware complexity.

$$ANS = Speedup * \frac{Area_{Baseline}}{Area_{DIMCRVV}}$$

B. Efficient Utilization of the DIMC Tile

To evaluate the computational throughput achieved by the DIMC accelerator, simulations were performed on each convolutional and fully connected layer in ResNet50 [20]. As shown in Fig. 5, the DIMC-enhanced processor reaches peak throughput values close to its theoretical limit (based on MAC unit performance and assuming no loading or storing penalties), achieving over 100 GOPS in many layers and peaking at 137 GOPS.

This high performance is explained by the efficient scheduling and utilization of the DIMC tile, thanks to the new custom instructions obtained through the extension of the instruction set architecture. The breakdown of operations per layer in Fig. 6 confirms that the DIMC spends the majority of execution time on computation rather than data loading or result storing, validating its ability to exploit hardware resources effectively. This result highlights the value of

TABLE I
COMPARISON OF IMC-INTEGRATED RISC-V ARCHITECTURES

	Scalar/Vector	Integration	Memory type	Memory size	Freq. [MHz]	Reported Perf.	Norm. GOPS[1]
CIMR-V [16]	Scalar	Loose	10T SRAM [18]	64 KB	50	26.2 TOPS @INT1	∼2.6 TOPS @INT4, 500 MHz*
AI-PiM [12]	Scalar	Tight (In-Pip.)	8T SRAM [19]	500 B	–	–	–
VPU-CIM [15]	Vector	Loose	RRAM	8 KB	25	–	–
Vecim [13]	Vector	Tight	8T SRAM	–	250	31.8 GOPS @INT8	∼63.6 GOPS @INT4, 500 MHz*
RDCIM [14]	Scalar	Tight	8T SRAM	64 KB	200	–	–
This Work	**Vector**	**Tight (In-Pip.)**	**8T SRAM**	**4 KB**	**500**	**137 GOPS @INT4**	**137 GOPS @INT4**

the In-Pipeline integration design in a Vector RISC-V core, which enables a high compute operation ratio.

C. Speedup over Baseline RVV Core

The integration of DIMC leads to a substantial speedup over the baseline RVV core. As shown in Fig. 7, across all ResNet50 layers, the DIMC-enhanced architecture consistently outperforms the baseline, with raw speedup values exceeding 200× in some layers and area-normalized speedup maintaining values well above 50×. These results demonstrate that even when accounting for the additional hardware cost, the tightly integrated DIMC provides significant performance advantages for edge AI workloads.

D. Analysis of the Accelerated Workload

To assess flexibility and generalization, we analyzed over 450 convolutional layers from models including AlexNet [1], VGG16 [21], ResNet [20], Inception [22], DenseNet [23], EfficientNet [24], and MobileNet [25]. Covering a broad range of feature maps and kernel dimensions, these configurations represent real-world scenarios. Across them, the DIMC-augmented system consistently outperforms the baseline, demonstrating high versatility.

While Table I compares representative architectures, our evaluation focuses on the baseline RVV core to isolate the impact of DIMC integration. Unlike prior works limited to layers fitting architectural constraints, our analysis includes configurations that exceed system limits. This stresses the flexibility of the system in scenarios requiring frequent kernel switching and sub-optimal memory mapping. The two primary architectural constraints are:

- a maximum single-kernel bitwidth of 1024 bits (requires *tiling*);
- a limit of 32 kernel in DIMC memory (requires *grouping*).

Fig.8 shows how *tiling* is employed when the kernel size exceeds the 1024-bit threshold. While this incurs a performance drop due to serial loading and computation, the DIMC architecture still maintains a strong advantage over the baseline. Similarly, Fig.9 evaluates the case of *grouping*. Despite the forced segmentation of compute, the architecture sustains notable speedup.

These results underline not just the high baseline throughput of the DIMC tiles, but more importantly, the architectural strategy introduced in this work. The combination of a streamlined custom instruction set (DL.I, DL.M, DC.P, and DC.F) and its tight integration with the RISC-V Vector Extension enables precise control over kernel loading, compute scheduling, and data reuse. This coordination is what allows the system to adapt dynamically to sub-optimal scenarios—such as fragmented kernel shapes or frequent weight switching—without losing efficiency. Rather than tailoring workloads to fit static hardware constraints, the proposed architecture adapts to the workload, sustaining high tile utilization and acceleration across diverse conditions. This makes it a practical and robust solution for applications with variable and unpredictable inference patterns.

[1]Normalized GOPS values with our work's precision and frequency.

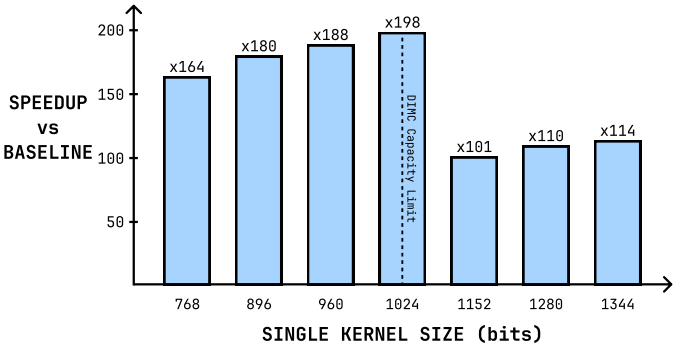

Fig. 8. Speedup Degradation due to tiling (OCH=32, KH=2, KW=2)

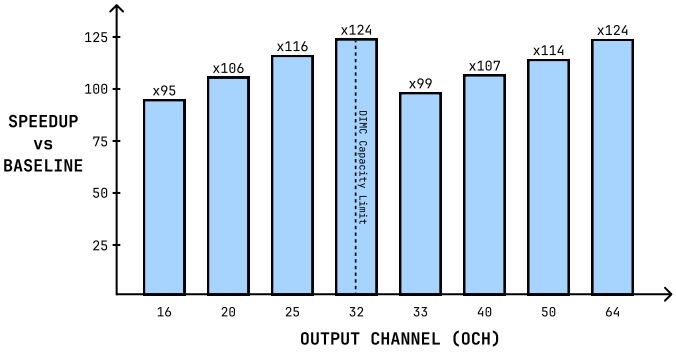

Fig. 9. Speedup Degradation due to grouping (ICH=32, KH=2, KW=2)

E. Summary and Comparison with Prior Architectures

Finally, to contextualize the achieved results, Table I compares the proposed architecture against representative IMC-integrated RISC-V processors, highlighting differences in integration strategy, core type, and peak performance.

Unlike prior works, this is the first design to tightly integrate a DIMC unit within a vector core pipeline as an FU. It achieves the highest reported performance (137 GOPS at INT4) while operating at 500 MHz with only 4 KB of DIMC memory. Compared to scalar or loosely coupled designs, our architecture offers superior throughput and control with minimal memory overhead.

VI. CONCLUSIONS

This work demonstrates the first in-pipeline integration of a DIMC unit within a RISC-V vector processor, supported by a custom ISA extension for fine-grained control of dataflow and computation. By extending an industrial-grade vector core with a single DIMC tile, we show that the proposed integration model reaches up to 137 GOPS and more than 200× speedup over the baseline, with over 50× gain even when area-normalized. Focusing on a single tile allowed us to precisely define how a vector core can efficiently manage and exploit

DIMC units, laying the foundation for future scaling to multiple tiles. Unlike prior designs that require workload tailoring, our architecture maintains high performance across diverse computations, making it a scalable and adaptive accelerator for next-generation edge AI.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 25. Curran Associates, Inc., 2012.

[2] "Data on Notable AI Models," Jun. 2024. [Online]. Available: https://epoch.ai/data/notable-ai-models

[3] "Data on Machine Learning Hardware," Oct. 2024. [Online]. Available: https://epoch.ai/data/machine-learning-hardware

[4] C. Silvano, D. Ielmini, F. Ferrandi, L. Fiorin, S. Curzel, L. Benini, F. Conti, A. Garofalo, C. Zambelli, E. Calore, S. F. Schifano, M. Palesi, G. Ascia, D. Patti, N. Petra, D. D. Caro, L. Lavagno, T. Urso, V. Cardellini, G. C. Cardarilli, R. Birke, and S. Perri, "A Survey on Deep Learning Hardware Accelerators for Heterogeneous HPC Platforms," Jul. 2024, arXiv:2306.15552. [Online]. Available: http://arxiv.org/abs/2306.15552

[5] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. San Francisco, CA, USA: IEEE, Feb. 2014, pp. 10–14. [Online]. Available: http://ieeexplore.ieee.org/document/6757323/

[6] S. Kim and H.-J. Yoo, "An Overview of Computing-in-Memory Circuits With DRAM and NVM," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 3, pp. 1626–1631, Mar. 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10320372/

[7] J.-M. Hung, Y.-H. Huang, S.-P. Huang, F.-C. Chang, T.-H. Wen, C.-I. Su, W.-S. Khwa, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, Y.-D. Chih, T.-Y. J. Chang, and M.-F. Chang, "An 8-Mb DC-Current-Free Binary-to-8b Precision ReRAM Nonvolatile Computing-in-Memory Macro using Time-Space-Readout with 1286.4-21.6TOPS/W for Edge-AI Devices," in *2022 IEEE International Solid- State Circuits Conference (ISSCC)*. San Francisco, CA, USA: IEEE, Feb. 2022, pp. 1–3. [Online]. Available: https://ieeexplore.ieee.org/document/9731715/

[8] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature Electronics*, vol. 1, no. 6, pp. 333–343, Jun. 2018. [Online]. Available: https://www.nature.com/articles/s41928-018-0092-2

[9] G. Desoli, N. Chawla, T. Boesch, M. Avodhyawasi, H. Rawat, H. Chawla, V. Abhijith, P. Zambotti, A. Sharma, C. Cappetta, M. Rossi, A. De Vita, and F. Girardi, "16.7 A 40-310TOPS/W SRAM-Based All-Digital Up to 4b In-Memory Computing Multi-Tiled NN Accelerator in FD-SOI 18nm for Deep-Learning Edge Applications," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2023, pp. 260–262, iSSN: 2376-8606. [Online]. Available: https://ieeexplore.ieee.org/document/10067422/?arnumber=10067422

[10] H. Fujiwara, H. Mori, W.-C. Zhao, M.-C. Chuang, R. Naous, C.-K. Chuang, T. Hashizume, D. Sun, C.-F. Lee, K. Akarvardar, S. Adham, T.-L. Chou, M. E. Sinangil, Y. Wang, Y.-D. Chih, Y.-H. Chen, H.-J. Liao, and T.-Y. J. Chang, "A 5-nm 254-TOPS/W 221-TOPS/mm^2 Fully-Digital Computing-in-Memory Macro Supporting Wide-Range Dynamic-Voltage-Frequency Scaling and Simultaneous MAC and Write Operations," in *2022 IEEE International Solid- State Circuits Conference (ISSCC)*. San Francisco, CA, USA: IEEE, Feb. 2022, pp. 1–3. [Online]. Available: https://ieeexplore.ieee.org/document/9731754/

[11] "riscv-v-spec/v-spec.adoc at master · riscvarchive/riscv-v-spec." [Online]. Available: https://github.com/riscvarchive/riscv-v-spec/blob/master/v-spec.adoc

[12] V. Verma and M. R. Stan, "AI-PiM—Extending the RISC-V processor with Processing-in-Memory functional units for AI inference at the edge of IoT," *Frontiers in Electronics*, vol. 3, Aug. 2022, publisher: Frontiers.

[13] Y. Wang, M. Yang, C.-P. Lo, and J. P. Kulkarni, "30.6 Vecim: A 289.13GOPS/W RISC-V Vector Co-Processor with Compute-in-Memory Vector Register File for Efficient High-Performance Computing," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*. San Francisco, CA, USA: IEEE, Feb. 2024, pp. 492–494. [Online]. Available: https://ieeexplore.ieee.org/document/10454387/

[14] W. Yi, K. Mo, W. Wang, Y. Zhou, Y. Zeng, Z. Yuan, B. Cheng, and B. Pan, "Rdcim: Risc-v supported full-digital computing-in-memory processor with high energy efficiency and low area overhead," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 4, pp. 1719–1732, 2024.

[15] C. M. J, V. Rayapati, N. Rao, and M. Suri, "VPU-CIM: A 130nm, 33.98 TOPS/W RRAM based Compute-In-Memory Vector Co-Processor," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. Singapore, Singapore: IEEE, May 2024, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/10558155/

[16] Y.-C. Guo, T.-S. Chang, C.-S. Lin, B.-C. Chiou, C.-M. Lai, S.-S. Sheu, W.-C. Lo, and S.-C. Chang, "CIMR-V: An End-to-End SRAM-based CIM Accelerator with RISC-V for AI Edge Device," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. Singapore, Singapore: IEEE, May 2024, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/10558177/

[17] L. Bordonaro, E. Rapuano, S. Bocchio, and L. Fanucci, "Integration of the Open-Source RISC-V Formal Verification Framework into the STMicroelectronics Toolchain and Its Application to the STRiVe2.4 CPU," in *Applications in Electronics Pervading Industry, Environment and Society*, M. Ruo Roch, F. Bellotti, R. Berta, M. Martina, and P. Motto Ros, Eds. Cham: Springer Nature Switzerland, 2025, pp. 48–55.

[18] C.-S. Lin, F.-C. Tsai, J.-W. Su, S.-H. Li, T.-S. Chang, S.-S. Sheu, W.-C. Lo, S.-C. Chang, C.-I. Wu, and T.-H. Hou, "A 48 tops and 20943 tops/w 512kb computation-in-sram macro for highly reconfigurable ternary cnn acceleration," in *2021 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 2021, pp. 1–3.

[19] Q. Dong, M. E. Sinangil, B. Erbagci, D. Sun, W.-S. Khwa, H.-J. Liao, Y. Wang, and J. Chang, "15.3 a 351tops/w and 372.4gops compute-in-memory sram macro in 7nm finfet cmos for machine-learning applications," in *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2020, pp. 242–244.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[21] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Apr. 2015, arXiv:1409.1556 [cs]. [Online]. Available: http://arxiv.org/abs/1409.1556

[22] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," Sep. 2014, arXiv:1409.4842 [cs]. [Online]. Available: http://arxiv.org/abs/1409.4842

[23] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," Jan. 2018, arXiv:1608.06993 [cs]. [Online]. Available: http://arxiv.org/abs/1608.06993

[24] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," Sep. 2020, arXiv:1905.11946 [cs]. [Online]. Available: http://arxiv.org/abs/1905.11946

[25] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," Apr. 2017, arXiv:1704.04861 [cs]. [Online]. Available: http://arxiv.org/abs/1704.04861