SecPIMlet: Confidential Computing for Spatio-temporally Shared Processing-in-Memory

Sunho Lee[†*], Minwoo Noh[‡], Jeongwon Choi[‡], Kwanghoon Choi[‡], Igjae Kim[‡], Chang Hyun Park[§], Jaehyuk Huh[‡]

[†]*University of Oxford*
myshlee417@gmail.com

[‡]*KAIST*
{mwnoh, jwchoi, khchoi, ijkim}@casys.kaist.ac.kr, jhhuh@kaist.ac.kr

[§]*Uppsala University*
chang.hyun.park@it.uu.se

***Abstract*—As memory-intensive applications, particularly language models, continue to grow, processing-in-memory (PIM) has emerged as a promising architecture to alleviate memory bandwidth limitations. We find that PIM systems need to support both spatio and temporal sharing between domains to extract the maximal performance. However, current PIM systems lack the necessary protection to securely support such sharing. To address these issues, we introduce *SecPIMlet*, a trusted multi-domain PIM that enables efficient spatio-temporal sharing. For fast and confidential spatio-temporal sharing, *SecPIMlet* protects both PIM-resident data and execution contexts, while also securing in-transit data between the CPU and PIM. Our evaluation shows that *SecPIMlet* achieves at least 175.4% higher throughput than state-of-the-art approaches that support only spatial or temporal sharing, while providing stronger security protection.**

I. Introduction

On-die processing-in-memory (PIM) architecture has emerged as an energy-efficient accelerator for memory bandwidth-intensive workloads [16], [33], [51], [54], [56]. Most of the PIM studies have focused on improving the performance of PIM-based acceleration for diverse workloads [53], [63], [89], but are limited in serving a single application context (hereafter defined as a *domain*) at a time. However, in modern systems, specifically client-side AI systems, where several domains are co-running, a single workload cannot monopolize the entire PIM resources due to other co-running domains and because some domains show poor scalability, resulting in diminishing returns of monopolizing the PIM. Following the trends in multi-domain GPUs and NPUs [12], [13], [17], [21], [78], sharing PIM resources among multiple domains is critical to improve utilization.

Sharing the PIM in both the temporal and spatial axis is necessary to extract the most out of the PIMs on multi-domain systems. Spatial sharing distributes distinct PIM resources among multiple concurrent domains, whereas temporal sharing switches the execution context of multiple domains within a single PIM resource over time. A prior work investigated the throughput benefit of spatial sharing of PIM resources [48], and another showed the latency benefit, specifically service level objectives (SLOs), by preventing short, latency-critical domains from being blocked by long-running ones through fine-grained temporal sharing [13]. We find that achieving both spatial and temporal sharing is necessary to improve utilization and SLO. Our motivational experiments demonstrate that existing spatial-only [48] or temporal-only [93] schemes attain at most 36.3% of the throughput and incur at least 87.6% higher latency compared to an efficient spatio-temporal sharing.

A PIM that supports multi-domain spatio-temporal sharing must ensure isolation and protection across domains. Client-side AI systems process private user data while hosting proprietary weights, naturally demanding strict PIM security. Similarly to modern GPUs, where confidential computing has become mandatory to ensure hardware-enforced isolation and to protect sensitive data [29], [44], [68], [72], [73], [84], PIM architectures must also deliver comparable guarantees. GPU confidential computing safeguards not only the confidentiality and integrity of model parameters and intermediate data but also enforces rigid execution and memory isolation boundaries among co-located domains.

Designing a PIM that provides secure spatio-temporal sharing poses **three unique challenges** due to the integration of processing elements (PEs) near DRAM banks on the same die. First, multiple domains co-located on a PIM PE need memory isolation to prevent one domain from accessing the data of another domain. While this can be achieved through memory virtualization, existing PIM virtualization schemes offer limited flexibility in mapping virtual addresses to PIM memory. Specifically, they are restricted to either spatial [48] or temporal [93] sharing, rather than enabling both simultaneously. Second, fine-grained temporal sharing requires a secure, low-overhead context switching mechanism. As each PIM PE can host multiple domains with spatio-temporal sharing, a lack of a secure context switching mechanism within PIM memory can leak data across the domains, and thus requires saving and restoring the context for each domain on the host memory, imposing significant overhead. Lastly, the PIM needs

[*] This work was done at KAIST.

to be protected against physical attacks. The communication channel (DIMM) between the host CPU and the PIM are exposed to physical attacks, requiring secure communication mechanisms. The PIM PE must be able to access plaintext data, but adding an encryption/authentication engine for each PIM PE is not feasible due to the area overheads. Furthermore, as the host core and the PIM PE have different data layout (chip-level interleaved order vs. continuous order), existing channel encryption/authentication schemes do not work with the PIM-side continuous order.

We propose ***SecPIMlet***, a trusted multi-domain PIM architecture that supports both spatial and temporal sharing. To the best of our knowledge, *SecPIMlet* is the first to introduce **spatio-temporal sharing to the PIM landscape**. While such sharing is a challenge in itself for unsecure environments, *SecPIMlet* takes a step further and specifically addresses **the unique advantages of integrating it with confidential computing**, enhancing both performance and security. *SecPIMlet* ensures strong domain isolation through 1) **chip-granular address translation** and 2) **hardware-enforced access control with PIM-offloaded context switching**. Building on these, *SecPIMlet* introduces 3) **per-chip and layout-aware secure CPU-PIM communication channel** to protect against physical attacks on CPU–PIM communication paths. From a performance perspective, 4) *SecPIMlet* proposes a simple yet effective **spatio-temporal scheduling** mechanism that improves resource utilization and overall system throughput by balancing spatial and temporal resources. We introduce the four elements of *SecPIMlet* in the following paragraphs.

The **chip-granular address translation** scheme adopts a finer-grained address translation mechanism. Unlike prior works [48], [93], *SecPIMlet* places no restrictions on translation both within and across PIM chips, thereby enabling spatio-temporal sharing. Although finer granularity typically enlarges the translation table, our design keeps the PIM-side table size unchanged, making this design feasible without additional area overhead. Moreover, *SecPIMlet* provides a secure translation mechanism by isolating the translation table management to the trusted hardware.

The **hardware-enforced access control with PIM-offloaded context switching** provides a fast and secure context switching mechanism by retaining the execution context within each PIM bank. The procedure is securely managed by per-chip hardware, which serves as the sole component authorized to access the stored context. Upon generation of a context switching request, the request is conveyed to the per-chip hardware via a dedicated instruction. After that, the hardware identifies the corresponding context region and performs context load and store operations accordingly.

The **per-chip and layout-aware secure CPU-PIM communication channel** protects the system against physical attacks. The discrete data layouts of the CPU and PIM PEs are managed through an intermediate buffer-based encryption and integrity mechanism that incorporates only a single additional buffer for security metadata. Also, by consolidating the protection engine at the per-chip level, *SecPIMlet* substantially lowers the required throughput of the engine from PIM internal bandwidth to CPU-PIM bandwidth, reducing hardware cost.

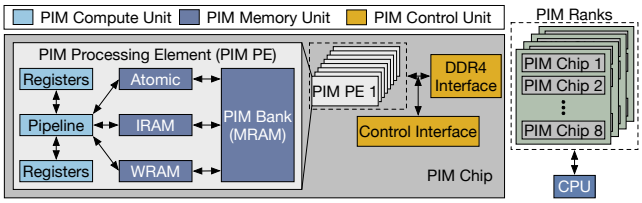


Fig. 1. The architecture of a general-purpose PIM [16].

The **Spatio-temporal scheduling** provides a dynamic scheduling algorithm that allocates a set of highly utilizable PIM PEs for efficient spatial sharing and evenly distributes requests across them to maximize temporal sharing. *SecPIMlet* identifies the point that maximize throughput and provisions PIM computation units accordingly. To further enhance temporal sharing efficiency, requests are spatially mapped to balance the load, and time slots are evenly assigned for temporal sharing, ensuring fairness across domains.

We evaluate *SecPIMlet* across six workload scenarios, including small language models (SLMs), recommendation models, and general PrIM workloads [26], [80]–[83], [87], [91], [94]. In the case of SLMs, model parameters and key-value caches reside in PIM to eliminate data copying during temporal sharing and inter-request execution. For recommendation models and other workloads, data is copied into PIM memory before execution and written back to the host memory afterward. Our results show that neither spatial nor temporal sharing can fully utilize PIM resources. Compared to *VirtualPIM* [48], which solely supports spatial sharing, *SecPIMlet* achieves a 175.4% throughput improvement by enabling spatio-temporal sharing while also providing security.

II. Background

A. Processing-in-memory (PIM)

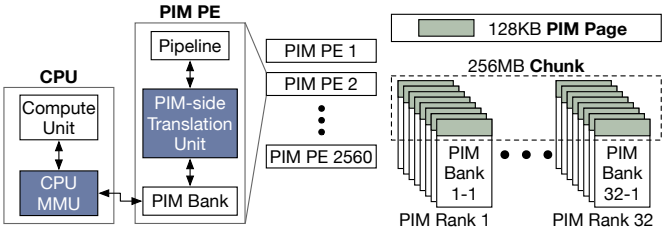
Processing-in-Memory (PIM) mitigates data movement bottlenecks by integrating compute units within memory. PIM architectures are classified into two categories: general-purpose PIM [16] and domain-specific PIM [33], [49]–[51], [54], [56]. While a domain-specific PIM relies on centralized host-side commands, a general-purpose PIM decodes programs within each PIM chip, offering superior flexibility and programmability. Given these advantages, many recent studies have adopted a general-purpose PIM [22]–[25], [27], [28], [30], [46], [52], [55], [60], [69]–[71], [76], [79]. Following this trend, *SecPIMlet* is also built upon a general-purpose PIM. As discussed in Section VI-C, it can be further extended to support domain-specific and future hybrid PIMs.

The general-purpose PIM system consists of two distinct memory types: the regular CPU-side memory without PIM computation, and the PIM-enabled memory (shown in Figure 1) that integrates the memory bank with processing elements (PEs). Each PIM rank comprises eight PIM chips, and each chip contains eight PIM PEs, resulting in a total of 64 PIM PEs per rank. Each PE is associated with a dedicated

TABLE I
COMPARISONS WITH THE PRIOR STUDIES

Study	Sharing		Virtualization		H/W Security	
	Spatial	Temporal	Inter PE	Intra PE	Access Control	Secure Channel
VirtualPIM [48]	✓	▲	✓	✗	✗	✗
UM-PIM [93]	✗	✓*	✗	✓	✗	✗
FACIL [77]	✗	✓*	✗	✓	✗	✗
SE-PIM [18]	✗	✗	✗	✗	✗	▲
***SecPIMlet* (Ours)**	✓	✓	✓	✓	✓	✓

(✓*: Theoretically possible, but was not explored.)
(▲: Possible with **extremely high overhead**.)



(a) Translation units of CPU and PIM PE. (b) Chunk/Page-granular translation.

Fig. 2. Virtual address translation of UM-PIM [93].

64 MB memory bank. The codes and data stored in PIM bank are loaded into three scratchpad memory regions within each PIM PE for execution: atomic memory (atomic RAM), instruction memory (IRAM), and working memory (WRAM). Basically, all memory accesses within the PIM subsystem utilize device physical addresses. Data communication between CPU-side regular memory and PIM-enabled memory is facilitated via AVX-512 load/store instructions with a peak transfer bandwidth of up to 19.2 GB/s per rank [26].

B. PIM Resource Sharing

As listed in Table I, PIM resources can be partitioned along two dimensions: spatial and temporal. Spatial sharing allows multiple workloads to be assigned to disjoint subsets of PIM PEs. Temporal sharing, on the other hand, enables multiple workloads to time-share the same PIM PE.

***VirtualPIM*:** *VirtualPIM* proposes a spatial sharing mechanism with an inefficient copy-based temporal sharing procedure [48]. However, since *VirtualPIM* does not support intra-PE virtualization and hardware security, data cannot securely remain in a PIM bank. In contrast, our design securely retains residual data within PIM bank using PIM confidential computing. Moreover, our system determines the optimal number of PIM PEs by evaluating performance across different number of PIM PEs, while *VirtualPIM* requires users to manually specify the number of PIM PEs.

***UM-PIM & FACIL*:** Although *UM-PIM* and *FACIL* could, in principle, be extended to support temporal sharing, their primary focus lies in enabling a unified memory space between the CPU and PIM with only intra-PE virtualization [77], [93]. These studies propose chunk-based memory virtualization mechanisms that leverage translation tables on both the CPU and PIM sides (Figure 2 (a)). Translation is performed at the granularity of a 256 MB chunk, with each PIM PE contributing 128 KB of its PIM bank to the virtualized memory space (Figure 2 (b)). While CPUs support memory interleaving across multiple banks, a PIM unit is restricted to accessing a single local bank. Consequently, two distinct data layouts may coexist within the system. To address this mismatch, these studies introduce data re-layout mechanism.

However, these approaches are only limited to temporal sharing in multi-domain environments since the translation unit encompasses the entire PIM bank. It leaves a critical aspect of PIM spatial resource management unaddressed. Moreover, these approaches remain vulnerable to malicious system software and physical attacks, as they address only one dimension of security (isolation, specifically virtualization) without ensuring hardware-enforced protection. In contrast, our design supports both temporal and spatial sharing with robust security guarantees via PIM confidential computing.

C. Hardware-enforced Confidential Computing

Many hardware vendors provide confidential computing solutions to support hardware-enforced security [6], [7], [29], [42], [43], [47], [72], [73]. Among them, Arm Confidential Compute Architecture (CCA) is the recent proposal from Arm that will be deployed onto next generation servers and user devices. This work is designed based on CCA, but the proposed mechanism is not limited to CCA and can be extended to Intel or AMD architectures [42], [43], [47].

Access control: Access control restricts programs from accessing parts of the system that it does not have access to, and this provides both confidentiality and integrity. To implement access control, CCA introduces the concept of a realm virtual machine (realm VM), a protected VM based on the two-stage address translation mechanism [7]. Stage 1 translation maps virtual addresses to intermediate physical addresses and is managed by the untrusted OS at exception level 1. Stage 2 translation maps intermediate physical addresses to physical addresses and is securely handled by the realm management monitor (RMM) and trusted hardware at exception level 2. By securely managing the Stage 2 translation tables, CCA enforces isolation across realm VMs. The RMM is a firmware-like verifiable software validated by hardware vendors, and the RMM is part of trusted computing base (TCB).

Secure CPU-PIM communication channels: Hardware attacks can leak data transmitted across unsecured channels such as the memory channel. Encryption and message authentication can be used to provide confidentiality and integrity on the channel. As shown in Table I, *SE-PIM* primarily focuses on establishing per-PE secure CPU-PIM communication channels [18]. However, *SE-PIM* incurs substantial overhead from its per-PE encryption and integrity engines, as it does not leverage the hardware packaging of PIM chips. In contrast, our design proposes chip-level encryption and integrity engines for lightweight hardware cost.

TABLE II
REAL-WORLD SCENARIOS

Personalized Mobile Assistant [4]	**(Mobile)** 8-24 GB memory with 128-384 PEs	
	GEMMA, TLAMMA	**(Fore)** Language Assistant
	GEMV, MLP, BS, SEL	**(Back)** Target Application Select. & Recomm.
Autonomous Vehicle [15], [32], [64]	**(Orin/Xavier)** 8-64 GB memory with 128-1024 PEs	
	GEMMA, TLAMMA	Talk to Drive
	NW, RED, UNI	Sensing
	BFS, GEMV, TS	Localization
	MLP, GEMV, SCAN	Object Detection
Edge/Fog Computing [85], [97]	**(Edge Node)** 1-64 GB memory with 16-1024 PEs	
	GEMMA, TLAMMA	Multiple AI Agents
	MLP, GEMV, SCAN, BS	Face Detection
	BFS, MLP, SCAN, TS	Location-based Game

III. MOTIVATION

A. Target Environment: Client-side AI System

In this study, we mainly target a client-side AI system equipped with general-purpose PIMs. The client-side AI system is the mainstream of PIM usage in both industry [49], [50], [74] and academia [62], [77].

Real-world scenarios: As summarized in Table II, client-side AI systems encompass a wide spectrum from personal devices to autonomous driving systems and edge/fog servers, requiring a multi-domain setting. Personalized mobile assistants [4] require the simultaneous execution of foreground language models alongside background engines for personalized task selection and recommendation. Similarly, autonomous driving platforms [15], [32], [64] orchestrate a wide array of concurrent workloads. These range from conversational *Talk to Drive* agents to mission-critical sensing, localization, and object detection, each requiring strict isolation. Furthermore, edge/fog computing nodes [85], [97] host multiple independent AI agents and diverse services including face detection, necessitating robust hardware-enforced isolation. All of these cases require multi-domain support.

Note that, the scope of a multi-domain extends beyond conventional multi-tenancy, encompassing multi-application and even multi-tasking within a single application. While multi-tenancy focuses on isolating disparate users, a multi-domain is essential for modern workloads where a single application consolidates multiple mutually distrustful sub-tasks. For instance, AI-agent system requires security boundaries for each agents and a trip-planning application integrates various third-party plugins to handle specific services such as flight bookings or hotel reservations. In such scenario, establishing multiple security domains and enforcing isolation between these domains is necessary.

To handle a client-side AI system, we design a system equipped with 16 GB of PIM banks with 256 PIM PEs and 8 GB of CPU-side regular memory. As shown in Table V, we construct a multi-domain setting that includes language models, recommendation models, and general-purpose workloads.

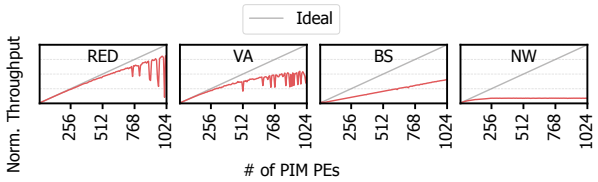


Fig. 3. Normalized throughput per number of PIM PEs. (Normalized to eight PIM PEs)

B. Threat Model

We adopt a general threat model widely used in both academic [1], [31], [37], [44], [57], [68], [84], [90] and industrial [14], [29], [42], [43], [47], [72], [73] studies on trusted execution environments (TEEs). We assume a strong attacker model in which attackers have full control over system software (both OSes and hypervisors), as well as physical accesses to the system. Furthermore, attackers may perform physical attacks on the exposed channels between the CPU and PIM/DRAM chips to intercept in-transit data. However, we assume that circuits inside packaged chips are physically protected, as assumed by prior work [2], [8], [11], [19], [29], [72], [73]. The trusted computing base (TCB) in this study comprises packaged hardware components (CPU and PIM/DRAM chips), and the verified firmware called realm management monitor (RMM). Rowhammer, side-channel, and availability attacks are beyond the scope of this paper.

C. The Need for Spatio-temporally Shared Multi-domain PIM

Observation 1: Two key opportunities motivate the need for spatio-temporal sharing and security on PIM: (1) Overcoming the PIM resource scaling bottlenecks in specific workloads, and (2) leveraging the under-utilized memory capacity to enable temporal sharing.

PIM resource scaling limitations of specific workloads (Spatial sharing opportunity): PIM workloads exhibit distinct resource scaling characteristics, making it both difficult and essential to determine the appropriate number of PIM PEs for efficient execution. We analyze the strong scaling performance (workload size is fixed) of the PrIM benchmark as the number of PIM PEs increased from 8 to 256. The detailed specification of workloads is described in Table V. Figure 3 shows the normalized throughput of each workload. `Ideal` shows the ideal throughput which increases proportionally with the number of PIM PEs. In general, the slope of the throughput decreases as the number of PIM PEs increases. Workloads such as `RED` scale well, while `VA` and `BS` show a lower slope of scaling, and `NW` shows low scalability. As shown in the scaling column of Table III, `TS`, `HST-L`, `GEMV`, `SEL`, and `MLP` exhibit high scaling behavior similar to `RED`. Furthermore, moderate, low, and minimum scaling workloads follow the behaviors of `VA`, `BS`, and `NW`, respectively. This trend shows that running multiple workloads sequentially across all PIM PEs leads to inefficient PIM resource usage.

TABLE III
SCALING AND MEMORY USAGE BEHAVIOR OF WORKLOADS

	Scaling ($\frac{Perf_{maxPE}}{Ideal}$)	Workload	Capacity	1 PE (64 MB)	1 PIM-chip (8 PEs, 512 MB)	Full-system (256 PEs, 16 GB)
PrIM	High (75,100]%	TS	16 MB	25%	3.13%	0.1%
		HST-L	48 MB	75%	9.38%	0.29%
		GEMV	328 MB	512.5%	64.06%	2%
		RED	398 MB	621.88%	77.73%	2.43%
		SEL	244 MB	381.25%	47.66%	1.49%
		MLP	328 MB	512.5%	64.06%	2%
	Moderate (50,75]%	UNI	244 MB	381.25%	47.66%	1.49%
		SCAN-SSA	244 MB	381.25%	47.66%	1.49%
		VA	80 MB	125%	15.63%	0.49%
		HST-S	48 MB	75%	9.38%	0.29%
	Low (25,50]%	SCAN-RSS	244 MB	381.25%	47.66%	1.49%
		BS	32 MB	50%	6.26%	0.2%
	Minimum (0,25]%	NW	512 MB	800%	100%	3.13%
		BFS	22 MB	34.38%	4.3%	0.13%
Recommendation Model		MOVIE	0.48 MB	0.75%	0.09%	0.003%
		BOOK	1.22 MB	1.91%	0.24%	0.01%
Language Model		GEMMA	2.3 GB	3680%	460%	14.38%
		TLLAMA	1.2 GB	1920%	240%	7.5%

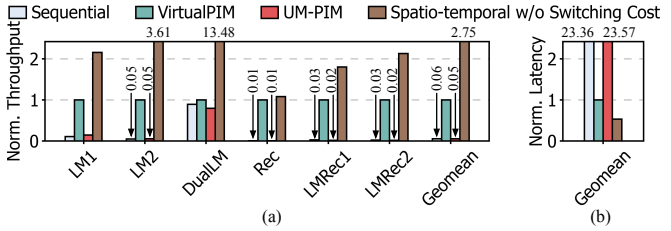


Fig. 4. (a) Normalized throughput and (b) normalized latency of *VirtualPIM*, *UM-PIM*, and the ideal spatio-temporal approach with no context switching overhead. (Normalized to *VirtualPIM*)

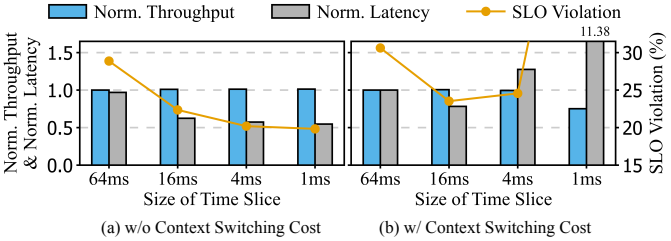


Fig. 5. Normalized throughput, normalized latency, and SLO violation according to the size of time slice, with and without context switching costs. (Normalized to 64 ms with context switching cost)

Instead, running workloads concurrently through spatial sharing can provide higher system throughput. The challenge is determining how many PIM PEs to allocate for each workload. **Memory capacity under-utilization (Temporal sharing opportunity):** We observe that most workloads in a general-purpose PIM system do not fully utilize the available PIM bank capacity. Table III presents the proportion of used memory regions. When allocating workloads to a chip, every workload of PrIM (TS to BFS) uses 100% or less of the PIM capacity. In contrast, language models (GEMMA and TLLAMA) require the full system configuration with 256 PIM PEs. This observation opens up two possibilities. First, for the workloads that consume very little memory, we can allocate these workloads to small spatial partitions of PIM PEs, while the larger workloads (language models) can utilize all the PIM PEs of the system. Second, there is enough free space in the PIM memory to hold the data of multiple domains, motivating the need for PIM PEs to store the data of multiple domains and providing memory isolation for data of different domains.

D. Shortcomings of Prior PIM Virtualizations

Observation 2: Spatio-temporal sharing enabled by both intra- and inter-PE virtualization offers substantial performance potential, but no prior work supports such sharing.

Figure 4 (a) shows the throughput of *VirtualPIM* [48], *UM-PIM* [93], and the ideal spatio-temporal sharing system that does not incur any context switching overhead (*Spatio-temporal w/o Switching Cost*) relative to *VirtualPIM*. The results indicate that *VirtualPIM* and *UM-PIM* achieve only 36.3% and 1.96% of *Spatio-temporal w/o Switching Cost*. This limited throughput stems from their constrained support for spatio-temporal sharing and data residuality. First, *VirtualPIM* supports inter-PE virtualization but is unable to securely retain residual data of other domains within the PIM bank, forcing residual data migration to the host memory. Second, *UM-PIM* employs an intra-PE virtualization scheme that groups identical offset memory regions across all PIM PEs into a single translation unit, enabling temporal sharing, but restricting spatial sharing across PIM PEs. In addition, Figure 4 (b) presents the normalized latency of *VirtualPIM*, *UM-PIM*, and *Spatio-temporal w/o Switching Cost* compared to the *VirtualPIM*. Given that the ideal latency is only 53.3% and 2.3% than those of *VirtualPIM* and *UM-PIM*, spatio-temporal sharing is essential for latency-sensitive workloads.

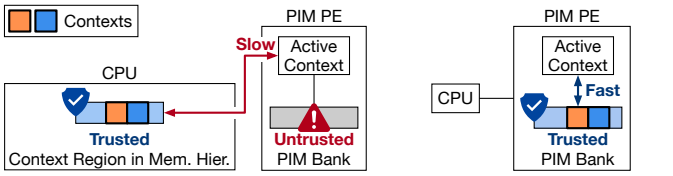
E. Current Limitations of Fine-grained Temporal Sharing

Observation 3: Fine-grained temporal sharing is required for low latency and reduced service level objective (SLO) violation rates[1]. However, it incurs frequent context switching, requiring low-cost context switching.

Observation 4: Due to the lack of hardware security in PIM chips, a naïve design stores domain contexts in the host memory, imposing a severe context switching penalty driven by low-bandwidth CPU-PIM context transfers.

Figure 5 (a) presents the performance with a zero-cost context switching as the time-slice size varies. Relative to the largest time slice (64 ms), reducing the time slice (1 ms) reduces latency and SLO violation rate by 43.5% and 31.3%,

[1]In this work, we define an SLO violation when a workload exceeds 3× of the single-execution latency.



(a) Naive CPU-centric. (b) *SecPIMlet*: PIM-offloaded.

Fig. 6. Comparison of context switching: (a) Naïve CPU-centric approach and (b) hardware-enforced PIM-offloaded approach in *SecPIMlet*.

TABLE IV
PER-PIM PE ENCRYPTION ENGINE COST (*SE-PIM* [18])

Area	Throughput	Memory Capacity		
		UPMEM	*HBM-PIM*	*GDDR6-AiM*
11.38 mm^2	1.28 TB/s per rank	-22%	-15%	-15%

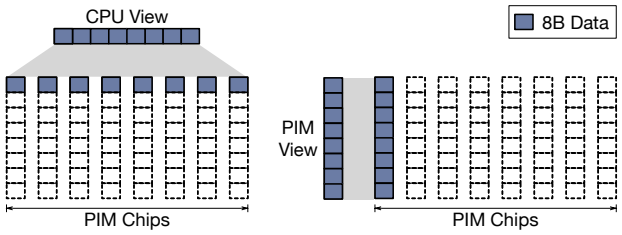
while maintaining similar throughput. However, frequent context switching under fine-grained temporal sharing can lead to significant performance degradation. As shown in Figure 5 (b), the overhead of context switching incurred at a 1 ms time slice reduces throughput by 25.8%, increases latency by 11.4×, and raises the SLO violation ratio by 3.4× compared to the configuration without the context switching cost.

As shown in Figure 6 (a), the main reason of context switching is the CPU-PIM communication overhead as contexts are stored in the CPU-side host memory. To mitigate this overhead, *SecPIMlet* supports hardware-enforce security solution for PIM chips and allows keeping the context within the PIM bank. Leveraging the trusted PIM bank, Figure 6 (b) describes the hardware-enforced PIM-offloaded context switching mechanism in *SecPIMlet*. It enables the use of the high internal PIM bandwidth instead of the relatively low external CPU-PIM bandwidth.

F. Architectural Constraints for Secure CPU-PIM Channel

Observation 5: Encryption and authentication logic can ensure confidentiality and integrity of the CPU-PIM channels, but incurs high area overhead on area-sensitive PIM chips and complicates the CPU/PIM data layouts.

Area constraints: As shown in Table IV, the per-PE encryption scheme in *SE-PIM* requires significant hardware cost [18]. *SE-PIM* adopts PE-level AES-GCM engines to sustain the internal PIM bandwidth of 1.28 TB/s per rank. Table IV shows the hardware overhead of *SE-PIM*. Given eight PIM chips per rank, each chip must support 0.16 TB/s encryption throughput. Achieving 160 GB/s requires approximately 2,655 lightweight AES-GCM logic with 494 Mbps (61.75 MB/s) [92], resulting in a total area of about 11.38 mm^2 per PIM chip. Such a large area cost in *SE-PIM* significantly reduces the effective memory capacity. The 11.38 mm^2 AES-GCM overhead is about 11% of a typical 8 mm × 13 mm PIM chip. This results in an additional 22% memory capacity reduction for UPMEM [16]



(a) CPU: Interleaved view (b) PIM: Continuous view

Fig. 7. Two discrete data layouts: (a) CPU-side interleaved view and (b) PIM-side continuous view.

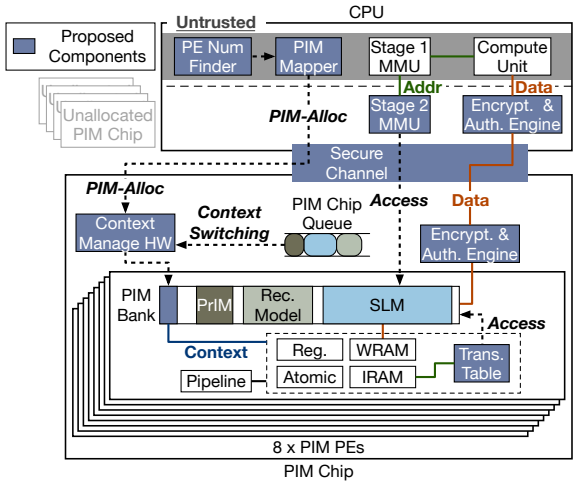


Fig. 8. *SecPIMlet* architecture overview.

and about 15% for both Samsung HBM-PIM [56] and SK Hynix GDDR6-AiM [33], [51], [54].

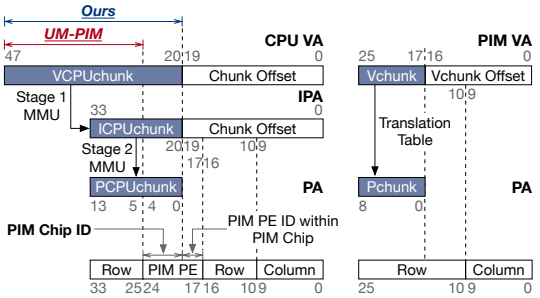
Disparate data layout: The design of the encryption engine should account for the underlying data layout. Modern memory systems exploit high parallelism through memory interleaving, which distributes consecutive data blocks across multiple chips on the rank. In contrast, PIM devices access data in a strictly contiguous manner because they are restricted to their local PIM banks. As described in Figure 7 (a), consecutive data blocks in the CPU-side layout are interleaved across multiple PIM chips. Consequently, a PIM device can only observe a single fragment of the data. On the other hand, Figure 7 (b) shows that consecutive data blocks in the PIM-side layout are placed within a single PIM chip, leading the CPU to see only a single block at a time. These divergent layouts yield different view on CPU and PIM devices, which introduces inconsistencies in the encryption results.

IV. SECPIMLET: SECURE & FAST MULTI-DOMAIN PIM

A. Overview

Figure 8 shows the overview of *SecPIMlet*. To support secure and fast multi-domain PIM, it integrates PIM confidential computing with spatio-temporal scheduling.

PIM confidential computing: *SecPIMlet* extends confidential computing to PIM by three approaches: 1) chip-granular address translation, 2) hardware-enforced access control with



(a) CPU-side translation. (b) PIM-side translation.

Fig. 9. Chip-granular translation.

PIM-offloaded context switching, and 3) per-chip and layout-aware secure CPU-PIM channel.

Spatio-temporal scheduling: *SecPIMlet* enables safe spatio-temporal sharing, where the chip-granular translation enables flexible spatial sharing, and temporal sharing of PIM PEs is made possible by means of hardware-enforced access control and context management hardware. Given the possibility of supporting spatio-temporal sharing, we propose an approach to find the optimal number of PIM PEs for an incoming workload (*PE num finder*), and a *PIM mapper* to schedule workload onto the requested number of PIM PEs.

PIM-side trusted hardware initialization: To securely initialize a PIM-side context management hardware, the system verifies the chain of trust encompassing secure boot, remote attestation, and session key exchange. First, during the secure boot phase, the context management hardware scans the initial firmware and hardware configuration states, encapsulating this booting report with a unique, manufacturer-embedded hardware-rooted private key. Second, for attestation, this cryptographic booting report is verified by trusted third-party attestation services. Lastly, upon successful attestation, the CPU-side trusted execution environment (TEE) and the PIM-side context management hardware share session (channel) keys by executing a Diffie-Hellman key exchange protocol [41].

B. Chip-granular Address Translation

To isolate spatio-temporal shared PIM data, *SecPIMlet* employs chip-granular address translation for both inter- and intra-PE virtualization as shown in Figure 9. Chip-granular translation constructs 1 MB translation units by grouping 128 KB frames from eight PEs within the same PIM chip. Unlike the temporal-only translation (*UM-PIM*), this scheme offers a higher degree of flexibility in address mapping by allowing all translations in a PIM chip granularity.

CPU-side address translation: For a memory access from the CPU to PIM banks, a 48-bit virtual address (VA) is first translated into an intermediate physical address (IPA) with a 1 MB chunk granularity, the upper 28 bits of the VA representing the virtual CPU-side chunk ID (VCPUchunk), which is translated into a 14 bit intermediate CPU-side chunk ID (ICPUchunk). Subsequently, the IPA is translated into a physical address (PA) and the corresponding physical CPU-side chunk ID (PCPUchunk) by the Stage 2 MMU, which is securely managed by the trusted firmware such as realm management monitor (RMM). The PCPUchunk is concatenated with the chunk offset to produce the final physical address which maps to a specific PIM PE, row and column.

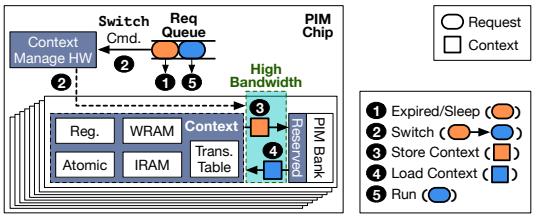


Fig. 10. Hardware-enforced PIM-offloaded context switching.

PIM-side address translation: Similar to the temporal-only translation (*UM-PIM*), the PIM-side translation constructs 128 KB frames. These frames are identified by virtual chunk IDs (Vchunks) and mapped to physical chunk IDs (Pchunks) by the PIM-side translation table.

Size of translation table: While chip-granular translation expands the CPU-side table, which remains manageable via a multi-level structure, the size of PIM-side tables stays constant.

C. Hardware-enforced Access Control with PIM-offloaded Context Switching

To mitigate threats posed by malicious system software (i.e. OSes or hypervisors), *SecPIMlet* introduces a hardware-enforced access control for PIM banks ensuring strict data isolation. This mechanism regulates PIM PE-initiated requests and enables trusted spatio-temporal sharing, accompanied by a secure context switching procedure. As shown in Figure 10, *SecPIMlet* places dedicated hardware, called the *context management hardware*, on each PIM chip. The PIM-side translation table is solely updated by the this hardware during PIM allocation and context switching procedure.

PIM Bank-residual PIM context: *SecPIMlet* defines a PIM context as the collection of architectural states, including the translation table, intermediate buffers (WRAM, IRAM, and atomic RAM), and all PIM registers. The total context size is approximately 88.7KB, comprising of 576B translation table, 64KB WRAM, 24KB IRAM, 32B atomic RAM, and 96B of PIM registers. By switching contexts, the system can efficiently suspend and resume PIM tasks without losing execution state. To support this mechanism, *SecPIMlet* introduces dedicated context-management hardware that performs context initialization and context switching.

Unlike naïve designs that store PIM context in CPU-side memory, *SecPIMlet* stores contexts within a 709.6KB reserved region in each bank, accounting for only 1.08% of the total PIM bank capacity. This reserved region is securely protected, as it is accessible exclusively by the context management hardware and inaccessible via translation tables. The region is statically partitioned into eight context slots, where each unit contains the translation table, WRAM, IRAM, atomic RAM, and register states, and is indexed by a 4B kernel ID.

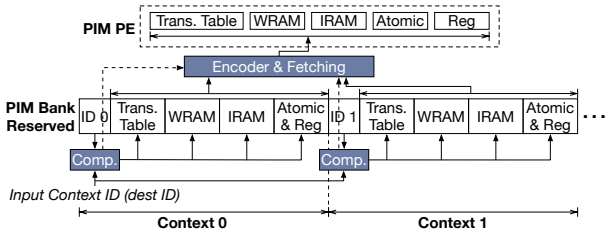
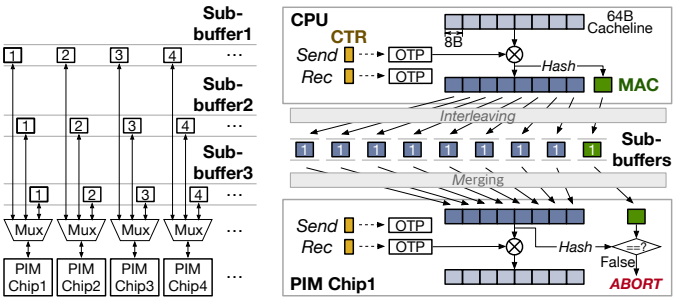


Fig. 11. Context management hardware for PIM-offloaded context switching.



(a) Sub-buffer. (b) Layout-aware data protection.

Fig. 12. Layout-aware encryption and integrity protection.

Hardware-enforced PIM-offloaded context switching: Figure 10 presents our hardware-enforced PIM-offloaded context switching procedure based on the bank-residual PIM context. ❶ When a time slice expires or the previously executing kernel completes, ❷ the PIM chip queue issues a `Switch(srcID, destID)` instruction to the context management hardware. Upon receiving this instruction, ❸ the hardware first writes back the current execution context to the reserved region of the PIM bank, and then ❹ loads the next context into the on-chip active context regions (i.e. registers, atomic RAM, WRAM, IRAM, and translation table). Finally, after the new context is loaded, ❺ the execution is started. Figure 11 shows the detailed operation of the hardware-assisted context loading mechanism. The context management hardware searches the target context by comparing the stored context IDs with the `destID` parameter of the `Switch(srcID, destID)` instruction. An 8-to-3 encoder subsequently selects the context offset associated with `destID`, and the selected context is loaded to prepare the next execution.

PIM allocation: `PIMAlloc(kerID)` is issued by the PIM mapper once PIM scheduling is completed, and is securely executed by the context management hardware to assign a new context. After the target PIM chip is selected by the PIM mapper, the mapper delivers the `PIMAlloc(kerID)` instruction to the context management hardware within the corresponding PIM chip. The context management hardware then selects an empty context entry within the reserved region of the PIM bank and statically creates a translation table that maps the entire bank region, rather than generating entries in an on-demand manner. After the translation table setup is completed, the table is transferred through a secure channel to the Realm Management Monitor (RMM) for synchronizing between the CPU-side Stage 2 table and the PIM-side translation table.

D. Per-chip and Layout-aware Secure CPU-PIM Channel

Lastly, to protect attacks during CPU-PIM communication, *SecPIMlet* establishes a secure communication channel between the CPU and PIM. Rather than augmenting each individual PE, the system equips each PIM chip with dedicated encryption and integrity protection engines that accommodate both CPU and PIM data layouts.

Per PIM-chip data protection: Compared to *SE-PIM* [18] which deploys engines for all PIM PEs, *SecPIMlet* adopts per-chip engines, with one engine per PIM chip (8 PIM PEs), leveraging the protection offered by the hardware packaging of the PIM chip. Since plaintext data within a PIM chip is inherently secure due to hardware packaging, *SecPIMlet* only requires a CPU-PIM bandwidth-controlled encryption engine, whereas *SE-PIM* demands significantly higher internal PIM bandwidth to encrypt data movement between the PIM PE and the PIM banks. This design choice substantially reduces hardware overhead of the encryption and integrity engine.

Layout-aware data protection: The disparate CPU and PIM data layouts need transposing (or aligning), and this is done via sub-buffers. The cachelines that would be interleaved across different banks need to instead be routed to one PIM chip, as also illustrated in Figure 7. Our design is based on the sub-buffer approach of *UM-PIM*, but we add one more sub-buffer to transfer the message authentication code (MAC) in addition to the data. Figure 12 (a) describes the sub-buffer architecture. The i-th buffer entry is mapped to PIM chip i, and each PIM chip employs a multiplexer to select a value from multiple sub-buffers. This organization enables layout-aware data protection. Figure 12 (b) presents the procedure for layout-aware protection. The CPU-side encryption engine generates a one-time pad (OTP) from the send counter and encrypts plaintext via a single-cycle XOR. Since OTP generation is data-independent, OTP values can be pre-computed and stored in the encryption engine (called OTP buffer). After the encryption, the MAC is computed by hashing the ciphertext.

The encrypted ciphertext and MAC are subsequently written to the sub-buffers in an interleaved manner. After one burst length, ciphertext and MAC are distributed to all i-th entries of the sub-buffers. During the following nine burst lengths, the ciphertext and MAC are sequentially transferred from the sub-buffers to the PIM chip i, while the $i+1$-th entries of the sub-buffer is being received. Once a PIM chip receives the ciphertext and MAC, the PIM-side encryption engine decrypts the ciphertext using the OTP generated from the receive counter. Additionally, the received MAC is compared against the MAC computed from the ciphertext to verify data integrity. If the computed and received MAC values do not match, the execution is aborted.

Counter-based encryption: *SecPIMlet* uses a synchronized counter mechanism in which the counters on both sides are incremented by one for each communication event. In this design, each CPU-PIM channel maintains one send counter

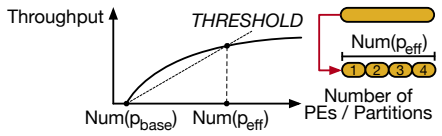


Fig. 13. The number of PIM PEs (or partitions) for efficient execution. We use *THRESHOLD* × *Ideal Scaling* as criteria to select the number of PEs.

and one receive counter at both the CPU side and the PIM side. These per-channel counters are then used to generate OTPs, which are stored in a dedicated OTP buffer. Consequently, each PIM chip requires only two counters and two OTP buffers, whereas the CPU maintains 64 counters and 64 OTP buffers (5.01 KB). Since *SecPIMlet* utilizes 8B counters, achieving a counter overflow requires more than 30 years of continuous communication under maximum bus utilization. In the event of a counter overflow, the system generates and shares a new communication session key via the Diffie-Hellman protocol [41]. Furthermore, since a new session key is established upon every system reboot, counter reuse across boot cycles cannot compromise the security guarantees.

Key management: Both CPU and PIM encryption engines have their own root key provisioned by manufacturers. The CPU engine generates a communication key based on the root key and securely shares the key with the allocated PIM chips via the Diffie-Hellman protocol [41]. With CPU memory encryption, when copying data from the CPU memory onto the PIM memory, the encrypted data stored in CPU memory is first decrypted by the CPU memory encryption key. Next, the decrypted data is re-encrypted using the communication key shared between the CPU and the PIM chip. Upon arrival at the PIM chip, the data is decrypted using the shared key and stored in plaintext within the PIM chip.

E. Spatio-temporal Scheduling

We propose spatio-temporal scheduling, which combines spatial scheduling based on the adequate number of PIM PEs and temporal scheduling based on time slices while satisfying memory capacity constraints. To enable this, three key challenges should be addressed: (1) determining the number of PIM PEs to allocate, (2) mapping each divided execution unit to an appropriate PIM PE, and (3) selecting the residual data within PIM banks for future use.

Finding the number of allocated PIM PEs (*PE num finder*): The number of PIM PEs to allocate is determined as the maximum PEs that exceed 60% of the `Ideal` performance for that number of PEs. The 60% *THRESHOLD* was heuristically selected based on the evaluations in Section V-B). As shown in Figure 3 and 13, performance improvement saturates as the number of allocated PIM PEs increases. Thus, the intersection point of the performance curve and the threshold line defines the number of PIM PEs for efficient execution. Since we allocate PIM PEs by a PIM chip (8 PIM PEs) granularity, the number of efficient PIM PEs should be a multiple of eight. We define per-PIM chip execution units as *partitions*.

Algorithm 1: Spatio-temporal Scheduling (*PIM Mapper*)

```
1  SpatioTemporalMapper (Req_new, ChipQs):
2  Num(p_eff) ← PENumFinder (Req_new)
3  for i = 1 to Num(p_eff) do
4      ChipQs ← AscendSort (ChipQs, key=# of partitions)
5      for ChipQ ← ChipQs do
6          Mem_ChipQ ← Current memory usage of ChipQ
7          if MemUsage (p_i) + Mem_ChipQ ≤ MemLimit (Chip)
             then
8              Alloc partition p_i to ChipQ
9              break
10         end
11     end
12 end
```

Determining executed PIM PEs (*PIM mapper*): Since the allocation unit is a PIM chip, each PIM chip maintains its own allocation queue for partition scheduling. Algorithm 1 shows our spatio-temporal mapping policy. To balance the number of executing partitions, PIM chip queues are sorted by fewest allocated partitions. (*line 4*) Starting from the least-loaded PIM chip, the PIM mapper checks whether enough memory is available to accommodate the incoming partition. (*line 7*) If the capacity constraint is satisfied, the incoming partition is newly mapped to that PIM chip. Otherwise, the PIM mapper checks the next PIM chip queue in order. When no PIM chip can accommodate the partition, the incoming partition remains unallocated and waits until a feasible mapping becomes available. The allocation procedure is repeated until all partitions are allocated. Note that, to confirm current mapping, *PIM mapper* fetches the allocated 4B Kernel IDs from each PIM chip. Since each PIM chip manages a shallow queue depth of eight tasks, this procedure requires only 32B fetching per PIM chip. Furthermore, by leveraging this mapping information, the *PIM mapper* can easily compute the memory capacity usage.

Time slice-based temporal scheduling: *SecPIMlet* executes *partitions* on each PIM chip using a time slice-based temporal scheduling. We employ PIM chip queues, updated at every static time slice, to implement this policy. If a partition completes executing within the time-slice, it is removed from the queue. Otherwise, any partition with remaining work is re-queued for future scheduling. When the PIM chip schedules a different partition compared to the previous time slice, a context switch is issued to the context management hardware.

Application-aware data residuality: *SecPIMlet* retains small language model (SLM) data in PIM bank to accelerate subsequent requests upon the completion of each request. SLMs exhibit high data reuse, particularly for weight parameters and key–value (KV) cache entries accessed frequently across consecutive requests. To exploit this, *SecPIMlet* retains all SLM data permanently in PIM banks, minimizing data movement overhead and enabling efficient re-execution. In contrast, the other workloads feature dynamic input data, and the data of these workloads are securely copied back to the CPU-side host memory on completion. The residual data in PIM banks is safe due to the isolation provided by PIM-side virtualization and hardware-enforced access control in *SecPIMlet*.

TABLE V
EVALUATED PIM BENCHMARKS AND SCENARIOS

PrIM Benchmark [26]			
Benchmark	**Configuration**	**Benchmark**	**Configuration**
TS	4M elements, 256 queries	GEMV	M=20480, N=4096
HST-L	8×(1536×1024) inputs	HST-S	8×(1536×1024) inputs
MLP	3 layers, 20K neurons	VA	20M element
SCAN-RSS	30M elements	SCAN-SSA	30M elements
SEL	30M elements	UNI	30M elements
RED	50M elements	BS	2M queries
NW	8K bps	BFS	*loc-gowalla [10]*
Recommendation Model [82], [83], [94]			
MOVIE	MovieLens-1M, 3952 item, 64 embed-dim		
BOOK	Goodbooks-10K, 10000 item, 64 embed-dim		
Small Language Model [80], [81], [87], [91]			
GEMMA	Gemma-2B, 2K embed-dim, 8K seq-len, 8 head		
TLLAMA	TinyLlama-1.1B, 2K embed-dim, 2K seq-len, 32 head		

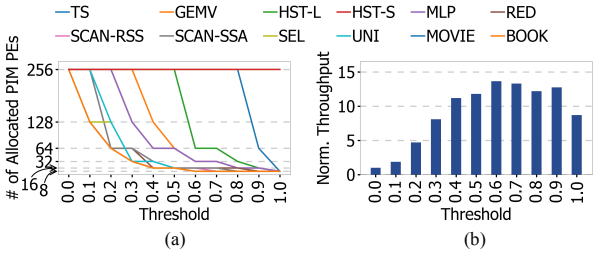
Multi-domain Scenario (PrIM: TS|GEMV|HST-L|HST-S|MLP|VA |SCAN-RSS|SCAN-SSA|SEL|UNI|RED|BS|NW|BFS) (REC: MOVIE|BOOK)			
Scen ID	**Domain1**	**Domain2**	**Domain3**
LM1	GEMMA	PrIM	PrIM
LM2	TLLAMA	PrIM	PrIM
DualLM	GEMMA	TLLAMA	PrIM
RecMajor	REC	PrIM	PrIM
LMRec1	GEMMA	REC	PrIM
LMRec2	TLLAMA	REC	PrIM

V. EVALUATION

A. Methodology

Evaluation infrastructure: Our PIM performance model is based on real UPMEM execution data to accurately represent execution times across different number of PIM PEs [16]. To determine the total latency, we add the overheads of virtualization, encryption, and context switching to the base execution time. Encryption and authentication costs are incorporated into the data copy phase using an 11-cycle encryption scheme with an 8B counter and an 8B MAC design [58]. Virtualization costs are calculated by adding a conservative 2-cycle translation penalty for PIM-side cost to every PIM bank read and write operation. Since the PIM-side translation table is capable of holding all address mappings, the translation penalty consists of only the latency of a single table lookup (9 bits per entry). Given that the native WRAM access bandwidth is 8 B/cycle [24], [40], our 2-cycle allocation for this lookup serves as a highly conservative cycle count. For CPU-side translation, we modeled with 4-level walk and 100 cycles per level [39]. In addition, our model also accounts for context switching. Context switching stalls execution to save and restore an 88.7KB context. Before execution, the request generator issues requests at a rate of d tokens or d requests per second, depending on the density (d; default $d = 2$).

Client-side AI system setup: As described in Table II, client-side AI systems generally integrate 24 GB memory. Following this configuration, *SecPIMlet* assumes an 8 GB CPU regular memory and a 16 GB PIM memory. It employs 256 PIM PEs by default, proportionally scaled to the memory capacity.



(a) (b)

Fig. 14. *THRESHOLD* analysis. (Normalized to *THRESHOLD*=0)

Benchmarks and scenarios: Table V presents our evaluated workloads and multi-domain scenarios. We adopt 14 general workloads from the PrIM benchmark [26] (proportionally scaled-down), two recommendation datasets [82], [83], [94], and two small language models (SLMs) [80], [81], [87], [91].

Based on these workloads, we configure four multi-domain scenarios for client-side AI systems. LM1 and LM2 represent configurations with two general workloads and one language workload. DualLM consists of two language workloads and one general workload, creating a more memory-intensive condition. For recommendation models, RecMajor consists of one recommendation workload and two general workloads. LMRec1 and LMRec2 each include one language, one recommendation, and one general workload.

Evaluated configurations: *Serialized* represents the naïve performance without applying both spatial and temporal sharing. *Temporal* applies temporal sharing, but the overhead of copying all data from the host regular memory to PIM banks causes a significant overhead during every context switching. *UM-PIM* represents temporal sharing without copying data [93], however, note that it does not provide inter-PE isolation across domain contexts. *VirtualPIM* represents spatial sharing without intra-PE isolation, and we use it as the baseline and normalize all results to it. *Spatio-Temporal* provides spatial sharing and does not copy data during temporal sharing. *+Switch Opt.* is based on the spatio-temporal sharing and differs in that all context switching during temporal sharing occurs entirely within the PIM without involving the CPU. Since overhead for the context switching is reduced, we set the reduced 1 ms time slice size from the default 4 ms time slice size for this mechanism. *+Addr Trans.* builds on *+Switch Opt.* and adds secure address translation when both the host CPU and PIM PEs access PIM banks. *Ours* extends the *+Addr Trans.* mechanism by adding encryption and integrity protection engines to enable secure CPU-PIM data movement.

Hyperparameters: As shown in Figure 5, we employ a 4 ms time slice for *Serialized*, *Temporal*, *UM-PIM*, and *VirtualPIM*, and a 1 ms time slice for the remaining schemes. This configuration reflects the substantial context switching overhead incurred at a 1 ms granularity prior to the introduction of hardware-enforced PIM-offloaded context switching (*+Switch Opt.*). Based on the analysis in Section V-B, we set *THRESHOLD* to 0.6 to determine the number of allocated PIM PEs.

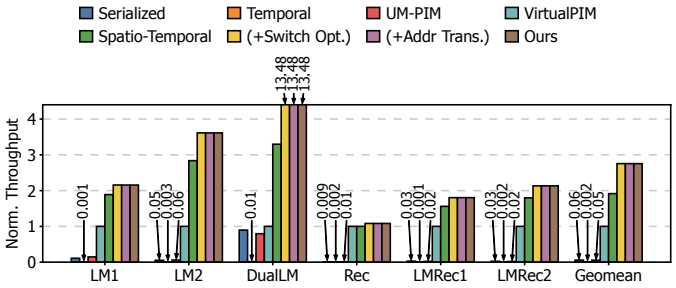


Fig. 15. Normalized throughput. (Normalized to *VirtualPIM*)

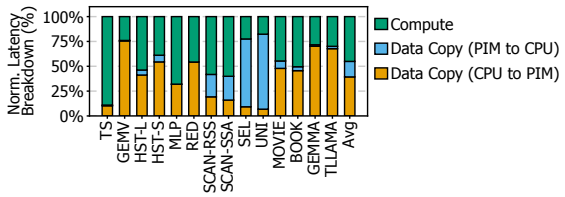


Fig. 16. The ratio of data copy.

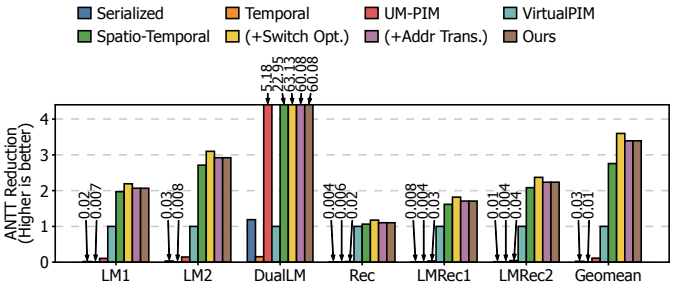


Fig. 17. Average normalized turnaround time (ANTT) reduction. (Normalized to *VirtualPIM*)

B. THRESHOLD Analysis

Figure 14 presents the number of efficient partitions to allocate for each workload and the corresponding throughput of *SecPIMlet* as the *THRESHOLD* value varies. As the *THRESHOLD* increases, the number of PIM PEs allocated to each workload decreases, directly impacting system performance. Meanwhile, low *THRESHOLD* leads to the over-allocation of PIM PEs, causing resource waste and reducing utilization. The throughput varies substantially, by up to 13.6x depending on the selected *THRESHOLD*. In this study, we determined the optimal *THRESHOLD* value to be 0.6, which yields the highest throughput, and used it for subsequent experiments.

C. Throughput Improvement

Figure 15 presents the performance improvements achieved in each scenario. *Serialized* shows, on average, a performance degradation by 94.3% compared to *VirtualPIM*. This result indicates that scalability deteriorates when a single workload monopolizes all PIM PEs and underscores the necessity of spatial sharing to improve resource utilization. *Temporal* demonstrates an average performance drop by 99.8% compared to *VirtualPIM*. This observation reveals the overheads of temporal sharing such as context switching and data copy. In *UM-PIM*, a performance degradation of 94.6% was observed compared to *VirtualPIM*. As shown in Figure 3, this is because *UM-PIM* distributes data across all PIM PEs, leading to degraded throughput for general or recommendation workloads. *VirtualPIM*, with spatial sharing, outperforms previous three mechanisms. *Spatio-Temporal* achieves performance gain by 91.6% compared to *VirtualPIM*. This is because, unlike *VirtualPIM*, which relies solely on spatial sharing, *Spatio-Temporal* provides temporal sharing without data copy and exploiting data retention within PIM banks.

+*Switch Opt.* outperforms *VirtualPIM* by 175.5% thanks to the context switching being performed in PIM instead of the CPU. +*Addr Trans.* outperforms VirtualPIM by 175.4%. Given that all PIM-side translation entries reside in translation tables and incur no misses, the introduced additional throughput overhead remains only 0.1%. Finally, *Ours* delivers a performance improvement of 175.4% over *VirtualPIM*. Although it incorporates both a secure address translation unit and a secure CPU-PIM communication engine, the performance overhead remains low compared to the optimal mechanism, +*Switch Opt.*. The performance overheads of our security mechanisms can be categorized into three primary sources. First, cryptographic operations are mostly limited to the initial loading and final write-back, restricting the MAC transfer overhead to a mere 1% latency increment. Second, OTP buffer stalls are entirely avoided because the security engine supports the full but low CPU-PIM bandwidth. Lastly, address translation overhead is minimized as the PIM translation tables contain every mapping of their respective PIM chip, and CPU-side translations are rare. *UM-PIM* and *Spatio-Temporal* achieve higher performance than *Temporal* and *VirtualPIM*, respectively. We can find the reason for this in Figure 16, where the latency of data copying dominates the computation latency for most workloads (average, 54.9%).

D. ANTT & TBT

Figure 17 presents the reduction of the average normalized turnaround time (ANTT) values compared to *VirtualPIM*. ANTT is a metric that normalizes the execution time of each workload, treating all workloads with equal weight. *Serialized* and *Temporal* achieve ANTT reductions by 0.03% and 0.01%, respectively, compared to *VirtualPIM*. *UM-PIM* results in ANTT reductions by 11.3% compared to *VirtualPIM*. However, in the case of `DualLM`, the ANTT reduction is improved by 5.18×, because *UM-PIM* can keep the data and contexts of both SLMs and the PrIM workloads residual within the PIM bank, thus reducing the cost of data copying. The ANTT reduction is increased by 2.75× and 3.60× for *Spatio-Temporal* and +*Switch Opt.*, but it regresses slightly with +*Addr Trans* and *Ours* to 3.39× and 3.36×.

Figure 18 shows the time-between-tokens (TBT) for language models. TBT reflects normalized decoding time, which suits *SecPIMlet* better than *VirtualPIM*. The `Rec` workload, which does not include a language model, is excluded from this analysis. *Serialized*, *Temporal*, and *UM-PIM* showed 21×,

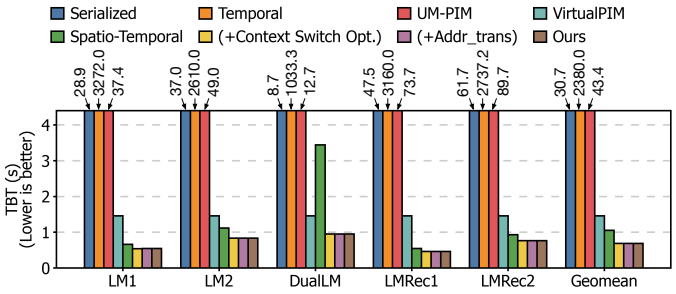


Fig. 18. Time between tokens (TBT).

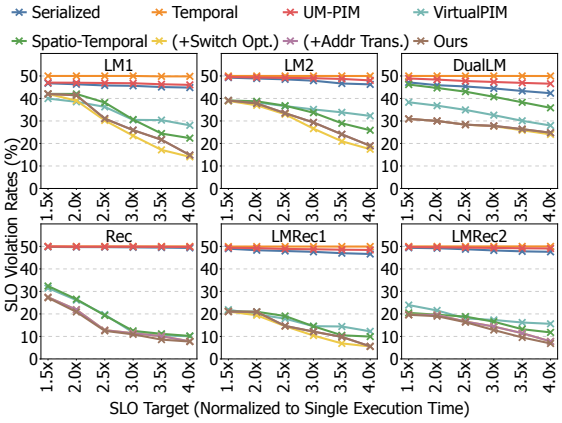


Fig. 19. SLO violation rates across various SLO targets (1.5× to 4.0× of single execution time).

1627.6×, and 29.7× higher TBT, respectively, compared to *VirtualPIM*. *Spatio-Temporal*, *+Context Switch Opt.*, *+Addr Trans.*, and *Ours* showed 27.9%, 52.9%, 52.9%, and 52.9% lower TBT respectively, compared to *VirtualPIM*. In the case of *spatio-temporal*, even when context switching occurs, the weights and KV caches and contexts of SLMs can be kept residually within PIM, which leads to lower TBT compared with *VirtualPIM*. With *+Context Switch Opt.*, the performance improves thanks to more efficient context switching within PIM, and we also observed that adding the address translation unit and secure CPU-PIM communication engine in *+Addr Trans.* and *Ours* did not meaningfully affect performance.

E. SLO Violation Rate Reduction

Figure 19 shows SLO violation rates under varying SLO targets from 1.5 times to 4 times of single execution time. *Serialized*, *Temporal*, and *UM-PIM* consistently exhibit prohibitively high violation rates exceeding 45%. However, we observed that *VirtualPIM*, *Spatio-Temporal*, *+Switch Opt.*, *+Addr Trans.*, and *Ours* show a substantial decrease in SLO violation rates as the SLO targets become more lenient. When the SLO target is set to 4 times of the single execution latency, *Spatio-Temporal* and *+Switch Opt.* exhibit SLO violation rate reductions of 8.1% and 39.3% on average, relative to *VirtualPIM*, respectively. *+Addr Trans.* and *Ours* show similar SLO violation rates to *+Switch Opt.*

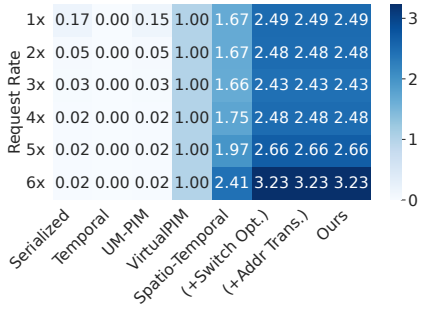


Fig. 20. Normalized throughput per request rate. (Normalized to *VirtualPIM*)

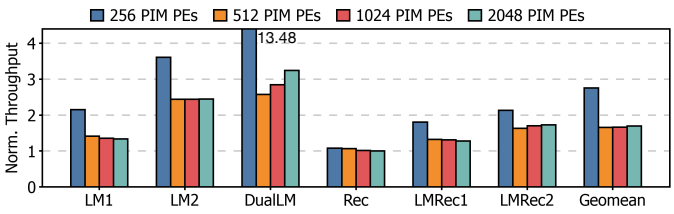


Fig. 21. Normalized throughput by scaling PIM PE numbers. (Normalized to each PIM PE number of *VirtualPIM*)

F. Scalability Study

Figure 20 presents the throughput under varying request rates from $d = 0.5$ requests or tokens per second ($d = 0.5$ is the baseline, denoted $1\times$). Darker shades in the figure correspond to higher throughput levels, showing that *Ours* achieves higher throughput across a wide range of request rates. *Serialized* and *UM-PIM* exhibit a sharp decline in performance as the request rate increases. This performance degradation is due to significant resource under-utilization. As the request rate increases, the number of concurrent requests grows, which amplifies the inefficiency of the baseline designs to fall further behind the offered load. Unlike *VirtualPIM*, our design achieves the same combination with significantly lower context switching overhead, improving resource utilization. As the request rate increases, this advantage becomes more pronounced. Consequently, when scaling the request rate by a factor of six, *Ours* achieves a remarkable 3.2× throughput advantage over *VirtualPIM*.

To reflect the various client-side AI systems (Table II) and scale to server-side AI systems, we scaled our experiments from the original 256 to 512, 1024, and 2048 PIM PEs, corresponding to 32GB and 128GB capacities. Figure 21 depicts the throughput of *Ours* normalized to the *VirtualPIM* baseline for each PIM PE count. Although the throughput improvement diminishes at 256 PIM PEs due to reduced opportunities for spatial and temporal sharing, our proposed method consistently outperforms *VirtualPIM* by 67%, 66%, and 69% at 512, 1024, and 2048 PIM PEs, respectively. Scalability improvements are reduced to large-scale configuration, as the memory capacity stalls of multiple requests are solved in large-scale *VirtualPIM*. However, as future workloads introduce higher memory capacity constraint, the performance benefits of our secure spatio-temporal sharing will scale accordingly.

G. Hardware Overhead

SecPIMlet incurs an area overhead of 3.23% on the PIM, and 0.44% (client-scale) and 2.37% (server-scale) on the CPU.

PIM hardware overhead: Under the target 20 nm DRAM process, the total combined PIM-side hardware overhead accumulates to 3.36 mm^2, occupying 3.23% of the 8 mm $\times$ 13 mm PIM chip. The PIM-side area overhead comprises of three primary components: the authenticated encryption engine, the context switching engine, and on-chip buffers. First, the authenticated encryption engine requires 348.99 kilo-gates (3.33 mm^2) to sustain a 50.49 Gbps throughput under a TSMC 180 nm process [61], [65]. By using a conservative linear scaling of 20 nm process projection and logic to DRAM process density penalty (10$\times$ for density, 3$\times$ for DRAM transistor speed), we estimate an area overhead of 3.33 mm^2 to handle the 2.4 GB/s PIM chip bandwidth. Second, the context switching engine, which consists of eight parallel 4B comparators (760 gates) and an 8-to-3 encoder (13 gates), requires 773 gates in total, incurring a 0.02 mm^2 area overhead. Lastly, analytical evaluations via CACTI [66] indicate that the on-chip buffers, comprising an 8B MAC sub-buffer and an 80B OTP buffer, introduce a 0.013 mm^2 area overhead.

CPU hardware overhead: Under the target 10 nm logic process, the total area overhead is 0.11 mm^2 per PIM chip, occupying 0.44% of 256-PE 400 mm^2 CPU chip and 2.37% of 2048-PE 600 mm^2 CPU chip, respectively. First, the CPU incorporates the authenticated encryption engine, and OTP tables. It is provisioned to scale with the total number of PIM chips. Fabricated on a 10 nm logic process, this infrastructure incurs an area overhead of 0.11 mm^2 per PIM chip. Second, the PIM mapping logic introduces near-zero hardware overhead since the scheduling mechanism (Algorithm 1) only requires a single summation and comparison. Furthermore, while the scheduling metadata for memory capacity scales up to 16 KB (for 1,024 concurrent requests), it is offloaded and stored entirely within the trusted region of the CPU memory managed by Arm CCA, completely eliminating the need for expensive on-chip storage.

VI. Discussion

A. Security Analysis & Attack Scenarios

SecPIMlet enforces three core security invariants.

- **Invariant 1 (PIM-side domain isolation):** Cross-domain access and modification are prevented among co-existing PIM domains, and the PIM-side address translation path is guaranteed to be completely isolated from the malicious system software (such as OSes and hypervisors).
- **Invariant 2 (Hardware-enforced PIM context switching):** During a PIM context switching, all architectural states from the preceding context must be cleared, while the context being preserved must be securely stored within a hardware-reserved PIM memory domain under exclusive trusted hardware access.
- **Invariant 3: (Secure CPU-PIM communication):** The CPU-PIM communication channel ensures the confidentiality, integrity, and freshness of all transferred data.

There are several attacks that violate these security invariants.

Isolation violation (Invariant 1): A malicious domain may attempt to breach domain boundaries to access the memory region allocated to another domain. *SecPIMlet* prevents this by extending virtual addressing for PIM-side operations, ensuring that each domain can only access its own memory region.

Compromised system software (Invariant 1, 2): An adversary with OS or hypervisor privileges may try to manipulate PIM-side address translation or PIM PE mapping to access unprivileged region. *SecPIMlet* defends these attempts through two ways. First, an attempt to alter the PIM-side translation, such as modifying the PIM-side translation table or causing malfunction in the PIM translation unit, is prevented since the dedicated context management hardware operates within the trusted hardware domain. Second, attacks targeting spatio-temporal sharing are also prohibited because context handling tasks (`PIM-Alloc` and `Switch`) are executed by trusted context management hardware.

Physical attack (Invariant 1, 3): Attackers may employ physical attacks to extract data from CPU-PIM channels or PIM chips. *SecPIMlet* protects data both in-transit and at-rest. In-transit data is secured via encrypted, integrity-verified channels, blocking interception or tampering in transit. At-rest data is protected against direct chip-level attacks through secure packaging (Section III-B). Additionally, chip-boundary encryption mitigates cold-boot attacks.

Replay attack (Invariant 1, 3): Malicious system software or physical attackers may attempt to replay past data or commands. *SecPIMlet* prevents these exploits through hardware isolation, secure packaging, and protected CPU-PIM channels. First, the trusted hardware isolates the execution status from the OS or hypervisor, eliminating replay attempts from a malicious system software. Second, secure packaging prevents physical replay attacks targeting at-rest data. Lastly, for physical replay attacks on the CPU-PIM channel, synchronized monotonic counters guarantee data freshness.

Threat of PIM-side trusted hardware (Invariant 1, 2, 3): The PIM-side context management hardware faces security risks from supply chain vulnerabilities and unauthorized command injection. To counter these threats, *SecPIMlet* validates the integrity of the PIM-side hardware through secure boot and attestation. Leveraging this attested hardware, a secure CPU-PIM channel is established. (Section IV-A) Therefore, the entire command processing pipeline from the CPU to the PIM is isolated using a CPU trusted execution environment (TEE), a secure CPU-PIM channel, and attested PIM hardware.

B. Side-channel Attack & Mitigation Technique

While the traditional confidential computing excludes side-channel attacks, we analyze side-channel risks introduced by our additional components and their mitigation strategies, aligning with modern high-level security solutions like *Apple Private Cloud Compute (PCC)* [5]. Note that these side-channel vulnerabilities are easily exploited across shared hardware resources, including GPU execution contexts in MPS and Green Context [86] as well as NPU processing elements (PEs)

in NPU partitioning [21]. *SecPIMlet* causes two side-channel attack vectors: 1) Bus-level and 2) timing-based side-channel.

First, regarding bus-level side channels, the deterministic mapping of cachelines via sub-buffer interleaving to specific PIM chips creates observable data movement variations on the memory bus. An adversary capable of monitoring bus-level power and energy probing trends could potentially exploit these footprints to infer the PIM chip mapping and estimate data transfer sizes. To obscure these access patterns, the memory controller can implement traffic shaping by injecting dummy memory traffic to idle PIM chips, thereby rendering the overall bus-level activity uniform and indistinguishable, similar to *Camouflage* and *ShieldCXL* [11], [96].

Second, regarding timing-based side channels, attackers can deploy co-located tasks to induce hardware resource contention, measuring the resulting execution latency increments to launch timing-based inference exploits. Prior timing-attack frameworks can infer the model architectures based on the delayed latency [34], [36], [88]. To counteract such timing attacks, *SecPIMlet* can adopt data dimension and execution time equalization, similar to *ModelObfuscator* [95].

C. Extension to Domain-specific PIM and Hybrid PIM

The design principles of *SecPIMlet* extend beyond general-purpose PIM to domain-specific PIM. This versatility allows *SecPIMlet* to support future hybrid PIMs that integrate advantages of both general-purpose PIM (programmability) and domain-specific PIM (efficiency).

First, chip-granular address translation is compatible with the centralized translation logic found in domain-specific PIM. Since a central translation table resolves row and column addresses, this feature is implemented by adjusting the virtualization granularity of the central unit to the PIM-chip level. Second, hardware-enforced access control is applicable by delegating PIM allocation and context switching to trusted components. In domain-specific PIM, the PIM driver manages PIM PE mapping based on execution modes such as per-bank or all-bank mode. Isolating the allocation logic within the PIM driver enables the trusted PIM allocation of *SecPIMlet*. In addition, the proposed context management hardware enables secure context switching in domain-specific PIM, mirroring its functionality in general-purpose PIM. The only distinction is the definition of the PIM context. For example, the context for Samsung HBM-PIM includes registers (GRF, CRF, SRF, etc), whereas for SK Hynix AiM, it includes the global buffers and registers [33], [56]. Third, the lightweight and layout-aware secure CPU–PIM channels are directly applicable, as domain-specific PIM also have CPU-PIM channels. Lastly, the spatio-temporal scheduling is incorporable to the domain-specific PIM based on the three aforementioned security features. This allows the integration of the *SecPIMlet* scheduling policy into the allocation logic of the PIM driver.

VII. Related Work

GPU confidential computing: *Graviton* introduces a trusted hardware component which is responsible for handling GPU page table updates and validating ownership during device memory accesses [84]. Without modification of the GPU hardware, *HIX* isolates the GPU driver within a trusted execution environment (TEE), known as the GPU enclave [44]. Beyond access control, *Common Counters* and *PSSM* focus on GPU memory protection [68], [90], whereas studies on multi-GPU security introduce secure inter-GPU communication channels [67]. In industry, NVIDIA constructs the GPU confidential computing by isolating the NVIDIA driver in the confidential virtual machine [29], [72], [73].

AI accelerator confidential computing: *sNPU* extends a TEE to NPU by supporting access control and on-chip scratchpad memory (SPM) isolation [20]. *TNPU*, *MGX*, *GuardNN*, and *Tunable NPU* provide efficient NPU memory protection by leveraging tensor-based access patterns or NPU-only memory protection schemes [35], [37], [57], [59]. For system-on-a-chip (SoC) scenarios, *CryptoMMU* and *Border Control* propose access control for third-party accelerators [3], [75], while *Multi-granular MAC & Tree* provides memory protection [58].

PIM optimization: Recent PIM studies focus on addressing the CPU-PIM communication bottleneck and strict area constraints. *UM-PIM* introduces a unified memory space for both the CPU and PIM [93], and research on all-bank DRAM PIM leverages computation-communication overlapping [45]. To mitigate area limitations, *pSyncPIM* and research on sparse attention exploit sparsity-aware PIM designs for data compression and area efficiency [9], [38].

VIII. Conclusion

The paper presents *SecPIMlet*, a novel confidential spatio-temporal PIM architecture. It enables spatio-temporal sharing by breaking the translation unit down to PIM chip granularity. It also achieves fine-grained access control with minimal performance penalty by hardware-securing data and contexts within PIM banks, which reduces CPU-PIM context transfer overhead during context switching. Furthermore, *SecPIMlet* establishes a per-chip layout-aware secure CPU-PIM channel, including an additional sub-buffer dedicated to security metadata. Finally, it provides a simple yet highly effective spatio-temporal sharing policy with application-aware PIM bank management. With these protection mechanisms and scheduling policies, *SecPIMlet* safely and efficiently supports multi-domain PIM. It delivers both performance and security (isolation) benefits, unlocking the significant potential of PIM to overcome the memory wall in modern AI systems.

Acknowledgment

This work was supported by the IITP (Institute of Information & Coummunications Technology Planning & Evaluation) grants funded by Ministry of Science and ICT, Korea (Information Technology Research Center IITP-2026-RS-2021-II211817, 30% and RS-2024-00402898). It was also supported by the National Research Foundation of Korea (NRF RS-2024-00347114), by Samsung Electronics Co., Ltd. (IO251211-14338-01), and by the Swedish Research Council (grant No. 2023-03272).

REFERENCES

[1] R. Abdullah, H. Zhou, and A. Awad, "Plutus: Bandwidth-efficient memory security for GPUs," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2023.

[2] S. Aga and S. Narayanasamy, "Invisimem: Smart memory defenses for memory bus side channel," in *International Symposium on Computer Architecture (ISCA)*, 2017.

[3] F. Alam, H. Lee, A. Bhattacharjee, and A. Awad, "CryptoMMU: Enabling scalable and secure access control of third-party accelerators," in *International Symposium on Microarchitecture (MICRO)*, 2023.

[4] M. Aliannejadi, H. Zamani, F. Crestani, and W. B. Croft, "Context-aware target apps selection and recommendation for enhancing personal mobile assistants," in *Transactions on Information Systems (TOIS)*, 2021.

[5] Apple, "Private cloud compute security guide," Apple, Tech. Rep., 2024, a new frontier for AI privacy in the cloud. [Online]. Available: https://security.apple.com/documentation/private-cloud-compute

[6] Apple, "The secure enclave," Apple, Tech. Rep., 2024, apple Platform Security. [Online]. Available: https://support.apple.com/en-gb/guide/security/sec59b0b31ff/web

[7] ARM, "ARM architecture reference manual for A-profile architecture," ARM, Tech. Rep., 2022.

[8] A. Awad, Y. Wang, D. Shands, and Y. Solihin, "Obfusmem: A low-overhead access obfuscation for trusted memories," in *International Symposium on Computer Architecture (ISCA)*, 2017.

[9] D. Baek, S. Hwang, and J. Huh, "pSyncPIM: Partially synchronous execution of sparse matrix operations for all-bank PIM architectures," in *International Symposium on Computer Architecture (ISCA)*, 2024.

[10] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *International Conference on Knowledge Discovery and Data Mining (KDD)*, 2011.

[11] K. Choi, I. Kim, S. Lee, and J. Huh, "ShieldCXL: A practical obliviousness support with sealed CXL memory," in *Transactions on Architecture and Code Optimization (TACO)*, 2025.

[12] S. Choi, S. Lee, Y. Kim, J. Park, Y. Kwon, and J. Huh, "Serving heterogeneous machine learning models on multi-GPU servers with spatio-temporal sharing," in *USENIX Annual Technical Conference (USENIX ATC)*, 2022.

[13] Y. Choi and M. Rhu, "PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[14] V. Costan and S. Devadas, "Intel SGX explained," in *IACR Cryptology ePrint Archive*, 2016.

[15] C. Cui, Y. Ma, S.-Y. Park, Z. Yang, Y. Zhou, J. Lu, J. Peng, J. Zhang, R. Zhang, L. Li *et al.*, "LLM4AD: Large language models for autonomous driving–concept, review, benchmark, experiments, and auture trends," in *ArXiv:2410.15281*, 2024.

[16] F. Devaux, "The true processing in memory accelerator," in *Symposium on High Performance Chips (Hot Chips)*, 2019.

[17] J. Duan, R. Lu, H. Duanmu, X. Li, X. Zhang, D. Lin, I. Stoica, and H. Zhang, "MuxServe: Flexible spatial-temporal multiplexing for multiple LLM serving," in *International Conference on Machine Learning (ICML)*, 2024.

[18] K. D. Duy and H. Lee, "SE-PIM: In-memory acceleration of data-intensive confidential computing," in *Transactions on Cloud Computing (TCC)*, 2022.

[19] A. Fakhrzadehgan, P. Ramrakhyani, M. K. Qureshi, and M. Erez, "SecDDR: Enabling low-cost secure memories by protecting the DDR interface," in *International Conference on Dependable Systems and Networks (DSN)*, 2023.

[20] E. Feng, D. Feng, D. Du, Y. Xia, and H. Chen, "sNPU: Trusted execution environments on integrated NPUs," in *International Symposium on Computer Architecture (ISCA)*, 2024.

[21] S. Ghodrati, B. H. Ahn, J. K. Kim, S. Kinzer, B. R. Yatham, N. Alla, H. Sharma, M. Alian, E. Ebrahimi, N. S. Kim *et al.*, "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks," in *International Symposium on Microarchitecture (MICRO)*, 2020.

[22] C. Giannoula, I. Fernandez, J. G. Luna, N. Koziris, G. Goumas, and O. Mutlu, "SparseP: Towards efficient sparse matrix vector multiplication on real processing-in-memory architectures," in *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2022.

[23] K. Gogineni, S. S. Dayapule, J. Gómez-Luna, K. Gogineni, P. Wei, T. Lan, M. Sadrosadati, O. Mutlu, and G. Venkataramani, "SwiftRL: Towards efficient reinforcement learning on real processing-in-memory systems," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2024.

[24] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking a new paradigm: Experimental analysis and characterization of a real processing-in-memory system," in *IEEE Access*, 2022.

[25] J. Gómez-Luna, Y. Guo, S. Brocard, J. Legriel, R. Cimadomo, G. F. Oliveira, G. Singh, and O. Mutlu, "Machine learning training on a real processing-in-memory system," in *Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022.

[26] J. Gómez-Luna, I. E. Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking a new paradigm: An experimental analysis of a real processing-in-memory architecture," in *ArXiv:2105.03814*, 2021.

[27] Gómez-Luna, Juan and El Hajj, Izzat and Fernandez, Ivan and Giannoula, Christina and Oliveira, Geraldo F and Mutlu, Onur, "Benchmarking memory-centric computing systems: Analysis of real processing-in-memory hardware," in *International green and sustainable computing conference (IGSC)*, 2021.

[28] Gómez-Luna, Juan and Guo, Yuxin and Brocard, Sylvan and Legriel, Julien and Cimadomo, Remy and Oliveira, Geraldo F and Singh, Gagandeep and Mutlu, Onur, "Evaluating machine learningworkloads on memory-centric computing systems," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2023.

[29] Z. Gu, E. Valdez, S. Ahmed, J. J. Stephen, M. Le, H. Jamjoom, S. Zhao, and Z. Lin, "NVIDIA GPU confidential computing demystified," in *ArXiv:2507.02770*, 2025.

[30] H. Gupta, M. Kabra, J. Gómez-Luna, K. Kanellopoulos, and O. Mutlu, "Evaluating homomorphic operations on a real-world processing-in-memory system," in *International Symposium on Workload Characterization (IISWC)*, 2023.

[31] H. Han, X. Zheng, Y. Wen, Y. Hao, E. Feng, L. Liang, J. Mu, X. Li, T. Ma, P. Jin *et al.*, "TensorTEE: Unifying heterogeneous TEE granularity for efficient secure collaborative tensor computing," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

[32] L. Han, Z. Zhou, and Z. Li, "Pantheon: Preemptible multi-DNN inference on mobile edge GPUs," in *International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2024.

[33] M. He, C. Song, I. Kim, C. Jeong, S. Kim, I. Park, M. Thottethodi, and T. Vijaykumar, "Newton: A DRAM-maker's accelerator-in-memory (AiM) architecture for machine learning," in *International Symposium on Microarchitecture (MICRO)*, 2020.

[34] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood *et al.*, "Deepsniffer: A DNN model extraction framework based on learning architectural hints," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[35] W. Hua, M. Umar, Z. Zhang, and G. E. Suh, "GuardNN: Secure accelerator architecture for privacy-preserving deep learning," in *Design Automation Conference (DAC)*, 2022.

[36] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *Annual Design Automation Conference (DAC)*, 2018.

[37] Hua, Weizhe and Umar, Muhammad and Zhang, Zhiru and Suh, G Edward, "MGX: Near-zero overhead memory protection for data-intensive accelerators," in *International Symposium on Computer Architecture (ISCA)*, 2022.

[38] S. Hwang, S. Lee, J. Lee, and J. Huh, "An operand divergence-aware all-bank PIM architecture for sparse attention," in *Computer Architecture Letters (CAL)*, 2026.

[39] S. Hwang, S. Lee, J. Kim, H. Kim, and J. Huh, "mNPUsim: Evaluating the effect of sharing resources in multi-core NPUs," in *International Symposium on Workload Characterization (IISWC)*, 2023.

[40] B. Hyun, T. Kim, D. Lee, and M. Rhu, "Pathfinding future PIM architectures by demystifying a commercial PIM technology," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2024.

[41] Intel, "Intel architecture memory encryption technologies," Intel, Tech. Rep., 2022.

[42] Intel, "Intel trust domain extensions," Intel, Tech. Rep., 2022.

[43] Intel, "Intel TDX connect architecture specification," Intel, Tech. Rep., 2023.

[44] I. Jang, A. Tang, T. Kim, S. Sethumadhavan, and J. Huh, "Heterogeneous isolated execution for commodity GPUs," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[45] S. Jeong, S. Kim, C. Eom, and J. Huh, "Enabling computation and communication overlap in PIMs for on-device LLM inference," in *Computer Architecture Letters (CAL)*, 2026.

[46] G. Jonatan, H. Cho, H. Son, X. Wu, N. Livesay, E. Mora, K. Shivdikar, J. L. Abellán, A. Joshi, D. Kaeli *et al.*, "Scalability limitations of processing-in-memory using real system evaluations," in *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2024.

[47] D. Kaplan, J. Powell, and T. Woller, "AMD memory encryption," AMD, Tech. Rep., 2016.

[48] D. Kim, T. Kim, I. Hwang, T. Park, H. Kim, Y. Kim, and Y. Park, "Virtual PIM: Resource-aware dynamic DPU allocation and workload scheduling framework for multi-DPU PIM architecture," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2023.

[49] J. H. Kim, S.-H. Kang, S. Lee, H. Kim, Y. Ro, S. Lee, D. Wang, J. Choi, J. So, Y. Cho *et al.*, "Aquabolt-XL HBM2-PIM, LPDDR5-PIM with in-memory processing, and AXDIMM with acceleration buffer," in *IEEE Micro*, 2022.

[50] J. H. Kim, Y. Ro, J. So, S. Lee, S.-h. Kang, Y. Cho, H. Kim, B. Kim, K. Kim, S. Park *et al.*, "Samsung PIM/PNM for Transformer based AI: Energy efficiency on PIM/PNM cluster," in *Hot Chips Symposium (HotChips)*, 2023.

[51] Y. Kwon, K. Vladimir, N. Kim, W. Shin, J. Won, M. Lee, H. Joo, H. Choi, G. Kim, B. An *et al.*, "System architecture and software stack for GDDR6-AiM," in *Hot Chips Symposium (Hot Chips)*, 2022.

[52] D. Lee, B. Hyun, T. Kim, and M. Rhu, "PIM-MMU: A memory management unit for accelerating data transfers in commercial PIM systems," in *International Symposium on Microarchitecture (MICRO)*, 2024.

[53] H. Lee, D. Baek, J. Son, J. Choi, K. Moon, and M. Jang, "PAISE: PIM-accelerated inference scheduling engine for Transformer-based LLM," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2025.

[54] S. Lee, K. Kim, S. Oh, J. Park, G. Hong, D. Ka, K. Hwang, J. Park, K. Kang, J. Kim *et al.*, "A 1ynm 1.25V 8Gb, 16Gb/s/Pin GDDR6-based accelerator-in-memory supporting 1TFLOPS MAC operation and various activation functions for deep-learning applications," in *International Solid-State Circuits Conference (ISSCC)*, 2022.

[55] S. Lee, C. Lim, J. Choi, H. Choi, C. Lee, Y. Park, K. Park, H. Kim, and Y. Kim, "SPID-Join: A skew-resistant processing-in-DIMM join algorithm exploiting the bank-and rank-level parallelisms of DIMMs," in *ACM SIGMOD Conference (SIGMOD)*, 2024.

[56] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin *et al.*, "Hardware architecture and software stack for PIM based on commercial DRAM technology: Industrial product," in *International Symposium on Computer Architecture (ISCA)*, 2021.

[57] S. Lee, J. Kim, S. Na, J. Park, and J. Huh, "TNPU: Supporting trusted execution with tree-less integrity protection for neural processing unit," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2022.

[58] S. Lee, S. Na, J. Choi, J. Pyo, and J. Huh, "Unified memory protection with multi-granular MAC and integrity tree for heterogeneous processors," in *International Symposium on Computer Architecture (ISCA)*, 2025.

[59] S. Lee, S. Na, J. Kim, J. Park, and J. Huh, "Tunable memory protection for secure neural processing units," in *International Conference on Computer Design (ICCD)*, 2022.

[60] C. Lim, S. Lee, J. Choi, J. Lee, S. Park, H. Kim, J. Lee, and Y. Kim, "Design and analysis of a processing-in-DIMM Join algorithm: A case study with UPMEM DIMMs," in *ACM SIGMOD Conference (SIGMOD)*, 2023.

[61] M.-B. Lin and J.-H. Chuang, "The design of a high-throughput hardware architecture for the AES-GCM algorithm," in *Transactions on Consumer Electronics (TCE)*, 2023.

[62] Y. Lin, C. Fang, X. Song, Q. Wu, A. Jiang, Y. Bai, and L. Du, "CD-PIM: A high-bandwidth and compute-efficient LPDDR5-based PIM for low-batch LLM acceleration on edge-device," in *ArXiv:2601.12298*, 2026.

[63] J. Liu, M. Zhou, Y. Pan, C.-Y. Yang, L. Josipovic, and T. Rosing, "OptiPIM: Optimizing processing-in-memory acceleration using integer linear programming," in *International Symposium on Computer Architecture (ISCA)*, 2025.

[64] L. Liu, K. G. Shin, J. Lee, C. Yang, and W. Shi, "Enhancing predictability of multi-tenant DNN inference for autonomous vehicles' perception," in *ArXiv:2602.11004*, 2026.

[65] D. MAURER, "Selecting an embedded MCU: How to avoid evaluation trap?" Design & Reuse, Tech. Rep., 2010, industry Articles. [Online]. Available: https://www.design-reuse.com/article/59553-selecting-an-embedded-mcu-how-to-avoid-evaluation-trap-/

[66] N. Muralimanohar, B. Rajeev, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," Tech. Rep., 2009.

[67] S. Na, J. Kim, S. Lee, and J. Huh, "Supporting secure multi-GPU computing with dynamic and batched metadata management," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2024.

[68] S. Na, S. Lee, Y. Kim, J. Park, and J. Huh, "Common counters: Compressed encryption counters for secure GPU memory," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2021.

[69] J. Nider, J. Dagger, N. Gharavi, D. Ng, and A. Fedorova, "Bulk JPEG decoding on in-memory processors," in *ACM International Conference on Systems and Storage (SYSTOR)*, 2022.

[70] J. Nider, C. Mustard, A. Zoltan, J. Ramsden, L. Liu, J. Grossbard, M. Dashti, R. Jodin, A. Ghiti, J. Chauzi *et al.*, "A case study of processing-in-memory in off-the-shelf systems," in *Annual Technical Conference (ATC)*, 2021.

[71] S. U. Noh, J. Hong, C. Lim, S. Park, J. Kim, H. Kim, Y. Kim, and J. Lee, "PID-Comm: A fast and flexible collective communication framework for commodity processing-in-DIMM devices," in *International Symposium on Computer Architecture (ISCA)*, 2024.

[72] NVIDIA, "Confidential compute on NVIDIA Hopper H100," NVIDIA, Tech. Rep., 2023.

[73] NVIDIA, "NVIDIA secure AI with Blackwell and Hopper GPUs," NVIDIA, Tech. Rep., 2025.

[74] NVIDIA, "NVIDIA RTX Spark," NVIDIA, Tech. Rep., 2026, NVIDIA RTX Spark. A new beginning. [Online]. Available: https://www.nvidia.com/en-gb/products/rtx-spark/

[75] L. E. Olson, J. Power, M. D. Hill, and D. A. Wood, "Border Control: Sandboxing accelerators," in *International Symposium on Microarchitecture (MICRO)*, 2015.

[76] S. Rhyner, H. Luo, J. Gómez-Luna, M. Sadrosadati, J. Jiang, A. Olgun, H. Gupta, C. Zhang, and O. Mutlu, "PIM-Opt: Demystifying distributed optimization algorithms on a real-world processing-in-memory system," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2024.

[77] S. Seo, J. Kim, D. Lee, S. Yoo, S. Moon, Y. Park, and J. W. Lee, "FACIL: Flexible DRAM address mapping for SoC-PIM cooperative on-device LLM inference," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2025.

[78] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, "Nexus: A GPU cluster engine for accelerating DNN-based video analysis," in *Symposium on Operating Systems Principles (SOSP)*, 2019.

[79] Y. Shin, D. Kang, and H. Sung, "ATiM: Autotuning tensor programs for processing-in-DRAM," in *International Symposium on Computer Architecture (ISCA)*, 2025.

[80] G. Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love *et al.*, "Gemma: Open models based on Gemini research and technology," in *ArXiv:2403.08295*, 2024.

[81] G. Team, M. Riviere, S. Pathak, P. G. Sessa, C. Hardin, S. Bhupatiraju, L. Hussenot, T. Mesnard, B. Shahriari, A. Ramé *et al.*, "Gemma 2: Improving open language models at a practical size," in *ArXiv:2408.00118*, 2024.

[82] K. Umemoto, "ML-1M++: Movielens-compatible additional preferences for more robust offline evaluation of sequential recommenders," in *International Conference on Information and Knowledge Management (CIKM)*, 2022.

[83] V. Vančura, R. Alves, P. Kasalickỳ, and P. Kordík, "Scalable linear shallow autoencoder for collaborative filtering," in *ACM Conference on Recommender Systems (RecSys)*, 2022.

[84] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on GPUs," in *Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[85] N. Wang, M. Matthaiou, D. S. Nikolopoulos, and B. Varghese, "Dyverse: Dynamic vertical scaling in multi-tenant edge environments," in *Future Generation Computer Systems (FGCS)*, 2020.

[86] J. Wei, Y. Zhang, Z. Zhou, Z. Li, and M. A. Al Faruque, "Leaky DNN: Stealing deep-learning model secret with GPU context-switching side-channel," in *International Conference on Dependable Systems and Networks (DSN)*, 2020.

[87] D. Xu, H. Zhang, L. Yang, R. Liu, G. Huang, M. Xu, and X. Liu, "Fast on-device LLM inference with NPUs," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2025.

[88] M. Yan, C. W. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to learn DNN architectures," in *USENIX Security Symposium (USENIX Security)*, 2020.

[89] S. Yu, H. Kim, K. Jeun, S. Hwang, S. Cho, and E. Lee, "ComPASS: A compatible PIM protocol architecture and scheduling solution for processor-PIM collaboration," in *International Symposium on Microarchitecture (MICRO)*, 2025.

[90] S. Yuan, Y. Solihin, and H. Zhou, "PSSM: Achieving secure memory for GPUs with partitioned and sectored security metadata," in *International Conference on Supercomputing (ICS)*, 2021.

[91] P. Zhang, G. Zeng, T. Wang, and W. Lu, "TinyLlama: An open-source small language model," in *Arxiv:2401.02385*, 2024.

[92] Y. Zhang, K. Yang, M. Saligane, D. Blaauw, and D. Sylvester, "A compact 446 Gbps/W AES accelerator for mobile SoC and IoT in 40nm," in *Symposium on VLSI circuits (VLSI-circuits)*, 2016.

[93] Y. Zhao, M. Gao, F. Liu, Y. Hu, Z. Wang, H. Lin, J. Li, H. Xian, H. Dong, T. Yang *et al.*, "UM-PIM: DRAM-based PIM with uniform & shared memory space," in *International Symposium on Computer Architecture (ISCA)*, 2024.

[94] Y. Zheng, B. Tang, W. Ding, and H. Zhou, "A neural autoregressive approach to collaborative filtering," in *International Conference on Machine Learning (ICML)*, 2016.

[95] M. Zhou, X. Gao, J. Wu, J. Grundy, X. Chen, C. Chen, and L. Li, "Modelobfuscator: Obfuscating model information to protect deployed ML-based systems," in *International Symposium on Software Testing and Analysis (ISSTA)*, 2023.

[96] Y. Zhou, S. Wagh, P. Mittal, and D. Wentzlaff, "Camouflage: Memory traffic shaping to mitigate timing attacks," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2017.

[97] S. Zobaed, A. Mokhtari, J. P. Champati, M. Kourouma, and M. A. Salehi, "Edge-multiAI: Multi-tenancy of latency-sensitive deep learning applications on edge," in *International Conference on Utility and Cloud Computing (UCC)*, 2022.