

Article

SPARTA: Sparse Parallel Architecture for Real-Time Threat Analysis for Lightweight Edge Network Defense

Shi Li ¹, Xiyun Mi ¹, Lin Zhang ^{2,*} and Ye Lu ^{1,*}¹ College of Cryptology and Cyber Science, Nankai University, Tianjin 300350, China; lishi@mail.nankai.edu.cn (S.L.)² Chinese Institute of New Generation Artificial Intelligence Development Strategies, Nankai University, Tianjin 300350, China

* Correspondence: zhanglin@nankai.edu.cn (L.Z.); luye@nankai.edu.cn (Y.L.)

Abstract

AI-driven network security relies increasingly on Large Language Models (LLMs) to detect sophisticated threats; however, their deployment on resource-constrained edge devices is severely hindered by immense parameter scales. While unstructured pruning offers a theoretical reduction in model size, commodity Graphics Processing Unit (GPU) architectures fail to efficiently leverage element-wise sparsity due to the mismatch between fine-grained pruning patterns and the coarse-grained parallelism of Tensor Cores, leading to latency bottlenecks that compromise real-time analysis of high-volume security telemetry. To bridge this gap, we propose SPARTA (Sparse Parallel Architecture for Real-Time Threat Analysis), an algorithm–architecture co-design framework. Specifically, we integrate a hardware-based address remapping interface to enable flexible row-offset access. This mechanism facilitates a novel graph-based column vector merging strategy that aligns sparse data with Tensor Core parallelism, complemented by a pipelined execution scheme to mask decoding latencies. Evaluations on Llama2-7B and Llama2-13B benchmarks demonstrate that SPARTA achieves an average speedup of 2.35× compared to Flash-LLM, with peak speedups reaching 5.05×. These findings indicate that hardware-aware microarchitectural adaptations can effectively mitigate the penalties of unstructured sparsity, providing a viable pathway for efficient deployment in resource-constrained edge security.

Keywords: LLM Sparsity; GPU architecture extension; algorithm-architecture co-design; AI-driven network security; edge computing



Academic Editor: Paolo Bellavista

Received: 28 December 2025

Revised: 22 January 2026

Accepted: 28 January 2026

Published: 6 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Cyber-Physical Systems (CPSs) and Industrial Control Systems (ICSs) constitute the critical backbone of modern infrastructure, integrating computation with physical processes. However, the increasing connectivity of these systems has exposed them to sophisticated cyber threats, particularly Distributed Denial of Service (DDoS) attacks [1,2]. Traditional signature-based detection methods are becoming inadequate against evolving attack patterns, such as low-rate DDoS or semantic attacks that mimic legitimate traffic. Consequently, Artificial Intelligence (AI), specifically Large Language Models (LLMs) and Transformer-based architectures [3–6], have emerged as a promising solution for robust DDoS detection and traffic anomaly analysis. These models can capture complex, non-linear dependencies in network logs, significantly enhancing network robustness.

Despite their superior detection accuracy, deploying large-scale AI models within CPS environments presents a fundamental challenge: the conflict between model complexity and the resource constraints of edge devices. CPS gateways and terminal controllers typically operate under strict power limits and possess limited memory, far less than cloud-grade GPUs like the NVIDIA RTX 4090 [7]. Although recent inference systems like PowerInfer [8] and FlexGen [9] attempt to optimize large model serving on consumer-grade hardware, the sheer scale of sophisticated models required for semantic reasoning over aggregated network logs—often containing billions of parameters [10]—creates fundamental obstacles for deployment. To address this, unstructured pruning [11–13] is widely adopted to reduce model size and memory footprint. Unlike structured pruning [14,15] which limits accuracy, unstructured pruning can remove up to 50–70% of redundant parameters while maintaining the high precision (accuracy loss < 1%) required for reliable threat detection.

However, realizing the theoretical efficiency of unstructured pruning on commodity hardware remains a bottleneck for real-time CPS security. Existing GPU architectures, particularly Tensor Cores (TC), are designed for dense computation. As illustrated in Figure 1, the mismatch between fine-grained sparsity and coarse-grained hardware tiles leads to significant inefficiencies: software approaches suffer from zero-value computation waste, while naive hardware extensions incur severe bank conflicts. These bottlenecks prevent existing solutions from meeting the real-time requirements of CPS security. This granularity difference creates two key problems: (1) storage fragmentation prevents effective register utilization; (2) zero-value elements still participate in computations. Research from DTC-SpMM [16] shows that when the effective non-zero element density drops, Tensor Core utilization falls below 12.5%. For a CPS Intrusion Detection System (IDS), this hardware inefficiency translates to increased inference latency and jitter, potentially delaying the identification of complex attack patterns from telemetry streams.

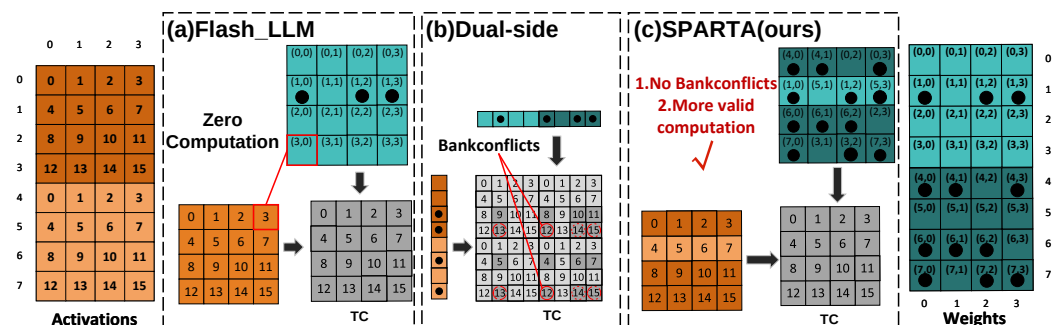


Figure 1. Tiled Sparse General Matrix-Matrix Multiplication (SpGEMM) accelerations and the impact on efficiency.

Current research addresses this mismatch through software optimization (e.g., Flash-LLM [17], FlashDecoding++ [18]) or hardware extensions (e.g., Dual-Side [19]), but both fall short of CPS requirements. While software pipelines like Flash-LLM employ a “Load-as-Sparse, Compute-as-Dense” paradigm, their complex decoding overheads and runtime scheduling introduce additional latency bubbles. This makes them unsuitable for high-throughput traffic inspection, where handling bursty DDoS packets requires consistent processing speeds. On the hardware side, Dual-Side attempts to leverage dual sparsity but suffers from severe Shared Memory Bank Conflicts. As shown in Figure 1, when accessing irregular sparse features typical of network data, conflicts reduce bandwidth utilization by 40–60%. In industrial control loops where deterministic response times are critical, such performance unpredictability undermines the reliability of the safety mechanism.

To bridge the gap between high-accuracy AI detection algorithms and resource-constrained CPS edge hardware, this paper proposes SPARTA. Our approach relies on a key

insight: non-zero elements, even in unstructured pruning, can be intelligently aggregated to match hardware granularity. We introduce a novel granularity conversion method that reorganizes fine-grained sparse patterns into coarse-grained block structures matching Tensor Core tiles (16×16). This design significantly increases the density of non-zero elements within computation tiles (Nnz/Tile), ensuring that AI-based defense models can run efficiently on edge GPUs.

Specifically, SPARTA (Sparse Parallel Architecture for Real-Time Threat Analysis) is a collaborative framework based on microarchitectural extensions. First, we model the weight matrix merging process as a graph coloring problem, achieving optimized resource allocation. Second, we introduce channel permutation to align sparse data. Finally, and most critically for hardware efficiency, we extend the Shared Memory I/O interface with a remapping logic. This allows the GPU to access merged, conflict-free data patterns while the hardware transparently handles the addressing logic. As shown in Figure 1, SPARTA enables zero-conflict memory access, strictly adhering to the GPU's bank parallel access rules, effectively doubling storage density and eliminating the bank conflict bottlenecks that plague current solutions.

We make the following contributions to enabling efficient AI security on the edge:

- We thoroughly analyze the workflow of unstructured pruning on Tensor Cores, identifying the specific bottlenecks that hinder real-time inference in resource-constrained environments.
- We propose a graph-coloring-based weight merging algorithm and a channel permutation strategy that transforms irregular sparse patterns into hardware-friendly block structures, ensuring high utilization of computing units.
- We design SPARTA, a lightweight microarchitectural extension providing a conflict-free Shared Memory address redirection interface. This design solves the bank conflict issue inherent in previous hardware sparse accelerators.
- The experimental results demonstrate that SPARTA achieves an average speedup of $2.35\times$ (up to $5.05\times$) compared to state-of-the-art methods. These findings validate that SPARTA provides a viable hardware foundation for deploying sophisticated, real-time DDoS detection models in resource-constrained edge security.

The rest of this paper is organized as follows: Section 2 provides the background on AI-driven defense mechanisms in CPS and analyzes the hardware constraints of edge GPU architectures. Section 3 presents the motivation, highlighting the specific latency and determinism bottlenecks in existing sparse accelerators. Section 4 details the software design of our conflict-free sparse reorganization strategy, incorporating similarity-driven regularization and graph-theoretic consolidation. Section 5 presents the hardware realization of SPARTA, detailing the microarchitectural extensions and the asynchronous pipeline design. Section 6 evaluates the performance, energy efficiency, and area overhead of our proposed framework through comprehensive experiments. Finally, Section 7 concludes this paper and discusses future directions.

2. Background

In this section, we provide the theoretical and architectural context necessary to understand the challenges in edge-based threat analysis. We first discuss the role of model sparsification in deploying AI defenses on constrained CPS devices. Next, we analyze the specific hardware constraints of edge GPUs that hinder current sparse implementations. Finally, we examine the latency and determinism bottlenecks present in existing acceleration approaches.

2.1. Sparsification in AI-Driven CPS Defense

Deep Learning models, particularly LLMs, have shown superior capability in detecting semantic-layer DDoS attacks compared to traditional statistical methods. However, deploying these models on resource-constrained CPS edge gateways requires significant compression. Model pruning technology serves as a critical enabler by selectively removing parameters to reduce memory footprint and computational overhead.

Existing methods are categorized by granularity: structured, semi-structured, and unstructured pruning. Structured pruning methods (e.g., LLM-Pruner [14], CoFi [20], SliceGPT [15]) remove entire components like heads or layers. While this aligns well with traditional hardware, the coarse-grained removal often destroys the subtle feature extraction capabilities needed to detect sophisticated, low-rate DDoS attacks, leading to significant accuracy decline. Semi-structured pruning (e.g., N:M sparsity in NVIDIA Ampere) offers a middle ground but is limited to 50% sparsity, which is often insufficient for constrained industrial edge devices.

In contrast, unstructured pruning (e.g., SparseGPT [11], Wanda [12]) is highly favorable for security applications. By performing fine-grained weight selection, it achieves higher sparsity rates (reducing memory usage significantly) while maintaining accuracy loss below 1%. As shown in Table 1, unstructured pruning significantly outperforms other methods in perplexity at high sparsity (>50%). This precision is crucial for maintaining the semantic understanding required to distinguish between legitimate industrial traffic and malicious anomalies. Therefore, this paper focuses on optimizing unstructured pruning to enable high-performance, high-accuracy threat detection on the edge.

Table 1. Accuracy Comparison.

Method	Pattern	Sparsity	LLaMA				LLaMA-2		
			7B	13B	30B	65B	7B	13B	70B
Dense	–	0%	5.68	5.09	4.77	3.56	5.12	4.57	3.12
FLAP [21]	Structured	30%	17.62	15.65	12.49	10.90	7.82	14.86	10.09
Wanda [12]	Semi-Structured	50%	11.53	9.58	6.90	6.25	11.02	8.27	5.16
Wanda [12]	Unstructured	60%	10.66	8.56	6.49	5.83	9.71	7.75	4.98

2.2. Hardware Constraints of Edge GPU Architectures

The NVIDIA Tensor Core (TC) [22] has become the standard accelerator for AI workloads in both data center and edge GPUs (e.g., RTX series used in industrial servers). Compared to traditional Single Instruction Multiple Thread (SIMT) cores (CUDA cores), TC provides orders-of-magnitude higher throughput for dense matrix multiplication. For example, in the A100 [23] GPU, FP32 accumulation throughput is 16 times higher than SIMT cores. Additionally, unstructured pruning and SIMT core sparse matrix multiplication acceleration schemes operate at the granularity of individual elements, allowing SIMT cores to easily skip zero element computations in Sparse Matrix-Matrix Multiplication (SpMM). However, TCs operate on coarse-grained tiles (e.g., 16×16), making them inherently inefficient for the fine-grained, irregular patterns produced by unstructured pruning. A single TC instruction cannot skip computations at the individual element level, meaning zero-values in sparse models still consume power and cycles.

Furthermore, Shared Memory (SMEM) architecture poses a specific challenge for sparse access. SMEM is organized into 32 banks. When a warp of threads accesses addresses that map to different rows within the same bank, a Bank Conflict occurs. In a real-time CPS context, bank conflicts are detrimental because they force parallel access requests to be serialized. This serialization introduces unpredictable latency spikes (jitter), which

can violate the strict timing requirements of industrial control protocols when processing high-velocity network traffic.

2.3. Latency and Determinism Bottlenecks in Existing Accelerators

Current research on Tensor Core-based sparse matrix multiplication (SpMM) acceleration primarily develops along two directions: computational strategy optimization at the software level and architectural extensions at the hardware level. These approaches aim to address the efficiency bottlenecks faced by traditional SIMT cores when processing sparse computations while attempting to leverage the immense matrix throughput of Tensor Cores. However, when applied to time-sensitive CPS security applications, both directions exhibit limitations that hinder real-time DDoS detection performance.

Regarding software optimization solutions, as illustrated in Figure 1a, Flash-LLM [17] innovatively proposes a “load as sparse, compute as dense” data invocation strategy. This approach filters and reorganizes sparse data during the loading phase, converting it into dense matrices before processing by Tensor Cores. Its optimization focus lies in reducing the memory footprint to accommodate large models on limited memory devices. This enables Flash-LLM [17] to demonstrate significant performance advantages in low sparsity scenarios (70–90%) and memory-bound conditions ($N = 8, 16, 32$). However, for low-latency edge inference, this approach has two critical limitations: (1) It fails to effectively utilize the tile-wise computational characteristics of Tensor Cores, forcibly converting sparse computations into complete dense matrix operations. This results in numerous zero-value elements still participating in ineffective computations (as shown in Figure 1a, where black dots represent non-zero elements, and zero-element computations account for 11/16). This waste of energy is critical for battery-powered or passive-cooled industrial gateways. (2) The sparse-to-dense data conversion process introduces additional memory access overhead, including index reorganization and data movement operations. In a high-throughput DDoS attack scenario, this decoding overhead competes for CPU/GPU cycles needed for packet parsing, creating a bottleneck that limits the system’s ability to handle bursty traffic.

Regarding hardware extension strategies, as shown in Figure 1b, research such as Dual-Side [19] approaches the problem from GPU architectural modifications, adapting computational paradigms to accommodate sparse characteristics. This method transforms the traditional inner product-based computation mode into an outer product computation and correspondingly modifies the arithmetic units and accumulation buffer structures of Tensor Cores. Its innovation lies in simultaneously considering the dual sparsity of both activations (network inputs) and weights (model parameters). However, when non-zero elements are densely distributed—common in fine-tuned security models—this method encounters severe bank conflict issues. Although researchers attempt to mitigate this using NVIDIA’s traditional operand collectors [24], due to the random memory access characteristics introduced by unstructured sparsity, significant bank conflicts still occur under high instruction-level parallelism. As shown in Figure 1, serious storage bank conflicts occur (e.g., two-way conflicts where activation values at relative positions 0 and 2 both come from bank0). These conflicts reduce theoretical bandwidth utilization by 40–60%. For industrial control systems requiring deterministic response times, this unpredictability causes inference latency jitter, potentially delaying the blocking of malicious control commands.

Overall, using Tensor Cores for fine-grained sparse matrix multiplication optimization presents two obvious mismatches with existing architectures, specifically in computational patterns and memory access granularity. Specifically, Tensor Cores employ coarse-grained computation and access patterns, with their minimum computational unit being a complete computational block, such as a single instruction that can complete multipli-

cation of 8×4 and 4×8 matrices. However, fine-grained sparse patterns cannot achieve performance benefits on the tile-wise Tensor Core computational mode. Furthermore, when computing SpMM for real-time traffic analysis, element-wise sparse access violates SMEM's memory access patterns, causing frequent bank conflicts that undermine the stability required for robust CPS protection.

3. Motivation

As mentioned above, software-based optimization algorithms face the challenge of effectively utilizing GPU tensor cores, which are high-performance computational units. Meanwhile, naive hardware extension approaches that convert sparse tensors to dense tensors through consolidation result in bank conflicts during data access. In the context of Cyber-Physical Systems (CPS), these hardware-level inefficiencies translate into unpredictable latency variance, posing severe risks to real-time security. Therefore, we aim to achieve sparse tensor consolidation in a bank conflict-free manner to reduce computational and storage overhead, ensuring the timeliness and stability required for DDoS defense on the edge.

Observation: Regarding the granularity of consolidation, bank conflicts are prone to occur when weights are directly compressed by rows or columns and then accelerated using tensor cores. As shown in Figure 2, the left side illustrates the workflow of tensor core-accelerated GEMM, where the pink TC 0 represents the weight dimensions, and the green color represents the distribution of activation values stored across 32 banks in shared memory. Activation 0 is distributed across 32 banks, and this approach reads operands through strictly aligned addressing and writes back the results. The right side shows direct compression based on the pruning granularity of non-zero elements. When using tensor cores for acceleration, bank conflicts occur when reading the corresponding activation values for weights, as different threads access different addresses within the same bank. This conflict arises because non-zero elements at the same relative positions across different TCs appear simultaneously in one TC. For example, in Figure 2, bank conflict occurs at the 5th bank, where the 5th non-zero element of pink TC 0 and the 5th non-zero element of yellow TC 2 both appear in the new TC_0. Due to GPU parallelism, conflicts arise when accessing the activation values corresponding to these two elements simultaneously. Such access collisions force serialization of memory requests, causing significant latency spikes (jitter) that can delay the identification of malicious packets in high-speed industrial networks.

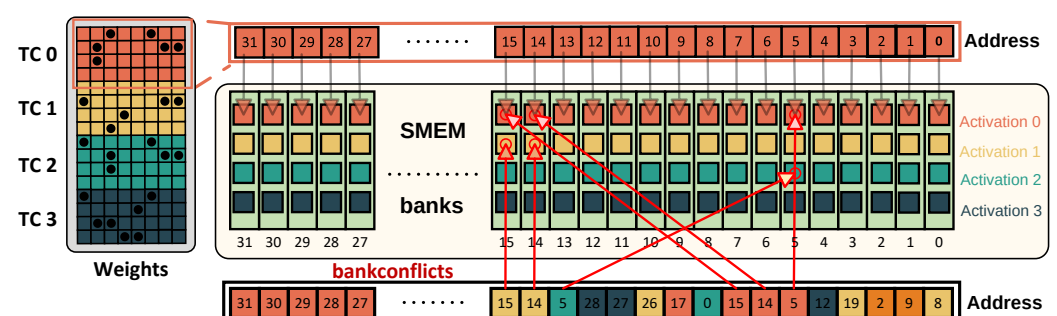


Figure 2. Tensor Core workflow and bank conflicts.

Challenges:

Lack of Hardware-Friendly Consolidation Algorithms: If sparse weight matrices are encoded for storage and then decoded for computation, when local sparsity is high, a large number of zero elements participate in computations, resulting in wasted Tensor Core resources. Alternatively, if sparse weight matrices are directly consolidated or activa-

tion values are reordered according to storage formats, data loading during computation incurs significant memory access overhead. There is an urgent need for a hardware-friendly sparse weight matrix consolidation algorithm that can improve the proportion of non-zero elements in current Tensor Core computations while ensuring memory access efficiency, thereby achieving effective SpMM acceleration. Addressing this is critical for deploying high-precision, large-scale DDoS detection models on resource-constrained industrial gateways without compromising throughput.

Hardware Requires Flexible Address Mapping Implementation: For sparse weight matrices, using SIMT cores to decode sparse formats inevitably introduces additional memory access overhead. Moreover, if this overhead is substantial, it generates numerous SpMM pipeline bubbles, leading to performance degradation. Therefore, the data processing of consolidated sparse weights during computation poses challenges to traditional GPU architectures (particularly storage components). How to implement flexible and efficient memory address access mechanisms becomes a critical issue for SpMM acceleration schemes. Eliminating these pipeline bubbles is essential for maintaining line-rate processing speeds, preventing packet loss during volumetric DDoS attacks due to hardware stalls.

Based on these findings, this paper proposes SPARTA, a software-hardware co-design approach that transforms irregular sparse problems into hardware-friendly dense formats. Without disrupting the native workflow of Tensor Cores, this approach fully leverages their dense computing efficiency to accelerate sparse LLM inference. This effectively bridges the gap between sophisticated AI algorithms and the strict real-time requirements of CPS security defense.

4. Conflict-Free Sparse Tensor Reorganization

In this section, we present the software core of the **SPARTA** co-design framework. To promote the predictable latency required for real-time DDoS detection in CPS environments, we offload the complexity of handling irregular sparsity from the runtime inference phase to an offline reorganization phase. This approach fundamentally reorganizes stochastic sparse weight distributions into hardware-friendly dense formats that fully saturate Tensor Core throughput without triggering bank conflicts. The process consists of two sequential stages: First, we employ Similarity-Driven Pattern Regularization to maximize the local similarity of sparse patterns, creating favorable conditions for consolidation. Subsequently, we apply Graph-Theoretic Conflict-Free Consolidation, which mathematically models the hardware resource contention as a graph coloring problem. By resolving these constraints offline, we ensure that the deployed defense model executes with maximum efficiency and predictability on resource-constrained edge devices.

4.1. Similarity-Driven Pattern Regularization

Native tile-wise General Matrix-Matrix Multiplication (GEMM) computations executed on Tensor Cores (TCs) process multiple columns in parallel. While the CUDA programming model typically exposes Tensor Core operations as Warp-level 16×16 tiles, the underlying microarchitecture computes partial products in finer granularities. To balance pruning flexibility with hardware alignment, we abstract the fundamental merging unit as a 4×4 sub-core block. However, a significant mismatch exists between this coarse-grained hardware requirement and the fine-grained sparsity of pruned models. In scenarios where non-zero elements are sparsely scattered, a TC block may be allocated to process a region containing only a single effective weight. Yet, due to hardware constraints, the system must still schedule the entire 4×4 tile for computation, resulting in substantial waste of computational resources and register space.

To mitigate this fragmentation, we propose a hierarchical clustering permutation strategy (Compact Column Permutation). This strategy performs column permutation within a defined local window to aggregate non-zero elements spatially. The process involves recursive clustering with $\log_2 K$ iterations, where K represents the matrix dimension (M , N , or K) handled by the TCs. Inspired by DTC-SpMM [16], we employ the Dice coefficient to cluster columns or groups with high similarity. This aggregation effectively “regularizes” the sparsity pattern, resulting in a more balanced workload distribution across Tensor Cores.

Crucially, due to the commutative property of matrix multiplication, the integrity of the final result is preserved as long as the corresponding row permutation is applied to the input activation matrix. Furthermore, this permutation involves only index adjustments without actual arithmetic computation. The additional overhead accounts for less than 1% of the overall GEMM process—negligible compared to the performance gains achieved by eliminating load imbalance. Through this optimization, we maximize Tensor Core utilization and minimize resource idling caused by sparse, elongated partial columns.

4.2. Graph-Theoretic Conflict-Free Consolidation

To achieve high compression ratios while maintaining regular and conflict-free memory access patterns for Tensor Cores, we propose a conflict-free column merging strategy. Unlike the Dual-Side [19] approach, which compresses based on global coordinates, our method utilizes individual Tensor Cores (referred to as sub-cores) as the fundamental merging granularity. This enables column-dimension compression strictly within hardware compatibility constraints.

Specifically, when a weight matrix is processed by four Tensor Cores in a 4×4 block format, elements from different rows within a single column are mapped to distinct Shared Memory banks and indexed by different threads. Arbitrarily adjusting the relative positions of elements during merging alters the mapping between threads and banks, leading to Bank Conflicts. Therefore, the merging operation must satisfy a strict position conflict-free constraint: sub-cores to be merged must not have non-zero elements at the same row index (as shown in Figure 3, sub-cores with overlapping row positions cannot be merged).

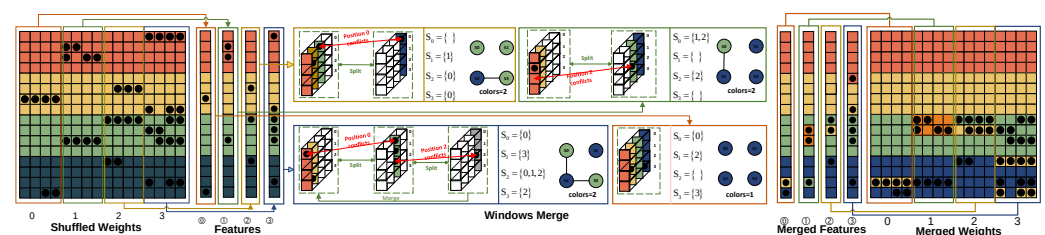


Figure 3. Graph-Based Conflict-Free Sub-core Merging Mechanism.

To maximize sub-core merging and minimize memory footprint, we model this constraint as a graph coloring problem with the following formal definitions:

Sub-core Feature Representation: Given a sub-core size of $M \times M$ (where $M = 4$ in this implementation), each sub-core is uniquely characterized by the set of row indices containing non-zero elements. For example, the first sub-core in the second column is denoted as $S_{[2,1]} = \{1, 2\}$, and the third sub-core in the third column is $S_{[3,3]} = \{0, 1, 2\}$.

Conflict Graph Construction: For N sub-cores within a single column window, we construct an undirected graph $G = (V, E)$.

- Vertex Set V : Each vertex $v_i \in V$ represents a unique sub-core.
- Edge Set E : An edge exists between two vertices if and only if their corresponding sub-cores share intersecting row position sets (i.e., $S_i \cap S_j \neq \emptyset$). As illustrated in

Figure 3, the 3rd and 4th sub-cores in the first column are connected by an edge because they both contain elements at row index 1.

Problem Transformation: The conflict-free constraint translates to the requirement that merged sub-cores must not be adjacent in graph G . Consequently, maximizing the number of sub-cores merged into a single dense block is equivalent to solving the Minimum Graph Coloring Problem (GCP) for G —identifying the minimum number of colors required such that no two adjacent vertices share the same color.

Offline Solution and Compression: Leveraging the static nature of model weights, we employ classical graph coloring algorithms to solve this problem offline. Sub-cores assigned the same color are compatible and are merged into a single dense structure. Finally, the merged weight matrix and the auxiliary metadata recording the mapping relationships are stored for efficient runtime retrieval.

5. Hardware Realization of SPARTA

Following the theoretical foundation of the conflict-free reorganization strategy presented in Section 4, we now propose its corresponding hardware realization: the SPARTA Architecture Extension. While offline reorganization efficiently aligns sparse weights into dense blocks, mapping these merged indices back to their corresponding activation values during runtime necessitates a flexible addressing mechanism. To support this new data layout without incurring runtime decoding penalties, we introduce a lightweight microarchitectural unit designed to execute address remapping in real-time. In this section, we first present the overall computation workflow under this framework. Subsequently, we describe the architectural design of the remapping interface and the required microarchitectural extensions, and finally, we detail the optimized pipeline mechanism designed to mask memory access latencies.

5.1. Unified Sparse-Dense Execution Workflow

Figure 4 contrasts the standard Tensor Core execution with our SPARTA-enhanced flow. In the standard scenario (shown in Figure 4a), processing a sparse workload (e.g., a workload of $M/K/N = 4/16/4$) is constrained by the rigid tile size of the hardware. This forces the allocation of four Tensor Cores to map disjoint sparse parameters, where the majority of MAC (Multiply-Accumulate) operations are performed on zero-padded elements, leading to severe resource fragmentation and energy wastage.

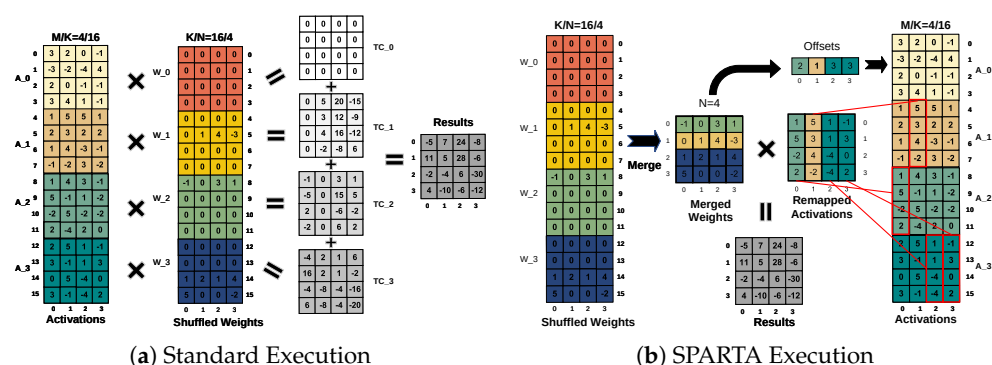


Figure 4. Comparative Workflow Analysis: (a) Standard vs. (b) SPARTA.

In contrast, Figure 4b demonstrates the optimized execution process using the SPARTA framework. Here, offline-shuffled weights are compacted into a dense Merged Weights matrix, effectively excising the zero-values. Paired with this are the lightweight Offsets, which serve as a hardware-managed lookup table. During the activation loading phase, these offsets drive the remapping logic to selectively gather discontinuous rows from the

activation memory (e.g., rows 2, 1, 3, 3 as indicated by the red mapping lines) to construct a dense Remapped Activations tile on-the-fly. This mechanism ensures perfect alignment between the compressed weights and inputs. Consequently, the entire computation is completed using only a single Tensor Core, effectively quadrupling the hardware utilization density while eliminating the overhead of accumulating partial sums from multiple TCs.

5.2. Microarchitectural Implementation

To materialize the algorithmic gains of the SPARTA strategy on resource-constrained edge GPUs, we propose a set of lightweight microarchitectural extensions. The primary design goal is to decouple the logical sparse indexing from the physical dense storage, enabling the Tensor Cores to consume compressed data with negligible execution penalties. In the following subsections, we first describe the hardware logic responsible for on-the-fly address translation, followed by the execution pipeline designed to maximize instruction-level parallelism.

5.2.1. Offset-Driven Address Translation Logic

In the execution pipeline of modern GPU architectures, memory access requests originate from the Load/Store (LD/ST) units within each Streaming Multiprocessor (SM). The address registers holding the target pointers reside in the thread-private register file. During a memory operation, the final physical address is computed by the address generation unit and subsequently transmitted via the on-chip high-bandwidth interconnection network (crossbar) to the Shared Memory I/O interface. To overcome the rigidity of standard addressing, we integrate a dedicated address remapping logic directly into this I/O circuitry, positioning it immediately preceding the physical memory banks. This architectural decision is critical; it allows the system to intercept incoming address signals and dynamically transform them based on pre-configured offset values. Consequently, this placement enables the hardware to support flexible, irregular access patterns natively before the actual memory read/write occurs, effectively decoupling the logical indexing from the physical data layout without incurring instruction-level latency.

As depicted in Figure 5, the SPARTA module is positioned immediately above the SMEM bank selection logic. Its core function is to redirect activation loading addresses based on the pre-calculated offsets. The hardware logic primarily comprises three components: 1. Bit Extender; 2. Shifter; 3. Full Adder. From a complexity perspective, this design introduces minimal logic depth. The address translation requires only simple arithmetic operations that can be completed within a single clock cycle, avoiding any critical path elongation. Unlike complex software decoding which requires iterative instructions, this hardware logic enables highly predictable, constant-time ($\mathcal{O}(1)$) address generation. Taking a standard 32-bit shared memory address as an example (where each bank has a width of 4 bytes), the lower 5 bits are utilized to index the 32 banks, while the upper 27 bits index the “rows” within each bank. The SPARTA mechanism utilizes the offsets—which record the relative positional shifts in the current Tensor Core’s weights derived from the reorganization phase—to modify the row index. By adding the offset to the row address component, the hardware locates the precise activation value corresponding to the compressed weight and loads it into the Tensor Core. Since a standard SM partition (SMP) typically involves 32 threads performing parallel memory operations, we implement 32 corresponding SPARTA circuits to handle concurrent requests with latency effectively masked by the pipeline.

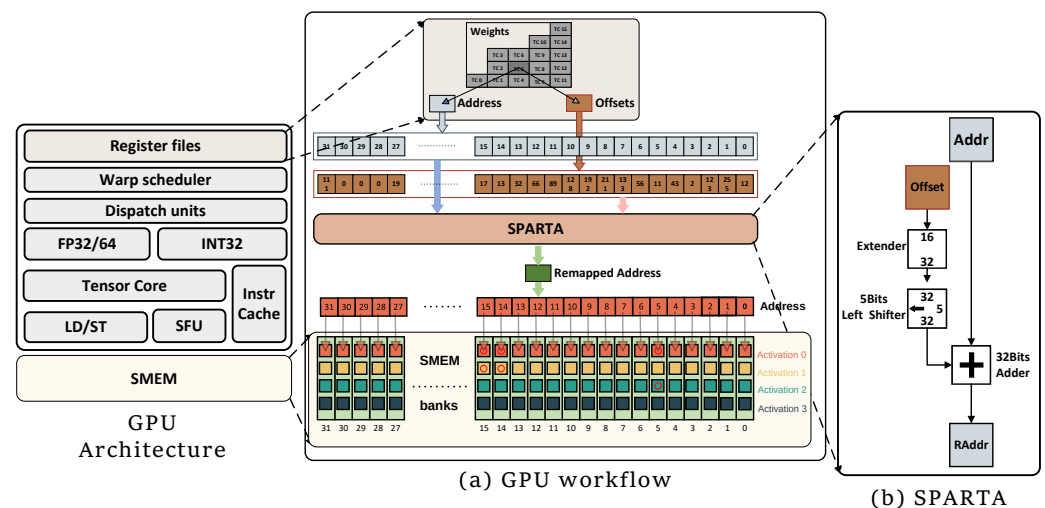


Figure 5. Microarchitectural Design of the SPARTA Address Remapping Interface.

5.2.2. Asynchronous Latency-Masking Pipeline

To maximize instruction-level parallelism and mask memory access latencies—a critical requirement for real-time threat analysis—we implement a specialized pipeline for the SPARTA execution flow. Computing a Tensor Core block involves four distinct stages: (1) GToSMEM: Load non-zero elements of the sparse matrix from Global Memory to Shared Memory. (2) GToREG: Load the auxiliary offset metadata from Global Memory to the Register File. (3) Id_dense: Load dense activation values from Global Memory to Shared Memory. (4) SMEM2TC: Load weights and remapped activations from Shared Memory to the Tensor Core for execution.

As illustrated in Figure 6, we adopt a double buffering mechanism, allocating two buffers (Buffer1 and Buffer2) in Shared Memory to parallelize these operations. In the initial phase, the sparse weights (A0) are loaded. Leveraging the asynchronous nature of the `cp.async` instruction, the loading of offsets (GToREG) and activation values (B0) occurs concurrently. Subsequently, the SPARTA circuit executes the remapped loading (SMEM2TC) for computation. Owing to double buffering, while the Tensor Core computes the current tile (A0, B0), the system simultaneously pre-fetches data for the next iteration (A1, B1), as shown in Iteration-2. This approach effectively overlaps sparse data loading with core computation, eliminating pipeline bubbles and ensuring the continuous throughput required for high-speed traffic analysis.

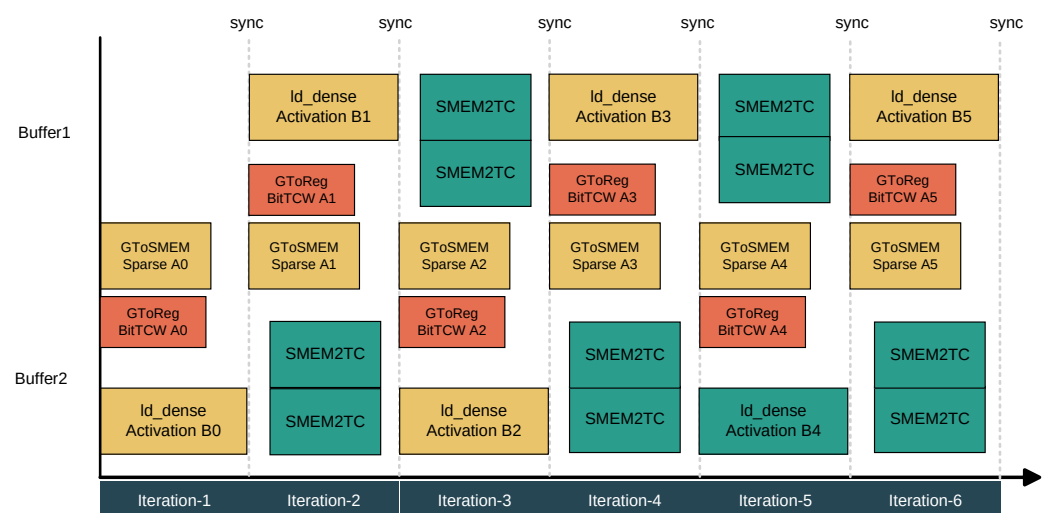


Figure 6. Asynchronous Latency-Masking Pipeline with Double Buffering Mechanism.

6. Evaluation

To validate the efficacy of SPARTA in enabling real-time, resource-efficient inference for CPS security applications, we conducted comprehensive evaluations. Our experiments focus on three key metrics critical for edge deployment: inference latency (speedup), energy efficiency, and hardware overhead.

6.1. Experimental Setup and Methodology

Simulation Platform: We used Accel-Sim [25], a cycle-accurate simulator based on GPGPU-Sim 4.0 [26] for our simulations. To implement the SPARTA framework, we modified and extended GPGPU-Sim 4.0 and Accel-Sim, configured to match the NVIDIA 3070 architecture. AccelWattch [27] was used for power evaluation.

Sparse Operator Baselines: We compared SPARTA with existing sparse operator acceleration works, including the original cuBLAS [28], Sputnik [29], and Flash-LLM [17]. Among these, cuBLAS is NVIDIA's official GEMM acceleration library, Sputnik is a sparse linear algebra library for deep learning that implements state-of-the-art SpMM operators based on SIMT cores, and Flash-LLM [17] is optimized for sparse acceleration with unstructured pruning.

Sparse Accelerator Baselines: We compared SPARTA with three sparse accelerators in terms of performance, power, and energy, including Tensor Core [22], Sparse Tensor Core [30], and Dual-Side [19]. Tensor Core [22] performs original GEMM operations for sparse workloads. Sparse Tensor Core [30] can accelerate workloads with fixed sparsity patterns (2:4, 4:8, etc.). Dual-Side [19] leverages the dual sparsity characteristics of both activation values and weight values.

Pruning Methods and Evaluation Models: To effectively evaluate the SPARTA framework, we performed unstructured pruning at different sparsity levels and 2:4 semi-structured pruning on the LLaMA [10] family (7B, 13B, 30B, and 65B) models using Wanda on the Wikitext2 [31] dataset, and evaluated the resulting sparse workloads (SpMM). It is important to note that while we utilize the Wikitext2 dataset for standardization, the computational characteristics of these workloads are isomorphic to real-world network traffic analysis. In AI-driven security, network logs (e.g., PCAP headers, application payloads) are tokenized into sequences identical in structure to natural language text. Therefore, the inference performance and sparsity benefits observed on these benchmarks are directly transferable to DDoS detection tasks where LLMs analyze serialized traffic data for anomaly patterns. All models and datasets mentioned above were obtained from the latest repositories on Huggingface.

6.2. Performance Evaluation

6.2.1. Comparison with Software-Optimized Kernels

Figure 7 compares the computational performance of the SPARTA algorithm with existing software-optimized SpMM kernels across different matrix scales and sparsity levels. The horizontal axis represents different workloads ($M/K/\text{Sparsity}$), where $M \times K$ denotes the dimensions of the weight matrix and Sparsity indicates the sparsity level. The experimental setup uses Batch_size = 8, meaning the sparse matrix multiplication operation involves multiplying an $M \times K$ matrix with a $K \times 8$ matrix, producing an $M \times 8$ output matrix. This experiment uses Flash-LLM [17] as the baseline (with performance normalized to 1) for comparative analysis. The experimental results demonstrate that Sputnik [29], which optimizes sparse matrix multiplication solely through SIMT cores, outperforms Flash-LLM [17] in certain workloads (5k/13k/80%, 5k/13k/90%, 13k/5k/90%, and 6k/17k/90%) under high sparsity conditions (80–90%) due to reduced decoding overhead. However, comprehensive analysis across all test cases reveals that the proposed SPARTA algorithm

achieves significant performance improvements compared to Flash-LLM [17], with an average speedup of $2.35\times$ and a maximum speedup of $5.05\times$ for specific workloads. SPARTA's performance advantages primarily stem from two optimization aspects: (1) an efficient tensor merging strategy that significantly reduces data access volume, and (2) a flexible shared memory (SMEM) access mechanism implemented through hardware extensions, further reducing computational overhead. Overall, compared to dense computation, SPARTA achieves an average reduction of 65% in tensor core computations.

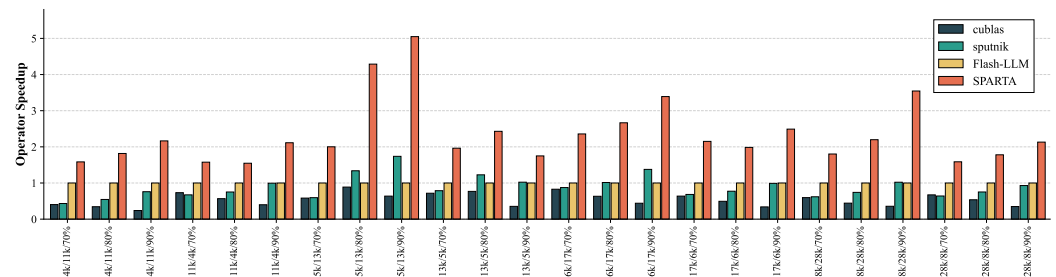


Figure 7. Speedup Comparison against Software Baselines.

6.2.2. Comparison with Hardware-Enhanced Kernels

Figure 8 compares SPARTA against hardware-accelerated baselines. To ensure a fair comparison, we reference the Dual-Side configuration, testing on 2:4 structured pruning workloads (50% sparsity) across four representative linear layers. Results indicate that SPARTA achieves an average speedup of $1.84\times$ over the standard Tensor Core implementation, approaching the theoretical $2\times$ limit of 50% sparsity. Crucially, SPARTA outperforms the Dual-Side method across all scenarios. This superiority stems from SPARTA's conflict-free merging strategy, which eliminates the severe bank conflicts that plague Dual-Side's outer-product approach, thereby ensuring deterministic high-bandwidth utilization.

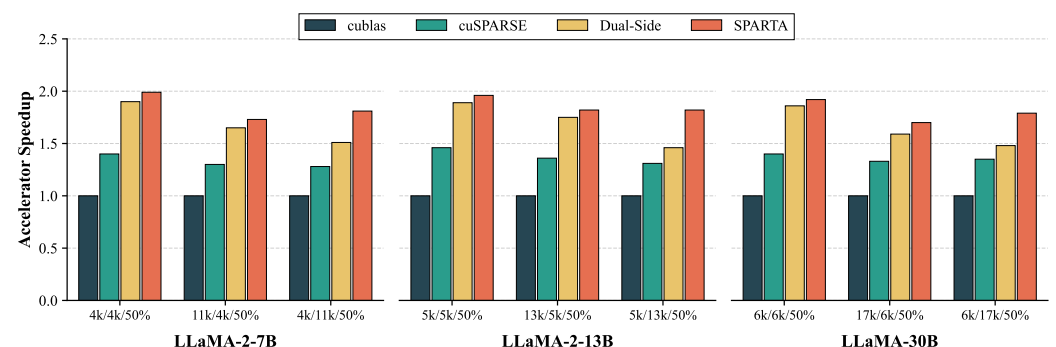


Figure 8. Speedup Comparison against Hardware Baselines.

6.3. Energy & Area Efficiency

6.3.1. Energy Consumption

Figure 9 presents the normalized energy consumption comparison results for different architectural designs, including both static energy and dynamic energy components. The dynamic energy analysis encompasses key components such as DRAM, L2 Cache, L1 Cache (including shared memory), register files, and processing units (Tensor Cores). It should be noted that L1 energy statistics include the combined energy consumption of L1 cache and shared memory. The experimental data shows that the SPARTA architecture exhibits the lowest overall energy overhead, an advantage primarily attributed to its innovative tensor merging mechanism and efficient SPARTA decoding circuit design. Specific analysis reveals that since sparse matrix multiplication (SpMM) is a memory-bound operation with high dependency on memory access, storage components dominate dynamic power consumption.

SPARTA significantly reduces system energy consumption by optimizing memory access patterns. Quantitative results demonstrate that under 75% sparsity conditions, compared to Flash-LLM [17], cuBLAS [28] (dense computation, 0% sparsity), and Sputnik [29], SPARTA achieves average energy reductions of $1.44\times$, $1.49\times$, and $4.29\times$, respectively.

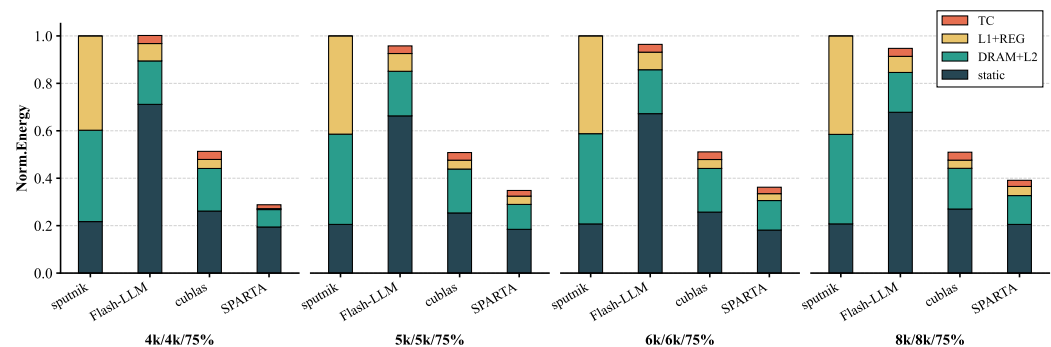


Figure 9. Normalized Energy Consumption Breakdown.

6.3.2. Area Overhead

To measure SPARTA's area overhead on GPU, we scaled SPARTA to Samsung's 8 nm process, the same as RTX 3070's fabrication process, to calculate tile area. We use the Ampere architecture as our baseline GPU, which includes 46 streaming multiprocessors (SMs), with each SM containing 4 tensor cores (184 total), each tensor core having four octets, and each octet containing four 8-bit element dot-product operators. Therefore, there are a total of $46 \times 4 \times 4 \times 4 \times 8 = 23,552$ 16-bit multiplication operators, along with 23,552 16-to-32-bit extenders and 11,776 32-bit full adders. Their areas are shown in Table 2. The die size of RTX 3070 [32] is 392 mm^2 . The 16-bit extenders and 32-bit full adders occupy only 0.129% and 0.078% of the total GPU area, respectively, which we consider to be very small overhead.

Table 2. Area Estimation and Comparison (8 nm Process).

Accelerators	Component	Area (mm^2)	Component Ratio	Total Ratio
Dual-Side	Float Point Adders	0.112	0.029%	1.461%
	Tile Index Aligner	0.351	0.090%	
	Accumulation Operand Collector	0.671	0.171%	
	Shared Accumulation Buffer	4.592	1.171%	
SPARTA	16-bit Extender Unit	0.507	0.129%	0.207%
	32-bit Adder Unit	0.306	0.078%	

6.4. End-to-End Performance Analysis

Figure 10 demonstrates the performance comparison results of various sparse matrix acceleration schemes on the LLaMA [10] family models (7B, 13B, and 30B parameter scales) under 75% unstructured pruning sparsity conditions, using Flash-LLM [17] as the baseline. The experimental data shows that the SPARTA scheme exhibits optimal acceleration performance. Specifically, compared to cuBLAS [28], Sputnik [29], Flash-LLM [17], and Dual-Side [19] baseline methods, SPARTA achieves average speedups of $4.5\times$, $4.8\times$, $3.0\times$, and $1.74\times$, respectively. This performance advantage primarily stems from two key design aspects of the SPARTA scheme: first, its weight compression algorithm maintains additional computational overhead at a low and fixed proportion; second, this design effectively mitigates performance degradation caused by increasing model parameter counts. Consequently, SPARTA demonstrates more significant acceleration advantages on larger-scale language models (such as 30B parameters).

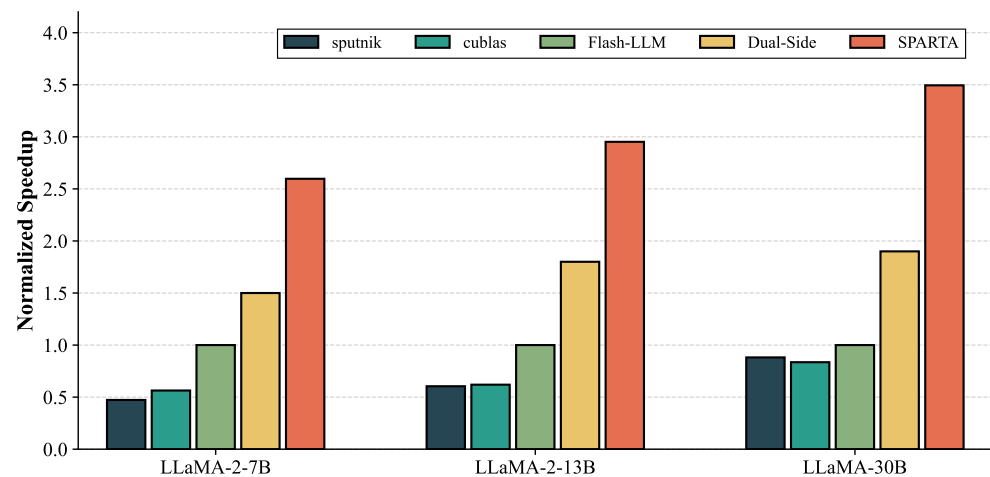


Figure 10. End-to-End Inference Speedup on LLaMA Models.

6.5. Ablation Study: Compression Efficiency

We further analyze the memory footprint reduction achieved by SPARTA's storage format. In Figure 11, the horizontal axis represents different model sizes and unstructured pruning sparsity levels. The vertical axis shows the memory overhead speedup ratio relative to Flash-LLM [17]. This experiment records the memory overhead speedup ratios of different acceleration methods at sparsity levels of 25%, 50%, 75%, and 95%. Overall, SPARTA achieves an average speedup ratio of $1.67\times$ in terms of memory overhead compared to Flash-LLM [17].

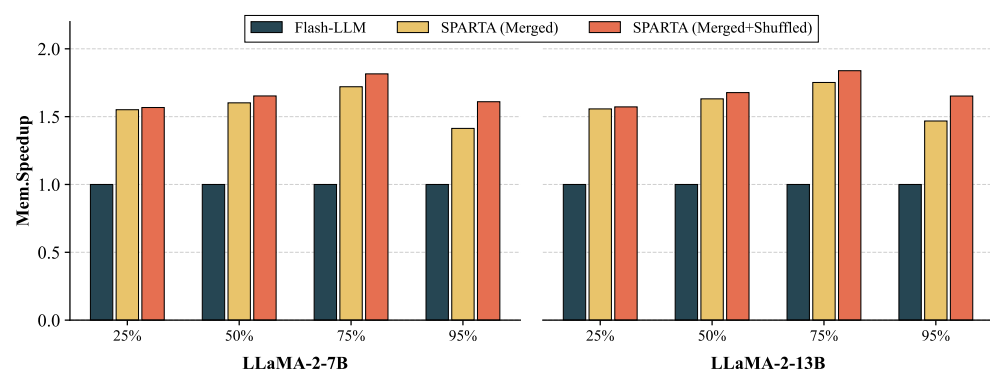


Figure 11. Memory Footprint Compression Ratio relative to Flash-LLM.

7. Conclusions

To address the critical bottleneck of deploying resource-intensive LLMs for real-time DDoS detection on resource-constrained CPS edge devices, this paper proposes **SPARTA** (Sparse Parallel Architecture for Real-Time Threat Analysis), a hardware–software co-design framework. Our research identifies that the irregular and conflict-prone memory access patterns of unstructured pruning significantly hinder the inference efficiency required for industrial security applications.

By addressing the fundamental mismatch between sparse patterns and Tensor Core granularity, we introduce a novel compilation strategy that models weight merging as a graph coloring problem. This transformation successfully converts irregular sparse operations into hardware-friendly dense formats, minimizing memory footprint while maximizing Tensor Core throughput. Complementing this, we designed the SPARTA microarchitectural extension, which optimizes Shared Memory I/O circuits to achieve low-latency, conflict-free data access.

The experimental results demonstrate that SPARTA delivers breakthrough efficiency compared to the state-of-the-art Flash-LLM solution, achieving an average speedup of $2.35\times$ (up to $5.05\times$) with negligible area cost. Building on this efficiency, SPARTA is envisioned as a lightweight microarchitectural extension (IP block) suitable for integration into next-generation hardware, particularly Smart Network Interface Cards (SmartNICs) in 5G Edge Servers. By embedding SPARTA as a dedicated sparse inference engine within the SmartNIC SoC, the system can perform near real-time semantic analysis of network telemetry logs at the edge. This design effectively offloads the heavy lifting of security reasoning from the host CPU. Ultimately, these findings indicate that SPARTA effectively bridges the gap between sophisticated AI algorithms and edge hardware limitations, providing a viable foundation for intelligent, log-based DDoS threat identification in resource-constrained Cyber-Physical Systems.

Author Contributions: Conceptualization, S.L. and L.Z.; methodology, S.L. and X.M.; software, S.L.; validation, S.L. and X.M.; formal analysis, S.L.; investigation, S.L.; resources, L.Z.; data curation, S.L.; writing—original draft preparation, S.L.; writing—review and editing, L.Z. and Y.L.; visualization, S.L.; supervision, L.Z.; project administration, L.Z.; funding acquisition, L.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Publicly available datasets were analyzed in this study. The LLaMA models and Wikitext-2 dataset can be found on HuggingFace (<https://huggingface.co/>) accessed on 13 June 2025. The simulation data and specific hardware configurations generated during this study are available from the corresponding author upon reasonable request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Li, J.; Lyu, L.; Liu, X.; Zhang, X.; Lyu, X. FLEAM: A Federated Learning Empowered Architecture to Mitigate DDoS in Industrial IoT. *IEEE Trans. Ind. Inform.* **2021**, *18*, 4059–4068. [CrossRef]
2. Hussain, B.; Du, Q.; Sun, B.; Han, Z. Deep Learning-based DDoS-attack Detection for Cyber-Physical System over 5G Network. *IEEE Trans. Ind. Inform.* **2020**, *17*, 860–870. [CrossRef]
3. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models are Few-Shot Learners. *Adv. Neural Inf. Process. Syst. (NeurIPS)* **2020**, *33*, 1877–1901.
4. Yang, A.; Xiao, B.; Wang, B.; Zhang, B.; Bian, C.; Yin, C.; Lv, C.; Pan, D.; Wang, D.; Yan, D. Baichuan 2: Open Large-scale Language Models. *arXiv* **2023**, arXiv:2309.10305. [CrossRef]
5. Chiang, W.-L.; Li, Z.; Lin, Z.; Sheng, Y.; Wu, Z.; Zhang, H.; Zheng, L.; Zhuang, S.; Zhuang, Y.; Gonzalez, J.E.; et al. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality. *LMSYS Blog* **2023**. Available online: <https://lmsys.org/blog/2023-03-30-vicuna/> (accessed on 27 January 2026).
6. Lin, X.; Xiong, G.; Gou, G.; Li, Z.; Shi, J.; Yu, J. ET-BERT: A contextualized datagram representation with pre-training transformers for encrypted traffic classification. In Proceedings of the ACM Web Conference 2022, Virtual, 25–29 April 2022; pp. 633–642.
7. NVIDIA. *NVIDIA Ada GPU Architecture Whitepaper*; Technical Report; NVIDIA Corporation: Santa Clara, CA, USA, 2023.
8. Song, Y.; Mi, Z.; Xie, H.; Chen, H. PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU. In Proceedings of the 30th Symposium on Operating Systems Principles (SOSP), Austin, TX, USA, 4–6 November 2024; pp. 590–606.
9. Sheng, Y.; Zheng, L.; Yuan, B.; Li, Z.; Ryabinin, M.; Chen, B.; Liang, P.; Ré, C.; Stoica, I.; Zhang, C. FlexGen: High-throughput Generative Inference of Large Language Models with a Single GPU. In Proceedings of the International Conference on Machine Learning (ICML), Honolulu, HI, USA, 23–29 July 2023; pp. 31094–31116.
10. Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. Llama: Open and Efficient Foundation Language Models. *arXiv* **2023**, arXiv:2302.13971. [CrossRef]
11. Frantar, E.; Alistarh, D. SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot. In Proceedings of the International Conference on Machine Learning (ICML), PMLR, Honolulu, HI, USA, 23–29 July 2023; pp. 10323–10337.
12. Sun, M.; Liu, Z.; Bair, A.; Kolter, J.Z. A Simple and Effective Pruning Approach for Large Language Models. In Proceedings of the Twelfth International Conference on Learning Representations (ICLR), Vienna, Austria, 7–11 May 2024.

13. Xu, P.; Shao, W.; Chen, M.; Tang, S.; Zhang, K.; Gao, P.; An, F.; Qiao, Y.; Luo, P. BESA: Pruning Large Language Models with Blockwise Parameter-Efficient Sparsity Allocation. In Proceedings of the Twelfth International Conference on Learning Representations (ICLR), Vienna, Austria, 7–11 May 2024.
14. Ma, X.; Fang, G.; Wang, X. LLM-Pruner: On the Structural Pruning of Large Language Models. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 10–16 December 2023; Volume 36.
15. Ashkboos, S.; Croci, M.L.; do Nascimento, M.G.; Hoefler, T.; Hensman, J. SliceGPT: Compress Large Language Models by Deleting Rows and Columns. In Proceedings of the Twelfth International Conference on Learning Representations (ICLR), Vienna, Austria, 7–11 May 2024.
16. Fan, R.; Wang, W.; Chu, X. DTC-SpMM: Bridging the Gap in Accelerating General Sparse Matrix Multiplication with Tensor Cores. In Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), La Jolla, CA, USA, 27 April–1 May 2024; pp. 253–267.
17. Xia, H.; Zheng, Z.; Li, Y.; Zhuang, D.; Zhou, Z.; Qiu, X.; Li, Y.; Lin, W.; Song, S.L. Flash-LLM: Enabling Cost-Effective and Highly-Efficient Large Generative Model Inference with Unstructured Sparsity. In Proceedings of the VLDB Endowment, Vancouver, BC, Canada, 28 August–1 September 2023; Volume 17, pp. 211–224.
18. Hong, K.; Dai, G.; Xu, J.; Mao, Q.; Li, X.; Liu, J.; Chen, K.; Dong, Y.; Wang, Y. FlashDecoding++: Faster Large Language Model Inference on GPUs. In Proceedings of the Machine Learning and Systems (MLSys), Santa Clara, CA, USA, 13–16 May 2024.
19. Wang, Y.; Zhang, C.; Xie, Z.; Guo, C.; Liu, Y.; Leng, J. Dual-Side Sparse Tensor Core. In Proceedings of the 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), Virtual, 14–18 June 2021; pp. 1083–1095.
20. Xia, M.; Zhong, Z.; Chen, D. Structured Pruning Learns Compact and Accurate Models. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL), Dublin, Ireland, 22–27 May 2022; pp. 1513–1528.
21. An, Y.; Zhao, X.; Yu, T.; Tang, M.; Wang, J. Fluctuation-Based Adaptive Structured Pruning for Large Language Models. In Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence, Vancouver, BC, Canada, 20–27 February 2024; Volume 38, pp. 10865–10873.
22. Markidis, S.; Der Chien, S.W.; Laure, E.; Peng, I.B.; Vetter, J.S. Nvidia tensor core programmability, performance & precision. In Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Vancouver, BC, Canada, 21–25 May 2018; pp. 522–531.
23. Choquette, J.; Gandhi, W.; Giroux, O.; Stam, N.; Krashinsky, R. NVIDIA A100 Tensor Core GPU: Performance and Innovation. *IEEE Micro* **2021**, *41*, 29–35. [[CrossRef](#)]
24. Liu, S.; Lindholm, J.E.; Siu, M.Y.; Coon, B.W.; Oberman, S.F. Operand Collector Architecture. U.S. Patent No. 7,834,881, 16 November 2010.
25. Khairy, M.; Shen, Z.; Aamodt, T.M.; Rogers, T.G. Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In Proceedings of the 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), Virtual, 30 May–3 June 2020; pp. 473–486.
26. Raihan, M.A.; Goli, N.; Aamodt, T.M. Modeling Deep Learning Accelerator Enabled GPUs. In Proceedings of the 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Madison, WI, USA, 24–26 March 2019; pp. 79–92.
27. Kandiah, V.; Peverelle, S.; Khairy, M.; Pan, J.; Manjunath, A.; Rogers, T.G.; Aamodt, T.M.; Hardavellas, N. AccelWattch: A Power Modeling Framework for Modern GPUs. In Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Virtual, 18–22 October 2021; pp. 738–753.
28. Fatica, M. CUDA Toolkit and Libraries. In Proceedings of the 2008 IEEE Hot Chips 20 Symposium (HCS), Stanford, CA, USA, 24–26 August 2008; pp. 1–22.
29. Gale, T.; Zaharia, M.; Young, C.; Elsen, E. Sparse GPU Kernels for Deep Learning. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Virtual, 9–19 November 2020.
30. Zhu, M.; Zhang, T.; Gu, Z.; Xie, Y. Sparse Tensor Core: Algorithm and Hardware Co-Design for Vector-Wise Sparse Neural Networks on Modern GPUs. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Columbus, OH, USA, 12–16 October 2019; pp. 359–371.
31. Merity, S.; Xiong, C.; Bradbury, J.; Socher, R. Pointer sentinel mixture models. *arXiv* **2016**, arXiv:1609.07843. [[CrossRef](#)]
32. NVIDIA. *NVIDIA AMPERE GA102 GPU Architecture*; Technical Report; NVIDIA: Santa Clara, CA, USA, 2021.

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