Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.082998

ARTICLE

An Architecture-Aware Hybrid CPU–GPU Approach for WEMA-Based Fast Pattern Matching in Network Intrusion Detection Systems

Adnan Hnaif[1,*], Hanadi Al-Shawabkah[2], Ayman Alqafaan[2] and Mohammad Alia[1]

[1]Department of Cybersecurity, Faculty of Science and Information Technology, Al-Zaytoonah University of Jordan, Amman, Jordan
[2]Department of Computer Science, Faculty of Science and Information Technology, Al-Zaytoonah University of Jordan, Amman, Jordan

*Corresponding Author: Adnan Hnaif. Email: adnan_hnaif@zuj.edu.jo

Received: 26 March 2026; Accepted: 15 May 2026; Published: 23 July 2026

ABSTRACT: Many fast pattern-matching mechanisms are used in NIDS (Network Intrusion Detection Systems) to filter higher volumes of network traffic prior to invoking expensive rule verification stages. This filtering phase in signature-based engines, such as Snort, needs to preserve exact matching semantics while being able to process at high throughput on commodity hardware. Here, we introduce a hybrid CPU–GPU architecture-aware framework for exact multi-pattern matching based on the Weighted Exact Matching Algorithm (WEMA). WEMA performs the most relevant matching based on deterministic ordered indexing of category units, which eliminates chaotic control flow (which occurs with automata learning) and also brings out more regular memory access. An extensive evaluation across CPU-only, GPU-only, and hybrid CPU–GPU execution models is performed to analyze how WEMA interacts with heterogeneous hardware architectures. Informed by these observations, we propose a hybrid design with CPU-based control-intensive tasks—rule parsing, index construction, batching, and rule verification—retained on the CPU while selective data-parallel payload scanning is off-loaded to the GPU. Experimental results show that CPU-only execution on a commodity multicore CPU and integrated GPU platform delivers stable performance across the entire range of payload sizes, while the hybrid model achieves modest but repeatable improvements for small and medium workloads. The architecture evaluated here offers a relatively small advantage for GPU-only execution. Given the very nature of NIDS alongside the results provided in this paper, it is clear that WEMA-based acceleration on NIDS highly depends on hardware features and workload size, while selective, architecture-aware GPU utilization is crucial to maintain deterministic run-time characteristics in security-critical applications.

KEYWORDS: NIDS; WEMA; exact string-matching algorithms; hybrid CPU–GPU architecture; heterogeneous computing

1 Introduction

The increased incidence, scale and sophistication of cyber-attacks have made network security a major industry in modern computing environments. In this context, Network Intrusion Detection Systems (NIDS) are the commonly deployed systems to monitor network traffic and detect suspicious or malicious behavior in real-time [1]. NIDS act as a crucial defensive mechanism in preventing data breaches, service downtime and illegal access into organizational networks by either installing preset signatures or using anomaly-based models.

Systems from this category include popular IDS systems such as Snort, and drive their detection capability largely from fast-pattern matching of packets or reassembled traffic streams against large collections

of rules [2]. As these systems are inline or near real time, pattern matching has to occur with low latency, lest application-layer security be diluted by the absence of timely inspection. Of the existing multi-pattern matching mechanisms, the Aho–Corasick algorithm has seen widespread use as it can match multiple signatures in linear time with respect to input size [3]. This feature makes it especially appealing to rule-based detection engines where the fast confirmation of known attack signatures is a priority.

But the constant growth of network bandwidth and traffic volume imposes increasing demands on CPU-only NIDS deployments [4]. Low latency data streams generated by high-speed links need to be processed continuously, and even with optimized multi-threaded CPU implementations it may be difficult to keep up with the required throughput. In these scenarios, inspection delays may be accumulated and lead to late or missed detections. In light of these limitations, this study investigates the Weighted Exact Matching Algorithm (WEMA), WEMA (Weighted Exact Matching Algorithm) is formally defined as a deterministic matching function:

$$M\left(P,S\right)=\left\{\left(i,s\_j\right)|\forall k\in \textit{selected offsets}\colon P\left[i+k\right]=s\_j\left[k\right]\right\}$$

where P is the payload and S is the pattern set. The selected offsets are determined using a weight function $W(c)$. a deterministic weight- and hash-based approach that improves adherence to heterogeneous CPU–GPU execution models while avoiding the control-flow irregularities caused by automaton-based methods.

And thus, heterogeneous architectures consisting of general-purpose CPUs along with Graphics Processing Units (GPUs) have been considered to speed up exact string matching in Network Intrusion Detection Systems (NIDS). For instance, by taking advantage of the data-parallel processing capabilities offered by GPUs, hybrid CPU–GPU systems promise to deliver high throughput as they process large traffic volumes [5]. These methods aim to push matching out of line without sacrificing the determinism and accuracy needed for signature-based detection.

In addition to raw performance improvements, hybrid architectures allow for architecture-aware workload partitioning. Assigning the right task to the hardware components that are best suited for them can limit memory contention, enhance cache behavior, and better utilize their available parallelism. Moreover, these systems provide flexibility and scalability; NIDS engines can adjust based on the rate of traffic and growing signature databases [6].

Optimized and hybrid approaches showed promising results recently in this area. For instance, the WEMA-based NIDS platform introduced in [7] achieves detection performance and scalability improvements; meanwhile, the header matching approach provided in [8] supplements payload inspection by reducing pattern matching expense in high-speed environments. Collectively, these studies underscore the growing need for algorithm–architecture co-design in the implementation of modern NIDS.

Motivated by this line of work, the current paper assesses a hybrid CPU–GPU framework for Snort-like NIDS engines that incorporates WEMA in the fast-pattern matching stage. This seeks to increase packet processing throughput while minimizing latency, with the goal of maintaining accuracy and deterministic patterns in detections.

The rest of this paper is structured as follows. Related work on fast-pattern matching and GPU acceleration for NIDS is reviewed in Section 2. Section 3: CPU and GPU Execution Characteristics Section 4 explains the methodology we propose; Section 5 sets out experimental results, and Section 6 concludes with a discussion of findings and implications.

This paper makes the following contributions: an architecture-aware evaluation of WEMA-based exact multi-pattern matching under CPU-only, GPU-only, and hybrid CPU–GPU execution models; a deterministic hybrid fast-pattern framework that integrates with Snort-like rule pipelines by retaining

parsing, indexing, and verification on the CPU while offloading only data-parallel payload scanning to the GPU; an empirical study on commodity hardware with an integrated GPU that clarifies when hybrid execution is beneficial and when GPU-only offloading is counterproductive; and practical design guidelines for batching and task partitioning that preserve low-latency detection while amortizing heterogeneous execution overheads.

The main contributions of this work are summarized as follows:

1. An architecture-aware evaluation of WEMA across CPU-only, GPU-only, and hybrid execution.
2. A deterministic hybrid framework integrating WEMA into Snort-like pipelines.
3. Empirical evaluation on commodity hardware highlighting workload-dependent performance.
4. Practical design guidelines for CPU–GPU workload partitioning.

2 Literature Review

2.1 Fast-Pattern Matching in Signature-Based NIDS

Examples of existing NIDS are signature-based, which use exact multi-pattern string matching techniques to scan network traffic for known signatures corresponding to attacks. In Snort-like engines, content options are commonly marked out as fast patterns so that the candidate rule set can be reduced before the expensive whole rule check. Automaton-based techniques, in particular the Aho–Corasick algorithm, are commonly used for this purpose as they allow matching of many patterns simultaneously and have linear time complexity with respect to the payload size. However, while advantageous in many respects, such approaches may result in non-uniform memory performance and frequent branching, both of which can degrade performance on modern processor architectures [9] and reduce scalability with large rule sets.

Besides traditional signature matching, recent network intrusion detection systems (NIDS) research is focusing more and more on machine learning–based detection approaches. Standard benchmark datasets have been used to evaluate both supervised and unsupervised models, all of which show that deep neural network approaches outperform traditional methods in terms of detection accuracy and robustness. Unsupervised models, in particular, have demonstrated their ability to uncover never-before-seen or emerging attacks. These outcomes have shown promising positive effectiveness but indicate a heavy dependence on challenging computational detection mechanisms, demonstrating the necessity of architecture-aware acceleration in active NIDS deployments [10].

To better highlight the differences between commonly used multi-pattern matching algorithms in NIDS, a comparative summary is provided in Table 1. This comparison focuses on computational complexity, memory requirements, and suitability for GPU acceleration, which are critical factors in high-performance intrusion detection systems.

Table 1: Comparison of pattern matching algorithms used in NIDS.

Algorithm	Time Complexity	Space Complexity	GPU Suitability
Aho–Corasick algorithm	$O(n+m+z)$	High	Medium
Rabin–Karp	$O(n+m)$	Low	High
WEMA	$O(n \cdot k)$	Medium	High

As shown in Table 1, WEMA offers a balance between computational efficiency and regular memory access patterns, making it more suitable for hybrid CPU–GPU execution compared to traditional automaton-based approaches.

On the algorithmic side, well-matching implementations optimized for multi-core systems have been shown to deliver tremendous throughput improvements over baseline Aho–Corasick designs—highlighting the critical nature of efficiency in high-speed inspection pipelines. Induced by the extensive parallelism offered on GPUs, many approaches investigated offloading computation intensive patterns matching phases on GPU kernels. Standard methods split the payload data into chunks, which are collocated and operated on in parallel by threads or thread blocks. In practice, however, effective performance is heavily impacted by kernel launch overhead, host–device data transfer costs and how well-suited data structures are to GPU execution. Pointer-heavy or irregular memory layouts in particular may lead to poor memory coalescing and increased thread divergence, thus limiting the anticipated speed-ups [11]. These considerations, in turn, set up for hybrid CPU–GPU designs where control-heavy latency-sensitive tasks remain on the CPU and GPUs are co-allocated to data-parallel operations with very high-performance throughput that can be leveraged without losing responsiveness.

2.2 GPU Acceleration for Payload Inspection

The SIMD-style execution of modern GPUs should be thoroughly examined depending on the architectural limits as well as the properties of matching algorithms for effective GPU-based acceleration in payload inspection for NIDSs. If pattern-matching engines, like Aho–Corasick algorithm, are redesigned specifically for GPU execution—using memory data structures tailored and optimized—we can observe a significant throughput performance gain over CPU adaptations [12]. But these improvements are not certain. More realistic performance can be very sensitive to e.g., host–device data transfer overhead, memory access coalescing or thread divergence during kernel execution.

Simultaneously, recent surveys of machine learning–based NIDS approaches have highlighted the opportunities as well as the limitations of real-time attack detection. More specifically, limitations in feature extraction complexity as well as inference latency continue to hinder deployability at high network speeds, thus further motivating architecture-aware acceleration strategies within hybrid CPU–GPU systems [13].

Several studies have proposed alternative string-matching techniques that can run on GPUs, and even GPU-accelerated variants of the Rabin–Karp algorithm designed to make up for some of the automaton-based matching restrictions. These techniques aim at scalability in the 1000s of concurrent threads, minimize validation overhead with respect to false-positives and perform hashing operations in a way that can better leverage shared memory usage. As a consequence, they have shown significant performance improvements on real-time NIDS workloads [14].

In addition to algorithmic and architectural optimizations, system-level improvements to intrusion detection systems have also been reported. For instance, the work done within local research activity [15] including evolutionary patterns designed a method to improve real-world detection efficiency within network-based services, consequently combining any low-level performance-boosting techniques with capabilities tailored for application-specific protections.

2.3 Hash-Based and Probabilistic Filtering

Some NIDS designs use hash-based prefilters or probabilistic data structures like Bloom filters to reduce the matching overhead even further. These methods can significantly reduce the number of candidates that need to be checked exactly, thus reducing average inspection costs. But Bloom filters do have a built-in false positive rate leading benign traffic to be flagged for investigation. Consequently, more verification work is added and the resource utilization also makes it hard to estimate under high traffic loads. In environments that need to be security-sensitive and quite deterministic in terms of predictable behavior, such uncertainty

is not always a desirable outcome either. This has driven more work on deterministic and hybrid approaches that target predictable throughput along with measurable efficiency benefits [16].

2.4 Weighted Exact Matching Algorithm (WEMA)

Overview Weighted Exact Matching Algorithm (WEMA) is a deterministic string-matching algorithm based on weighted indexing, which highlights informative characters by adopting weights to accelerate the process of discarding unseen input segments and cut out useless comparison operations. Instead of going through an automaton like for example in the Aho-Corash WEMA uses a precomputed weight matrix containing weights for every character, this way direct matching decisions can be made. Designing in this manner creates a more regular control flow and memory access patterns, allowing throughput improvements for exact matching workloads. Consequently, WEMA is suited for use as a baseline or comparative technique in NIDS environments where the cost of indexing an attack flow must be carefully balanced with matching efficiency and predictable performance. First, general challenges of exact and multi-pattern matching on parallel architectures have been explored in previous work, including works focused in GPU-based implementations of automaton-driven approaches [17].

Hybrid CPU–GPU techniques have been widely studied to accelerate pattern matching in NIDS. As a concrete example, the length-bounded hybrid CPU/GPU algorithm in [18] combines prefiltering (in the CPU domain) with full pattern matching in the GPU at higher throughput rates by balancing computation on both architectures while controlling overhead for data transfers. Extending this idea, follow up studies focused on how to optimize task allocation strategies and decrease host–device communication overheads to achieve higher levels of parallelism exploitation.

The WEMA model is hybrid in design, and follows similar principles as the weightings, but differs due to a weighted pattern matching process and the explicit exploration of the interactions within the architecture. The idea is to optimize the division of labor between CPU and GPU, while considering characteristics of the hardware for better efficiency both on integrated and discrete GPU based platforms. All in all, these studies highlight the role of architecture-aware hybrid designs to maintain high throughput with low latency under real-time NIDS workloads.

Recent advancements in exact string-matching methods led to significant further performance gains, leveraging scalable implementations for multiprocessing architectures and striking a careful tradeoff between hashing/indexing overheads and traversal costs. As part of hybrid detection frameworks, such improvements have been demonstrated to yield significant throughput increase. Among them, research on optimized multi-pattern matching for GPU highlights how reorganizing matching structures and memory access patterns can significantly improve efficiency by minimizing irregular accesses and making full use of the vast parallelism offered by modern GPUs. In line with this, these results further highlight the importance for architecture-aware optimizations when deploying high-performance pattern matching within NIDS workloads [19].

2.5 Research Gap

Although extensive work has been done on both CPU- and GPU-based acceleration of NIDS payload inspection, an important gap persists. Specifically, it lacks efficient deterministic fast-pattern matching frameworks that seamlessly integrate into the Snort-like rule pipeline, preserve exact matching semantics without introducing any probabilistic false positives, and can exploit heterogeneous CPU–GPU platforms in an architecture-relevant way.

Due to irregular execution and synchronization overhead, purely computational solutions based on GPUs struggle during preprocessing and control-intensive stages of computation. On the other hand,

CPU-only designs are limited by finite core counts and memory bandwidth, leading to dependence on "scalability" as traffic volumes scale up. To overcome these challenges, WEMA presents a deterministic and architecture-aware matching scheme that facilitates a clear separation of responsibilities between processing units. In real-world applications, control-dominated tasks (such as rule parsing, stream and flow management, and index construction) are done on the CPU, but the GPU is reserved strictly for small- and large-scale data-parallel payload scanning.

3 Proposed Architecture-Aware Hybrid CPU–GPU Methodology Based on WEMA

Here we present the proposed architecture-aware hybrid CPU–GPU approach with WEMA, to accelerate the fast-pattern matching stage in Snort-like Network Intrusion Detection Systems (NIDS). It is designed to not interact with the standard Snort detection pipeline, but rather carry out signal processing in a heterogeneous computing device agnostically and maximally deterministically and hardware aware.

3.1 System Model and Design Assumptions

The architecture is applied towards a passive, Snort-like NIDS deployment where traffic on the network are mirrored to detection engine through either the SPAN or TAP interface. Decoding of packets takes place on the CPU, followed by stream re-assembly to generate a continuous buffer of payloads ready for inspection. Developments here in the design led to the assumption that exact pattern matching will have to remain completely deterministic, with zero tolerance for false positives and false negatives—a hard requirement in security-centric operational contexts.

To limit the overhead associated with GPU processing and maintain low detection latency, payload data are aggregated into small to medium-sized batches over short time intervals. This batching strategy allows for partial amortization of GPU execution costs while still honoring the real-time constraints of intrusion detection so that acceleration does not come at the cost of timely response.

3.2 CPU-Based Rule Processing and WEMA Index Construction

The first stage of the pipeline is all CPU as it is control-intensive (preprocessing). Snort rules are parsed and the exact strings, or content, is extracted to use as fast patterns. These patterns are then provided at the input of WEMA where each character receives a weight based on its significance in distinguishing it from other. To formally define the weighting mechanism, the character weight is computed as follows:

The weight of a character c is computed as:

$$W(c) = 1/freq(c)$$

where $freq(c)$ is the frequency of character c in the pattern set.

Characters in the complete pattern set. Instead of traversing automaton, WEMA builds up an easy to access weighted index matrix that directly points to candidate matching positions without any processes of state transformations or failure links. This yields significant reductions in branching and irregular memory access which makes it suitable to modern CPU architectures as well as heterogeneous execution environments.

CPU is responsible for building the WEMA index and associating pattern identifiers with detection rules. Once a candidate match is reported, this mapping allows the rules to check themselves efficiently. Since index building is done only once and reused for subsequent payload batches, it amortizes the preprocessing cost over time (which is beneficial in scenarios like traffic monitoring that runs continuously). The WEMA-based fast-pattern scanning process is formalized in Algorithm 1.

Algorithm 1: WEMA-based fast-pattern scanning

Input:
P: payload buffer
S: pattern set
W: weight matrix
I: index structure

Output:
M: set of match events (pattern_id, offset)

```
M ← ∅
for each candidate position i ∈ P do
    C ← SelectWeightedChecks(I, i)
    matched ← true
    for each (char, offset) ∈ C do
        if P [i + offset] ≠ char then
            matched ← false
            break
        end if
    end for
        if matched = true then
            if ExactVerify(P, S, i) = true then
                M ← M ∪ {(pattern_id, i)}
            end if
        end if
  end for
return M
```

Time Complexity: O (n · k), where n is the payload size and k is the number of weighted checks.

Select Weighted Checks: selects the most discriminative character positions.

Exact Verify: performs full deterministic pattern verification.

Involvement of CPU in the first phase of proposed framework along with deployment of WEMA index as shown by Fig. 1. Snort rules are parsed for the fast-pattern content used during intrusion detection. Next, these patterns are utilized to create a weighted WEMA index, facilitating direct access to candidate matching positions during the scanning step. Pattern identifiers are mapped to their respective detection rules so that once there is a match, the rule can be verified efficiently.

Running these control-heavy tasks on the CPU decreases overall system complexity while guaranteeing that pattern data is effectively organized in advance of GPU offload. Such a design is then cost-effective in terms of preparing the needed data structures for high-throughput parallel scanning without incurring too much preprocessing overhead on the accelerator.

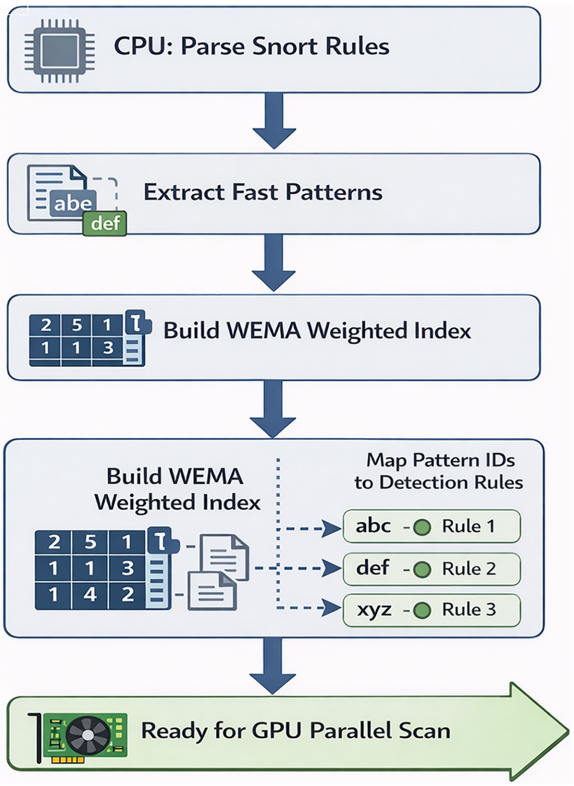

Figure 1: CPU-based rule processing and WEMA index construction pipeline prior to GPU offloading in the proposed hybrid NIDS architecture.

3.3 GPU-Based Fast-Pattern Scanning Using WEMA

Following index construction, the payload scanning stage is selectively offloaded to the GPU. The CPU assembles batches of reassembled payload segments and transfers them to GPU memory together with the precomputed WEMA index structures. On the GPU, each thread independently processes a payload segment using WEMA's weighted matching procedure. Because WEMA avoids automaton traversal and instead relies on direct indexed comparisons, GPU execution exhibits more regular memory access patterns and reduced control-flow divergence compared to tries-based approaches.

Each thread deterministically evaluates candidate positions and records match events containing the corresponding pattern identifier and absolute payload offset. Upon completion of the scanning phase, the GPU returns compact match buffers to the CPU for subsequent processing. No rule verification or control-intensive logic is executed on the GPU, ensuring that the accelerator is dedicated exclusively to high-throughput, data-parallel matching operations.

The proposed hybrid CPU–GPU methodology, illustrated in Fig. 2, outlines the end-to-end processing workflow of the system. Network traffic is first mirrored to the CPU, where packet capture, rule parsing, and WEMA index construction are performed. Reassembled payloads are subsequently grouped into batches and transferred to the GPU for data-parallel fast-pattern matching. After scanning completes, the GPU returns compact match results and continuation information to the CPU. The CPU then performs rule verification and generates the corresponding alerts and logs, ensuring accurate intrusion detection while maintaining efficient use of heterogeneous computing resources.

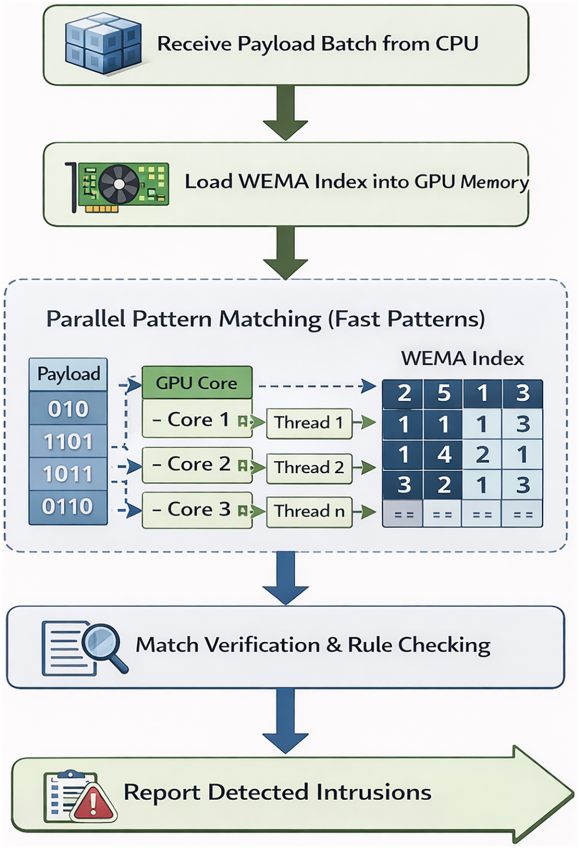

Figure 2: Detailed workflow of the proposed hybrid CPU–GPU methodology for fast-pattern matching in Snort-like NIDS.

3.4 CPU-Based Rule Verification and Alert Generation

All rule verification is performed on the CPU after fast-pattern detection. A match reported by WEMA identifies a potential rule candidate but does not, by itself, trigger an alert. Each candidate is then validated by the CPU against the remaining rule conditions (including further content checks, positional constraints (offset, depth, distance, within), header fields for a protocol and complex expressions such as PCRE).

As the verification is performed over a subset of rules discovered through matches to WEMA, this reduces the computational effort substantially under high traffic loads. Simultaneously, this design allows for Snort's original detection semantics to be preserved and full compatibility with existing rule sets to be achieved while allowing fully deterministic behavior throughout the detection pipeline.

3.5 Correctness and Determinism Guarantees

Before we describe the implementation pipeline, allow us to explain the rationale behind that design choice. Correctness here is made sure by enforcing completely deterministic execution along the detection pipeline. WEMA only does exact string matching with probabilistic filtering mechanisms not needed, guarantees no false positives would happen at the fast-pattern stage. Moreover, payload segmentation is agnostic to overlapping regions, as WEMA inherently evaluates candidate match positions in isolation and does not need to rely on cross-segment state.

The strong separation of roles between CPU and GPU also helps reinforces correctness. The totality of all control flow, rule semantics and verification logic remain on the CPU, with the GPU abstractly limited

to data-parallel matching ops. Such a design circumvents semantic incompatibilities and ensures that all compatible matches are correctly identified, regardless of execution model or payload size.

3.6 Architectural Rationale

The new hybrid architecture takes advantage of the complementary strengths of CPUs and GPUs to offer fast-pattern matching that is both efficient and scalable. Where control flow and complex logic dominate preprocessing, index construction, rule management and verification, it is the CPU that rules supreme. Unlike traditional payload scanning, the computationally intensive portions of WEMA are possible to accelerate using our GPUs due to regular and predictable memory access patterns. This architecture–aware division of labor achieves high throughput exact pattern matching on commodity hardware while maintaining full detection accuracy and deterministic behavior.

4 Experimental Evaluation

To validate the proposed hybrid CPU–GPU methodology implementation and compare the results with a reference sequential execution, several experiments were carried out on a heterogeneous platform to measure execution time, throughput, and architecture behavior when processing different payload sizes and executing metrics.

4.1 Hardware and Setup

We performed experiments using a commodity monitoring platform, equipped with Intel Core i5-9300H CPU (4 cores, 8 threads running at 2.4 GHz), 16 GB RAM and Intel UHD Graphics 630 integrated GPU. The paper evaluated three different execution models: CPU-only, GPU-only and hybrid CPU–GPU. The workloads consisted of batches containing 10, 20 or 50 payloads that selected to mimic specific traffic rates and buffer size periods typically occurring in passive NIDS deployments.

4.2 Overall Performance Trends

The CPU-only execution remained stable and consistent regardless of the payload size evaluated. On the other hand, GPU-only execution was limited by integrated GPU capability and included kernel launch cost as well as host–device data transfers, both of which rendered marginal gains in performance and sometimes reduced execution time. The hybrid CPU-GPU architecture showed significant performance improvements on small and medium payload batches. But with larger workloads, the CPU-side control overhead and memory bandwidth limitations dominated, diminishing the relative benefit of offloading to a GPU.

4.3 Execution Time Comparison

Execution times for batch sizes of 1, 10 and 100 MB are summarized in Table 2. Its hybrid CPU–GPU model performs significantly better with respect to execution time compared with both the traditional CPU-only configuration and GPU-based configurations for small to medium batches. For 100 MB batch sizes, there is negligible performance gain, with CPU side processing and memory bandwidth limitations taking precedence. These results highlight the impact of batch size in hybrid execution performance.

In terms of effective throughput, the hybrid CPU–GPU configuration achieves approximately 1.33, 1.43, and 1.32 MB/s for 1, 10, and 100 MB payload batches, respectively, compared to 1.25, 1.33, and 1.28 MB/s under CPU-only execution.

Table 2: Execution time comparison (seconds) for different execution models.

Payload Batch Size (MB)	CPU-only	GPU-only	Hybrid
1	0.8	0.9	0.75
10	7.5	8.2	7.0
100	78	82	76

4.4 Impact of CPU Multi-Threading

In general, adding 2 CPU threads (from 4 to 6) improved execution time significantly as we execute more batches of medium and large payloads. However, beyond six threads, performance gains began to saturate when memory bandwidth limitations and synchronization overheads came into play. These observations stress the impact of CPU-side optimization and thread scaling in NIDS workloads deployed on commodity hardware platforms.

4.5 Comparison with GPU Filtering Approaches

While discrete GPUs can yield high throughput performance with hash-based structures such as Bloom filter–based approaches, they usually rely on specialized hardware and are susceptible to false positives that must be separately verified. The proposed hybrid CPU–GPU method, in contrast, maintains deterministic exact matching while enabling greater architectural flexibility. Table 3 demonstrates that performance gains on integrated GPUs are quite modest, but much higher improvements can be observed on discrete GPUs. These findings point out the fundamental trade-off between throughput for hardware independence and accuracy for accelerated NIDS solutions.

Table 3: Conceptual and qualitative comparison of execution models under different hardware assumptions.

Approach	Hardware	Performance Outcome	Main Limitation
GPU hash/Bloom prefilter	Discrete GPU	High throughput in favorable cases	Hardware-specific; may add false positives
CPU-only (this work)	Multicore CPU	Stable across batch sizes	Limited by CPU cores/bandwidth
GPU-only (this work)	Integrated GPU	No meaningful acceleration	Limited GPU resources; overheads
Hybrid CPU–GPU (this work)	CPU + Integrated GPU	Minor gains for small/medium batches	Gains depend on GPU capability

4.6 Contextual Comparison

The performance results of the proposed approach are also compared with existing high-level GPU-based and hybrid pattern-matching techniques in Table 4. Other studies that combine vendor-specific discrete GPUs with Wu–Manber or Aho–Corasick algorithms report 4× speedups over CPU-only execution Hybrid designs like HPMA have been reported to improve performance compared to CPU-only and GPU-only execution in previous studies [19].

Table 4: Conceptual comparison with representative GPU filtering approaches.

Approach	Hardware	Payload Batch Size	Throughput/ Speedup	Main Observation
GPU-only [19]	Discrete GPU	Variable	0	Massively parallel GPU improves multi-pattern matching throughput
CPU-only (This work)	Intel i5-9300H	1/10/100 MB	0.8/7.5/78 s	Stable and predictable; CPU bottleneck dominates for large payloads
GPU-only (This work)	Integrated GPU	1/10/100 MB	0.9/8.2/82 s	Limited acceleration due to integrated GPU, kernel and transfer overheads
Hybrid CPU–GPU (This work)	CPU + Integrated GPU	1/10/100 MB	0.75/7.0/76 s	Modest gains for small/medium batches; performance depends on GPU capability

Conversely, our evaluation indicates that the proposed hybrid approach offers marginal performance gains for small and medium payload sizes on integrated GPU platforms. This aligns with previous results showing the effectiveness of hybrid execution is heavily dependent on GPU capability and batch size.

The hybrid CPU–GPU approach provides modest performance improvements for small and medium payload batches, while gains are marginal for large batches due to CPU-side control and memory overheads, as illustrated in Fig. 3.

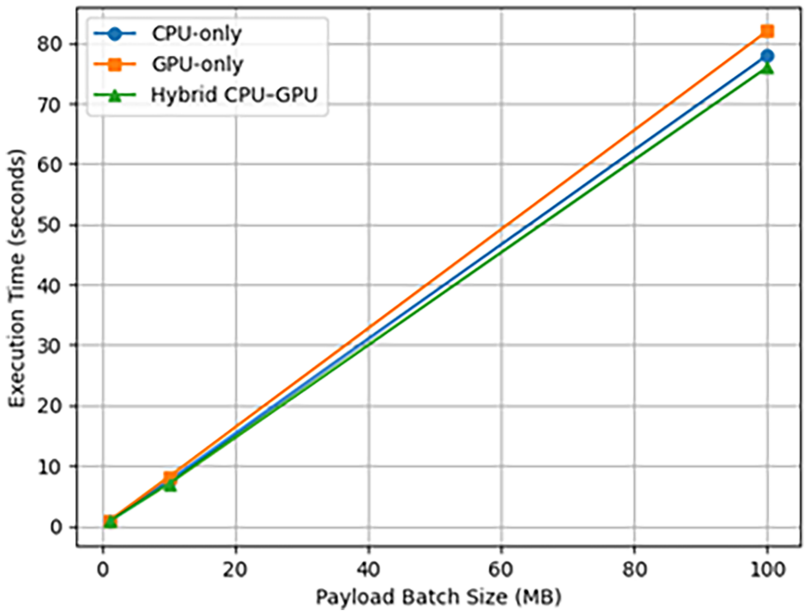

Figure 3: Execution time had been compared for CPU-only, GPU-only, and hybrid CPU–GPU models across payload batch sizes.

4.7 Complexity and Architectural Implications

Since the matching operation is deterministic, the execution time of exact pattern matching over WEMA operates in a way such that it grows linearly with input size. Data preprocessing—which includes WEMA index construction, pattern weight computation, and mapping pattern identifiers to their associated rules—are determined by the number of patterns, lengths of the patterns, and size of the alphabet. Under optimal conditions, these attributes promote predictable and repeatable execution semantics.

The total execution time can be modeled as:

$$T\_total = T\_CPU\_pre + T\_transfer + T\_GPU\_match + T\_CPU\_verify$$

Performance, however, is heavily determined by the underlying hardware architecture in practical deployments. In both CPUs and GPUs, multiple architectural considerations dictate how much throughput can be achieved.

Memory bandwidth has a major influence on this type of algorithm, since the speed with which payload data and index structures can be fetched directly influences matching throughput. On integrated GPUs, restricted shared memory bandwidth may be a limiting factor, especially when consuming large payload batches or host–device data transfers can contend with CPU accessing its memory.

Also, cache behavior matters a lot for performance. Because CPU and GPU caches mitigate the latency of memory accesses, poor data locality or irregular access patterns can lead to cache misses. While WEMA yields improvements in regularity over automaton traversal, the weight of an index may put stress on hierarchical caches where pattern sets or payloads are large.

Additionally, thread divergence on GPUs further hurts efficiency. By the conditional checks intrinsic to weighted matching, threads in a warp may be diverged and execute on different paths, resulting in a low effective degree of parallelism, i.e., each GPU execution unit is likely to be underused.

At last, Host–device transfer overhead which creates extra latency during the exchange of payload batches and results buffers between CPU and GPU. On integrated GPU platforms, these transfers use the same system memory bus and therefore add contention, reducing effective throughput for larger workloads.

Finally, CPU control-intensive operations such as preprocessing, WEMA index construction, rule verification and state management may account for a significant proportion of shot execution time. As these tasks become the dominant bottleneck, and particularly for large batch sizes where CPU-side processing scaling is less favorable, so does the relative advantage of offloading on-the-fly to GPU.

With consolidated use of integrated GPUs, hybrid CPU–GPU combinations yield measurable performance gains only for small to medium workloads—only dynamical memory management strategies can deliver better scaling with workload size. Its effectiveness is much higher on discrete GPUs, with higher memory bandwidths, more execution units and the dedicated device memory. In these environments, the GPU treats large payload batches with parallelism while the control-intensive tasks are delegated to the CPU thereby yielding higher throughput without sacrificing determinism or detection correctness.

5 Discussion

Herein, we present the experimental results of our proposed hybrid CPU–GPU framework based on WEMA and discuss its performance characteristics, architectural implications and relevance to security-critical applications with a particular focus on Network Intrusion Detection Systems (NIDS).

We find that the performance of WEMA is highly sensitive to the size of workload and execution architecture. While many may generalize that GPU, acceleration does yield better performance, it is found

that with small to medium workloads and depending on the use case, a CPU-only execution can beat GPU-only execution. It is mainly attributed to the kernel launch overhead and host–device data transfer latency, as well as the constrained computational and memory resources for integrated GPU architectures.

In addition, the scalability of the proposed framework depends on several factors, including workload size, rule set complexity, and the computational capabilities of the underlying hardware. As network speeds and traffic volumes increase, efficient workload distribution between CPU and GPU becomes critical to maintaining real-time performance.

The hybrid CPU–GPU execution model, however, shows more consistent and effective performance over a range of dataset sizes. The proposed hybrid approach overcomes important bottlenecks such as control-flow divergence and irregular memory access patterns by moving preprocessing and control-flow–intensive tasks to the CPU side while offloading the computation-intensive weighted matching phase of WEMA to the GPU. This is more visible on automaton based and exact string-matching methods in summary, as the dataset size grows the hybrid model continues to significantly outperform both CPU-only execution and GPU-only execution, as can be seen in Fig. 3, confirming once again that heterogeneous computing is advantageous once overhead costs have been amortized.

Importantly, the hybrid framework retains the deterministic exact matching properties of WEMA and avoids probabilistic false positives—a non-negotiable for any sensitive deployments. The findings also suggest that there is no best execution mode in all circumstances. CPU-only execution is still preferable for small workloads, while hybrid execution offers significant advantages for larger inputs.

Building upon this observation is the motivation for using adaptive execution strategies that choose the best-suited execution model depending on workload characteristics and available hardware.

On the side of algorithms/domain knowledge, WEMA is as strongly compatible with heterogeneous architectures. Its deterministic, hash-oriented design avoids several state transitions and enables relatively access patterns in memory that allow for greater parallel efficiency with respect to CPUs and GPUs alike. While this pre-processing adds some overhead, WEMA is a particularly good fit for high-throughput and repeat-query scenarios, which are common in real-world NIDS deployments.

In conclusion, this research proves that architecture-aware optimization is crucial to WEMA based exact string matching. Contrary to what could be expected from standard uniform GPU offloading, the results show that processor performance improvement is directly tied to the additional execution model and workload partitioning. We present a safe, scalable, and deterministic solution for high throughput pattern matching across modern heterogeneous compute engines with practical hybrid framework.

6 Conclusion and Future Work

We presented an architecture-aware hybrid CPU–GPU framework for exact pattern matching in Snort-like Network Intrusion Detection Systems (NIDS) based on the Weighted Exact Matching Algorithm (WEMA). Unlike traditional automaton-based approaches, WEMA employs deterministic weighted indexing to enable direct matching decisions, eliminating the need to navigate complex state transitions and irregular memory access patterns. These characteristics make WEMA particularly well suited for heterogeneous CPU–GPU environments, where regular control flow and predictable memory access are critical for achieving high parallel performance.

In the proposed framework, CPU-centric tasks such as rule parsing, fast-pattern extraction, index construction, and flow management remain on the CPU, while the data-parallel payload scanning stage is

selectively offloaded to the GPU using WEMA's weighted matching mechanism. This division of responsibilities preserves exact matching semantics, avoids probabilistic false positives, and maintains compatibility with existing Snort-like rule verification pipelines.

Experimental results obtained on commodity hardware equipped with a multicore CPU and an integrated GPU demonstrate that WEMA provides stable and predictable execution in CPU-only mode across all tested payload sizes. The hybrid CPU–GPU execution model delivers modest but consistent performance improvements over pure CPU execution for small and medium-sized workloads. However, GPU-only execution proves less effective due to the architectural limitations of integrated GPUs. These findings confirm that the performance gains achievable through WEMA-based acceleration are strongly influenced by both hardware characteristics and workload size, highlighting the importance of selective and hardware-aware acceleration strategies.

Future work will focus on extending the proposed framework to support discrete GPU platforms, investigating optimized WEMA index layouts to further improve memory coalescing, and developing adaptive workload partitioning strategies that dynamically select between CPU-only and hybrid execution modes based on traffic conditions and system state. Additional directions include integrating advanced variants such as EWEMA and evaluating the framework under live network traffic to further assess its scalability and real-time capabilities.

Acknowledgement: The authors would like to thank Al-Zaytoonah University of Jordan—Faculty of Science and Information Technology for its support that enabled us to complete this work.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Adnan Hnaif; methodology, Adnan Hnaif and Hanadi Al-Shawabkah; software, Hanadi Al-Shawabkah; validation, Ayman Alqafaan and Mohammad Alia; formal analysis, Adnan Hnaif and Mohammad Alia; investigation, Hanadi Al-Shawabkah and Ayman Alqafaan; resources, Adnan Hnaif, Hanadi Al-Shawabkah and Ayman Alqafaan; data curation, Ayman Alqafaan; writing—original draft preparation, Hanadi Al-Shawabkah; writing—review and editing, Mohammad Alia; visualization, Adnan Hnaif; supervision, Adnan Hnaif. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the Corresponding Author, [Adnan Hnaif], upon reasonable request.

Ethics Approval: Not applicable.

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

Abbreviations

The following abbreviations are used in this manuscript:

NIDS	Network Intrusion Detection Systems
WEMA	Weighted Exact Matching Algorithm
CPU	Central Processing Unit
GPU	Graphics Processing Unit
IDS	Intrusion Detection System
SIMD	Single Instruction Multiple Data

References

1. Abbas A, Khaled H. Enhanced Aho-Corasick algorithm for NIDS. Int J Intell Comput Inf Sci. 2024;24(3):83–92. doi:10.21608/ijicis.2024.315432.1349.
2. Abbasi M, Afshari Haghdoost M. Improvement and parallelization of Snort network intrusion detection mechanism using graphics processing unit. JSDP. 2021;18(1):150–35. doi:10.52547/jsdp.18.1.150.
3. Karim I, Vien Q-T, Le TA, Mapp G. A comparative experimental design and performance analysis of snort-based intrusion detection system in practical computer networks. Computers. 2017;6(1):6. doi:10.3390/computers6010006.
4. Shenify MA, Alghamdi AS, Alharthi AF. Hybrid supervised machine learning-based intrusion detection system of Internet of Things. Int J Adv Soft Comput Its Appl. 2024;16(2):68–84. doi:10.15849/ijasca.240730.05.
5. Lee CL, Lin YS, Chen YC. A hybrid CPU/GPU pattern-matching algorithm for deep packet inspection. PLoS One. 2015;10(10):e0139301. doi:10.1371/journal.pone.0139301.
6. Lin RCH, Huang YH, Lin CY, Lin YC. GPU application on multi-pattern matching of network intrusion detection. J Internet Technol. 2013;14(7):1033–41.
7. Hnaif AA. A new platform NIDS based on WEMA. Int J Inf Technol Comput Sci. 2015;7(6):52–8. doi:10.5815/ijitcs.2015.06.07.
8. Alia MA, Hnaif AA, Al Anie HK, Abu Maria K, Manasrah AM, Sarwar MI. A novel header matching algorithm for intrusion detection systems. Int J Netw Secur Appl. 2011;3(4):59–73. doi:10.5121/ijnsa.2011.3406.
9. Lee CL, Yang TH. A flexible pattern-matching algorithm for network intrusion detection systems using multi-core processors. Algorithms. 2017;10(2):58. doi:10.3390/a10020058.
10. Talaei Khoei T, Kaabouch N. A comparative analysis of supervised and unsupervised models for detecting attacks on the intrusion detection systems. Information. 2023;14(2):103. doi:10.3390/info14020103.
11. Suresh P, Sukumar R, Ayyasamy S. Efficient pattern matching algorithm for security and Binary Search Tree (BST) based memory system in Wireless Intrusion Detection System (WIDS). Comput Commun. 2020;151(10):111–8. doi:10.1016/j.comcom.2019.11.035.
12. Vasiliadis G, Antonatos S, Polychronakis M, Markatos EP, Ioannidis S. Gnort: high performance network intrusion detection using graphics processors. In: Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection; 2008 Sep 15–17; Cambridge, MA, USA. p. 116–34. doi:10.1007/978-3-540-87403-4_7.
13. Jawad MA, García J, Masoud M. A survey of network intrusion detection systems (NIDS) based on machine learning algorithms for industrial Internet of Things (IIoT). In: Artificial intelligence in business. Cham, Switzerland: Springer Nature; 2025. p. 158–67. doi:10.1007/978-3-031-96631-6_16.
14. Khaled H, Abbas A, Fayez M. Multi-pattern GPU accelerated collision-less Rabin-karp for NIDS. Int J Distrib Syst Technol. 2024;15(1):1–16. doi:10.4018/IJDST.341269.
15. Al Mimi H, Hamad NA, Abualhaj MM, Daoud MS, Al Dahoud A, Rasmi M. An enhanced intrusion detection system for protecting HTTP services from attacks. Int J Adv Soft Comput Its Appl. 2023;15(2):1–18.
16. Nazzal D, Hnaif AA, Al'otoum IS, Alia MA, Aldahoud A. Multiprocessing scalable string matching algorithm for network intrusion detection system. Int J High Perform Syst Archit. 2018;8(3):159. doi:10.1504/ijhpsa.2018.10022485.
17. Tumeo A, Villa O, Sciuto D. Efficient pattern matching on GPUs for intrusion detection systems. In: Proceedings of the 7th ACM International Conference on Computing Frontiers; 2010 May 17–19; Bertinoro, Italy. p. 87–8. doi:10.1145/1787275.1787296.
18. Lin YS, Lee CL, Chen YC. Length-bounded hybrid CPU/GPU pattern matching algorithm for deep packet inspection. Algorithms. 2017;10(1):16. doi:10.3390/a10010016.
19. Çelebi M, Yavanoğlu U. Accelerating pattern matching using a novel multi-pattern-matching algorithm on GPU. Appl Sci. 2023;13(14):8104. doi:10.3390/app13148104.