ScaleMon: Scalable System and Application Monitoring and Attack Detection for Secure HPC Systems.

Abstract

This paper presents ScaleMon, a scalable monitoring framework for identifying malicious execution in HPC This paper presents ScaleMon, a scalable monitoring framework for identifying malicious activities in both applications and systems for high-performance computing (HPC) systems. Since a small number of applications run extensively on HPC systems, ScaleMon focuses on patterns specific to each application and detects anomalies by analyzing I/O behaviors. To achieve this, ScaleMon 1) leverages existing HPC system monitoring tools to collect data, 2) introduces a novel data representation that converts voluminous, fine-grained DXT logs into fixed-size 3D image tensors, preserving key I/O patterns (offset, time, size) while drastically reducing data volume, 3) employs a Synthetic Anomaly Generator (SAG) to perturb benign log images, creating diverse synthetic anomalies to teach the model subtle distinctions between normal and malicious behaviors, and 4) adopts a flexible, machine learning–based detection pipeline that models individual applications and generalizes to unseen workloads. We implement ScaleMon and evaluate its performance on the Leadership-scale HPC systems using I/O logs from real scientific applications. Our evaluation results show that ScaleMon improves detection performance significantly **by XX**, outperforming four baseline one-class models in both Recall and Precision in all test cases.

1 Introduction

- citation should be added in introduction. usage of slurm log: identify log, concept drift?

High-performance computing (HPC) has become indispensable for modern scientific breakthroughs as applications grow increasingly complex [47, 83, 96]. These systems integrate thousands of processing units (e.g., CPU [6] and GPU [13]) and high-throughput storage devices (e.g., SSD [35] and HDD [44]), interconnected via high-speed fabrics such as Slingshot [21], to achieve extreme scalability and computational power.

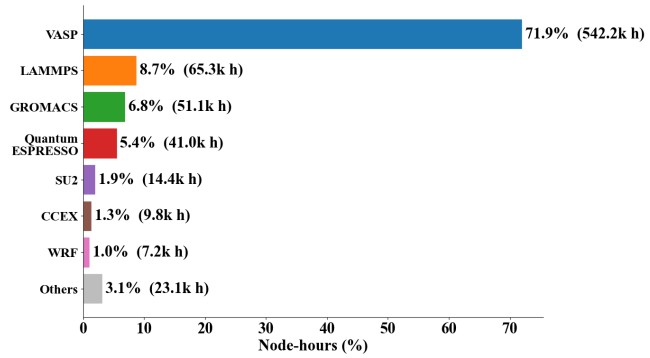


Figure 1: Total runtime per application in a HPC cluster

However, the performance-oriented nature of HPC environments introduces unique security challenges [11, 33, 71]. Because performance often outweighs security considerations, rapid data exchange and open collaboration can create weak points in system defenses. As a result, HPC systems face threats such as data leakage, integrity violations (e.g., code or data modification), and availability attacks like cycle misuse or enial-of-service [71]. Moreover, the growing heterogeneity of hardware and software components further expands the attack surface—including the adoption of different computation units, such as CPUs and GPUs, from various vendors that support diverse software stacks [8, 27, 40, 63, 78]. These factors demand systematic approaches to safeguard these mission-critical systems.

However, these software and hardware hetrogenity, HPC systems have two distinct characteristics that can be utilized for security

Same applications executed multiple times In most HPC clusters, a relatively small and fixed set of applications is executed repeatedly, resulting in much lower diversity compared with general-purpose computing systems. Figure 1 shows the top 7 application runtimes in a production HPC cluster from July 2023 to June 2024. The results indicate that the top application, VASP, accounts for over 70% of the total execution time, while the top 10 applications together account for more than 97%. This regularity indicates that HPC work-

Table 1: Comparison of Modern Intrusion Detection Systems.

Method	Data Source	HPC-Applicable	I/O Monitoring	
			Inter-File	Intra-File
NIDS				
Bro [70]	Network Packets	△		
Cerberus [97]	Network Packets	△		
HIDS				
DeepLog [25]	System Event Logs	△	△	
Kairos [12]	Audit Logs		✓	
Flash [75]	Audit Logs		✓	
MAGIC [45]	Audit Logs		✓	
ORTHRUS [46]	Audit Logs		✓	
VELOX [7]	Audit Logs		✓	
ScaleMon (Ours)	**HPC I/O Logs**	✓	✓	✓

loads are highly concentrated, providing a unique opportunity to enhance security by leveraging machine learning (ML). ML-based approaches have demonstrated strong performance in pattern recognition and have been widely explored for attack detection. However, despite their potential, deploying ML-based cybersecurity systems in real-world environments remains challenging due to the high diversity and variability of benign behavior. In contrast, the repetitive and well-defined workload patterns in HPC systems can help mitigate this challenge, making them an ideal environment for ML-driven anomaly detection.

Feature-rich low-overhead logging systems Most HPC clusters are equipped with a variety of logging tools that continuously monitor system behavior in real time. These tools are designed to impose minimal overhead due to the performance-centric nature of HPC environments. For instance, resource- and hardware-level metrics such as CPU/GPU utilization, memory usage, and network traffic can be monitored, alongside Slurm job scheduler logs and Darshan I/O profiling tools, which track various aspects of job execution and application I/O behavior. Consequently, HPC systems generate a rich set of logs, from which valuable information can be extracted. This data can be used not only for performance optimization and identifying system bottlenecks, but also for security monitoring and anomaly detection.

Given the richness of HPC log data, security monitoring should be conducted from multiple perspectives. This is necessary because certain attacks may leave little or no trace from a single viewpoint, either due to the attack's inherent nature or the attacker's efforts to conceal their activities. Among the various perspectives, I/O behavior provides particularly informative signals. Many attacks in HPC environments eventually leave traces at the I/O level. Attackers often require I/O operations to accomplish their malicious goals, and even when they do not, attacks can indirectly alter I/O behavior. Even if an attacker attempts to appear benign by renaming binaries or mimicking normal behavior, their malicious intent will eventually produce abnormal I/O patterns when the real payload is executed. Such deviations can be revealed through careful I/O observation. Therefore, monitoring I/O activity is a valuable approach for enhancing security.

While numerous Intrusion Detection Systems (IDS) exist, as summarized in Table 1, they fall short in addressing the specific constraints of HPC I/O monitoring. Network-based Intrusion Detection Systems (NIDS) like Bro(Zeek) [70] and Cerberus [97] are restricted to monitoring network boundaries and cannot inspect the I/O behavior of an executions. Host-based Intrusion Detection Systems (HIDS) offer deeper analysis. However, approaches like DeepLog [25], which analyze general system event sequences, can only indirectly infer inter-file relationships and do not capture any intra-file details. More recent provenance-based IDS [12, 45, 75] successfully model inter-file I/O by analyzing graphs from kernel audit logs. Nevertheless, they face two fundamental limitations in the HPC context: first, the significant performance overhead of their required logging tools (e.g., auditd, Falco, CamFlow) makes them impractical for production systems [19, 39]; second, they still lack visibility into the intra-file level, such as the specific offsets and sizes of I/O operations. In contrast, `ScaleMon` distinguishes itself from these previous studies by providing deep, multi-level I/O monitoring at both the inter-file and intra-file levels, while being specifically designed to meet the stringent performance demands of production HPC systems.

In this paper, we present `ScaleMon`, a scalable monitoring framework that introduces a hierarchical inspection methodology to detect a wide spectrum of attacks in HPC systems. Our work is driven by three primary goals: 1) Comprehensive Threat Coverage 2) Exascale Scalability 3) Zero-Day Readiness To achieve these goals, `ScaleMon` 1) employs a hierarchical architecture consisting of an `Identity Verifier`, `Inter-Mon`, and `Intra-Mon`, with each module focusing on a different scale of I/O behavior; 2) utilizes lightweight data representations derived from a lightweight Darshan-based profiling pipeline, including execution profile vectors and I/O Images, to drastically compress raw I/O traces while preserving the spatial-temporal access patterns necessary for detection; and 3) builds upon one-class anomaly detection models trained exclusively on benign data. Our evaluation on widely used scientific applications on a production HPC system demonstrates the effectiveness and practicality of this approach. `ScaleMon` successfully detects all diverse range of attacks, ranging from blatant masquerading to stealthy data tampering, while inducing minimal overhead acceptable for production environments (e.g., less than xxx% of application runtime). As our models are trained solely on benign data, these high-performance results also validate `ScaleMon` 's Zero-Day Readiness. We have open-sourced the code for `ScaleMon` at `https://github.com/ScaleMon/ScaleMon.git` and the dataset at `https://doi.org/10.5281/zenodo.XXXXXX`.

2 Background

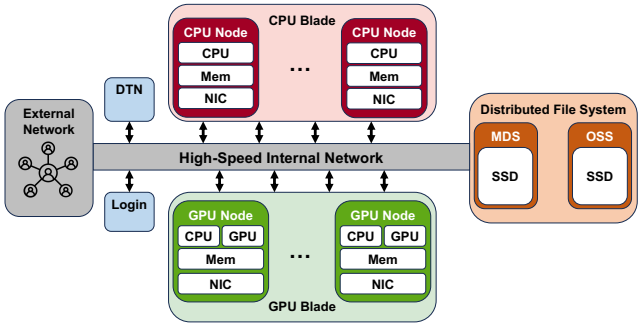


Figure 2: HPC architecture

2.1 HPC Architecture

Figure 2 shows the overall architecture of HPC system. As HPC systems are design to solve large-problems in a short period time, HPC systems have thousands of compute nodes connect via high-speed interconnect such as Infiniband [72] and Slingshot [21]. Each compute node is a complete server with CPU, Memory and Storage but each HPC system have different level of abstraction and operablilty to the user. Some systems run full-scale operating system [68] while others run lightweight version of the operationing system for performance and security [48, 84, 89]

2.2 I/O Tracing in HPC

I/O tracing in HPC environments prioritizes minimal performance interference, leading to the widespread adoption of user-level profiling tools over heavy system-wide monitors. [37] Darshan serves as the de facto standard for I/O characterization in this domain, designed to capture application behavior by intercepting I/O calls at the library level (e.g., MPI-IO, HDF5, POSIX) rather than utilizing heavy kernel instrumentation. While the standard Darshan module aggregates statistics to reduce log size, its Darshan eXtended Tracing (DXT) [90] module provides the granular visibility required for security analysis by recording a complete trace of every I/O operation, including timestamps, file offsets, and request sizes. This detailed telemetry enables the reconstruction of precise intra-file access patterns without the overhead associated with system call interception.

In contrast, general-purpose security monitors (e.g., Auditd, CamFlow [69], Falco) provide deep system visibility but face fundamental challenges in HPC. Intercepting system calls at the kernel boundary introduces prohibitive overheads and log loss under high-throughput workloads [**?**], while OS-bypass mechanisms (e.g., RDMA) create significant blind spots. Although recent HPC provenance frameworks [**?**, 19, 36] have addressed these issues by shifting toward user-level instrumentation, they primarily focus on capturing high-level data lineage rather than the fine-grained spatial-temporal access patterns (e.g., offsets, strides) essential for detecting subtle I/O anomalies, thereby reinforcing the necessity of utilizing specialized profiling tools like Darshan and DXT.

2.3 Security vulnerabilities in HPC

The complexity and scale of HPC infrastructures also introduce substantial security vulnerabilities. Due to their interconnected and shared-resource nature, HPC systems are particularly susceptible to unauthorized access, breaches of confidentiality, and operational disruptions. These risks necessitate detailed threat assessments and comprehensive security strategies to safeguard HPC operations effectively [51].

Hardware-based Attacks in HPC: Hardware-level vulnerabilities pose significant threats to HPC environments due to the widespread use of performance-oriented processors and GPUs. Speculative execution attacks, such as Spectre, exploit processor mechanisms intended for computational efficiency, enabling attackers to bypass isolation measures and access confidential information across processes [38]. Additionally, Graphics Processing Units (GPUs), essential for accelerating HPC computations, have been shown to unintentionally leak sensitive data during routine operations, further compromising data confidentiality and system security [24]. Finally, Side-channel attacks are a growing concern in multi-tenant HPC environments where different users share compute nodes or memory channels. These attacks exploit shared hardware resources like caches, memory buses, or performance counters to infer sensitive information from co-located processes. In HPC contexts, where high utilization is common, such co-residency scenarios are frequent and thus increase the feasibility of attacks [71]. Recent studies show that even GPU-based computation can leak data through memory access patterns, creating new channels for information exfiltration [34].

Exploitation of Parallel Runtime Systems and Job Schedulers
Parallel runtimes such as MPI and job schedulers like SLURM or PBS are core components of HPC operations. However, their complexity and central role in resource orchestration make them attractive targets for attackers. For instance, poorly validated job scripts can be abused to inject payloads or manipulate environment variables to gain unintended access [26]. In addition, the attack can manipulate the systems log generated from the malicious activity to avoid attack detection. HPC systems face broader vulnerabilities stemming from their software and infrastructure components. Comprehensive risk assessment methodologies are required to identify and manage these threats effectively [58]. Moreover, proactive strategies involving specialized security testbeds enable researchers and system administrators to simulate potential attacks, thereby strengthening defensive measures and minimizing security risks [10].

Data Breach HPC facilities are engineered for extreme data movement, utilizing high-speed interconnects and Data Transfer Nodes (DTNs) optimized for performance, not inspection. This architecture, while beneficial for scientific throughput, allows attackers to quickly exfiltrate data once access is obtained. Traditional security mechanisms like firewalls or deep

packet inspection are often disabled or relaxed to maintain I/O performance [71]. As a result, once attackers infiltrate a system, they can transfer large volumes of sensitive data with minimal detection. For example, ES-net [52]. In addition, as the application that utilizes network connection becomes diverse, it is becomming increasingly difficult to detect and defend [9, 30]

2.4 Defense Mechanisms in HPC

Ensuring the security of HPC systems requires a multi-layered defense strategy that integrates both protection and detection mechanisms. Due to the large-scale, distributed, and performance-sensitive nature of HPC environments, conventional enterprise security models are often insufficient. HPC systems must simultaneously achieve minimal performance interference, fine-grained isolation, and real-time anomaly awareness. Therefore, security strategies are designed not only to prevent attacks through architectural hardening but also to rapidly detect and respond to abnormal behaviors.

Protection mechanisms focus on reducing the attack surface and enforcing strict access control. The Science DMZ architecture [18, 20] delineates exposed and non-exposed network zones, while lightweight kernels (LWKs) [54, 57] reduce the trusted computing base to lower kernel-level exploits. Hardware-assisted Trusted Execution Environments (TEEs) like Intel SGX [17], AMD SEV [77], ARM TrustZone [73], and NVIDIA Confidential Computing [23] guarantee computation integrity. Complementing these are access control mechanisms such as Access Control Lists (ACLs) [59] for packet-level filtering and system-level Access Control (DAC/MAC) [66, 76] models. Furthermore, Two-Furthermore, two-factor [3] and multi-factor authentication (MFA) [4] are adopted to harden the authentication process.

Detection mechanisms complement protection by continuously monitoring network and runtime environments. At the network level, tools like Bro (Zeek) [70] transform packets into event streams, flow-based methods like FlowRadar [60] and sFlow [32] provide scalable monitoring, and machine learning models [53] learn features of anomalous behaviors. Beyond network observation, execution-level frameworks such as DPEM [50] and Varan [42] detect control-flow deviations, while DeepLog [25] models system logs to find anomalies. Other strategies include analyzing MPI communication patterns [14, 56, 88], monitoring CPU utilization [31, 98], utilizing power-based side-channel analysis [16], and applying I/O-level profiling via Ghostbuster [62] to uncover misuse.

However, existing execution-level monitoring frameworks still lack mechanisms to distinguish stealthy attacks that mimic benign I/O activities. To address this gap, our work focuses on fine-grained I/O behavior analysis at both inter-file and intra-file levels. Since low-level I/O operations are inherently difficult to conceal, monitoring these microscopic patterns enables the detection of subtle deviations that appear legitimate at a macroscopic perspective but diverge within the intra-file access context.

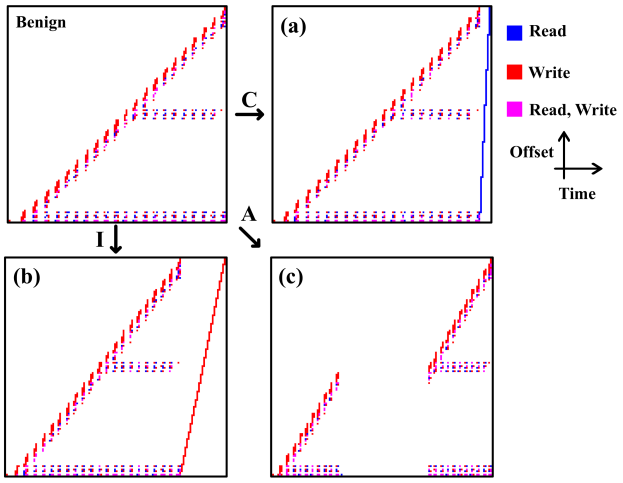


Figure 3: Examples of intra-file I/O behavior

Table 2: Mapping of Targeted Attacks to I/O Footprints.

Security Goal	Observation Level	Anomalous I/O Footprint	Attack Scenario
Confidentiality	Inter-file	Anomalous File Path Access	Data Breach
	Intra-file	Full Sequential Scan	
Integrity	Inter-file	Illegitimate Write Operation	Data Tampering
	Intra-file	Full Sequential Overwrite	
Availability	Inter-file	Anomalous File Count	Denial of Service
	Intra-file	I/O Delay	Resource Exploitation

3 Threat Model and Scope

In this section, we define the threat model for which `ScaleMon` is designed. We first establish our core assumptions and the operational scope of our system, and then detail the specific attack categories we target and their traces in I/O behavior.

3.1 Attack Vectors and Threat Model

`ScaleMon` specifically targets stealthy attacks that mimic benign applications. These attacks appear normal on the surface, making them difficult to detect with conventional security methods. Therefore, a lower-level inspection of system behavior is required to identify them. Specifically, `ScaleMon` detects jobs that show I/O activities inconsistent with the expected behavior of a legitimate HPC application. We consider two primary attack vectors:

Direct Execution of Malicious Applications: An attacker who has already gained access to the system can run an application that looks benign but secretly contains malicious attack code.

Supply-Chain Attacks: An attack can be carried out by linking a maliciously tampered library to a normal application.

The latter vector is particularly dangerous, as innocent users can unknowingly execute the compromised code, allowing the threat to spread widely and cause concurrent attacks across the system [55].

Our threat model relies on a few key assumptions. We assume that the underlying system hardware, operating system kernel, and the I/O logging infrastructure itself form a trusted computing base (TCB) and are not compromised. Therefore, the I/O traces we collect are considered reliable. Furthermore, attacks targeting the machine learning models directly, such as poisoning or evasion attacks, are considered out of scope for this work. We assume that all data used to train the components of ScaleMon consists of benign, uncompromised execution traces.

3.2 Targeted Attacks and their I/O Footprints

To motivate our design, we consider a range of attack scenarios that compromise the core security goals of a system: Confidentiality, Integrity, and Availability (C-I-A). `ScaleMon` is designed to detect these attacks by analyzing I/O behavior at two distinct levels of granularity:

The Inter-file Level: This macroscopic view treats files as atomic objects. The analysis focuses on file metadata (e.g., path, type), the operations performed on them (e.g., read, write), and their relationships to other files within the same execution (e.g., count, relational distance). **The Intra-file Level:** This microscopic view inspects the fine-grained, spatio-temporal I/O patterns within a single file. The analysis focuses on patterns formed by the sequences of request offsets, sizes, operation types, and their respective timings.

Table 2 summarizes the mapping between core security goals, the anomalous I/O footprints observable at each level, and the corresponding attack scenarios. Figure 3 visually contrasts the intra-file I/O behavior of a benign LAMMPS execution with that of its simulated C-I-A–compromising attack variants. The x-axis indicates time, the y-axis indicates file offsets, and colors denote operation types.

Confidentiality Attacks: Data Breach. HPC systems often store valuable intellectual property and sensitive data, making data breaches a primary threat [**?**]. At the inter-file level, a blatant attempt can manifest as an Anomalous File Path Access, where a compromised application accesses files far outside its normal project scope. A more stealthy attack, however, might target the output data of the Experiment itself. While appearing benign from an inter-file perspective, this attack leaves a clear footprint at the intra-file level.too conclusive? To stage the data for exfiltration, the application may perform a Full Sequential Scan after the main Experiment is completed. An example of this simulated I/O footprint is visualized in Figure 3(a).

Integrity Attacks: Data Tampering. Attacks targeting data integrity are a significant threat in HPC environments, as tampered experimental results can undermine scientific research or compromise critical decisions in fields like national security [**?**].At the inter-file level, a straightforward integrity attack leaves an obvious footprint as an Illegitimate Write Operation, for instance, writing to files that are normally read-only, such as source code or input configuration files. A more sophisticated attack, however, could appear legitimate from an inter-file perspective by performing a seemingly proper write operation on an output file. The malicious intent is only revealed at the intra-file level. For example, an attacker might aim to sabotage results or deploy ransomware by overwriting the entire content of an output file. We simulate this behavior as a Full Sequential Overwrite performed after the main experiment is completed. An example of this simulated I/O footprint is visualized in Figure 3(b).

Availability Attacks: DoS & Resource Exploitation. Availability attacks aim to disrupt services or misuse system resources. A Denial-of-Service (DoS) attack can be particularly critical in HPC environments, as system downtime leads to substantial losses in scientific productivity. [] At the inter-file level, such a DoS attack can be launched against a parallel file system's metadata server by creating or accessing an Anomalous File Count. Because of their extraordinary computing power, HPC environments are also a prime target for resource exploitation []. At the intra-file level, a resource exploitation attack like cryptocurrency mining can be detected via its secondary effects. As the malicious code consumes CPU cycles, the host application's I/O operations are effectively paused, causing a distinct I/O Delay footprint. An example of this simulated I/O footprint is visualized in Figure 3(c).

4 Design

In this section, we present `ScaleMon`, blarr blarr

4.1 Overall Architecture

`ScaleMon` is a novel, lightweight, and scalable hierarchical IDS for HPC systems. Its detection pipeline is composed of two data processing stages—a LogParser and a FingerprintGenerator—and three specialized detection modules: the `Identity Verifier`, `Inter-Mon`, and `Intra-Mon`. Each module is designed to focus on different scales and aspects of I/O behavior to provide comprehensive threat coverage. Figure 4 illustrates the overall architecture and data flow of `ScaleMon`'s detection pipeline.

The pipeline begins post-execution by scrutinizing the generated Darshan logs. The LogParser first converts the binary `.darshan` file into two intermediate text files, `darshan.txt` and `dxt.txt`, using Darshan's utility tools. During this phase, the application's Claimed Identity is extracted from the binary name found in `darshan.txt`. These text files are then parsed into two structured tables: a `summary.csv` and a

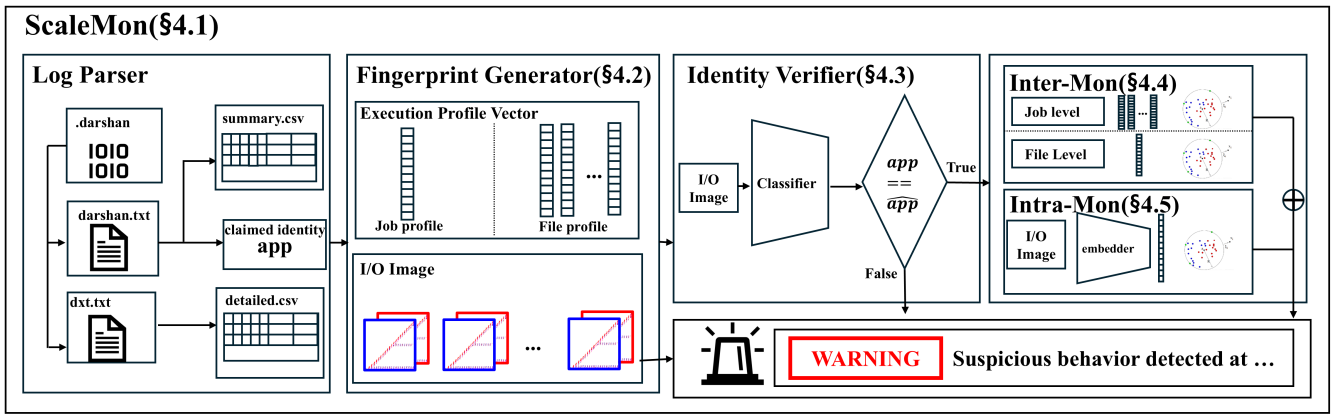


Figure 4: Overall Architecturedraft

`detailed.csv`. The `summary.csv` provides a comprehensive, execution-level overview of all accessed files and serves as the basis for inter-file analysis. The `detailed.csv`, conversely, contains the fine-grained I/O traces for files accessed via specific interfaces like `POSIX I/O`. Finally, the Fingerprint Generator processes these tables, aggregating and transforming the summary.csv into Execution Profile Vectors and the `detailed.csv` into I/O Images for the subsequent detection stages.

The detection pipeline then commences with the `Identity Verifier`, which compares the Claimed Identity with an Inferred Identity predicted from the I/O image. If they do not match, an alert for a potential masquerading attempt is raised, and further analysis is bypassed. If the identity is verified, the execution proceeds to the in-depth inspection stages, which operate in parallel. Specifically, `Inter-Mon` analyzes the Execution Profile Vectors to detect anomalies at the inter-file level, capturing the macroscopic behavior and context of file accesses. Concurrently, `Intra-Mon` scrutinizes the I/O Images to identify suspicious patterns at the fine-grained, intra-file level. An alarm is triggered if an anomaly is detected by either module, leveraging their complementary perspectives to provide comprehensive detection.

4.2 Fingerprint Generator

Finger print Generator represent table data to ...

4.2.1 Execution Profiles Vector: Inter-file Level Fingerprint

To comprehensively summarize the inter-file level I/O behavior, we generate two complementary vectors for each execution: the **Job Profile Vector** and the **File Profile Vectors**. Specifically, a single Job Profile Vector represents the execution globally, while a File Profile Vector is generated for each file accessed during the execution. These vectors treat files as atomic objects, capturing both the specific characteristics of individual file accesses within the context of other accessed

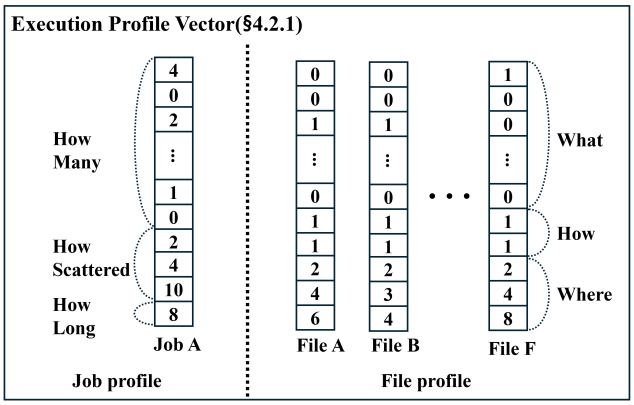


Figure 5: Execution Profile Vector

files and the aggregate behavior of the entire job. Figure 5 visualizes the composition of these vectors.

File Profile Vector The File Profile Vector creates a unique fingerprint for each accessed file by addressing three fundamental contextual questions: *what* type of file it is, *how* it is operated, and *where* it is located relative to others. By concatenating the sub-vectors corresponding to these questions, we form a unified representation for every file involved in the execution.

What: File Type. This component identifies the semantic category of the file based on its name or extension. Specifically, a predefined keyword dictionary is utilized to classify the file. If a direct match is not found, the last token of the base name serves as a fallback feature. During the training phase, all unique file types are mapped to positions in a one-hot encoded vector. In the inference phase, any unseen file types result in a zero-vector. In the `Inter-Mon` module, such unknown types are flagged for immediate inspection before passing through the ML model, as accessing unusual file types is inherently suspicious in HPC applications.

How: Operation Type. This component describes the nature of the I/O operations performed on the file. It is represented as a 2-bit vector, where the first bit indicates a 'Read' operation and the second indicates a 'Write' operation. For

instance, a file that is both read from and written to during an execution is encoded as `[1, 1]`, whereas a read-only file is `[1, 0]`. By analyzing 'What' and 'How' jointly, we can detect semantic anomalies, such as illegitimate write operations to typically read-only configuration files. However, these two dimensions alone cannot detect attacks involving legitimate operations on anomalous file paths, which can still pose a severe threat to confidentiality. To address this, a spatial component representing the file location is required.

Where: Relational Location. This component quantifies the spatial relationship of a specific file within the directory structure relative to other files accessed in the same execution. We quantify this "spatial distance" using the Lowest Common Ancestor (LCA) in the directory tree. Formally, the distance function $d(\text{file}_1, \text{file}_2)$ is defined as:

$$d(\text{file}_1, \text{file}_2) = d(\text{file}_1, \text{LCA}) + d(\text{file}_2, \text{LCA}) \quad (1)$$

where LCA denotes the lowest common ancestor directory of file_1 and file_2.
We define the base cases as:

$$d(x, x) = 0, \quad (2)$$

$$d(\text{parent}, \text{child}) = 1. \quad (3)$$

For each accessed file, we compute three statistical features based on its pairwise LCA distances to all other accessed files: the *min*, *mean*, and *max* distance. Using these components, a file located in a distant path relative to others will exhibit prominent distance values, effectively flagging it as a spatial outlier.

Job Profile Vector While the File Profile Vector characterizes individual files, relying solely on it has limitations. An attack might consist of operations that appear benign at the file level—such as legitimate operations on valid file types—but are malicious at the job level, for instance, accessing an excessive number of files to induce a Denial of Service. Therefore, the Job Profile Vector is necessary to comprehensively represent the macroscopic inter-file level I/O behavior.

This vector aggregates statistics to answer three macroscopic questions: *how many* files were accessed per type, *how scattered* the accesses were across the file system, and *how long* the execution persisted. First, to capture the volume of activity (**how many**), we accumulate the one-hot encoded file type vectors from all File Profiles, resulting in a distribution vector that counts the occurrences of each file type. Second, to measure the spatial dispersion of I/O (**how scattered**), we compute the global minimum, mean, and maximum LCA distances between all pairs of files accessed during the execution. Finally, we record the total runtime (**how long**) to provide temporal context relative to the I/O behavior. By combining these macroscopic features, the Job Profile Vector effectively complements the microscopic details of the File Profile Vector.

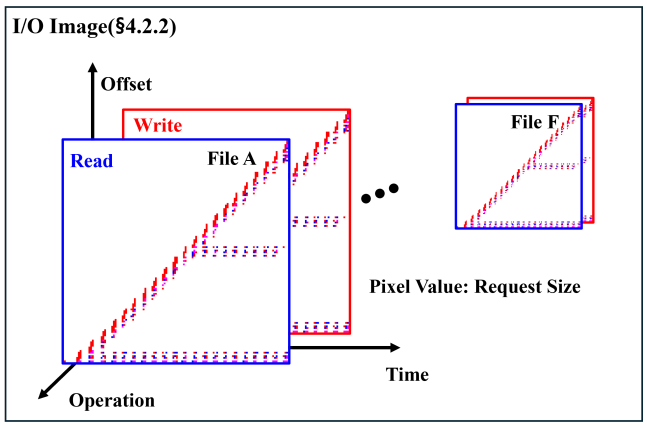


Figure 6: I/O Image

4.2.2 The I/O Image: Intra-file level Fingerprint

At the core of our microscopic analysis lies the I/O Image, a novel representation designed to transform voluminous, fine-grained I/O logs into a compact and analyzable format. For each execution, we selectively generate I/O Images only from the files recorded in the DXT logsdxt.txt?, which provide detailed traces of POSIX I/O operations. The complete log of I/O activities for a single file is converted into one fixed-size I/O image tensor with dimensions of $2 \times \texttt{image\_size} \times \texttt{image\_size}$, where `image_size` is a configurable hyperparameter.

references should be added to enforce our argument difficulty of The motivation for this transformation is twofold, enhancing both efficiency and effectiveness. From an efficiency standpoint, raw DXT logs can be excessively large, often reaching gigabytes in secondsshould be measured, making direct analysis computationally prohibitive in terms of both time and memory. Our fixed-size image representation drastically compresses this data, enabling lightweight and scalable processing. More importantly, this transformation is often more effective than analyzing the raw time-series data directly. Due to the extreme length of I/O sequences, too deterministic?any time-series analysis must operate on small, windowed segments of the data. This window-level inspection inherently struggles to capture the global, long-range patterns of an execution, focusing only on local behaviors. In contrast, our I/O image normalizes the entire execution's I/O activity for a file into a single, holistic view, allowing our models to capture both local and global patterns simultaneously. efficiency and effectiveness of this representation is evaluated on Section 5.3.1

Figure 6 visualizes the process of converting the spatio-temporal I/O patterns of a file into an I/O image. The three dimensions of the image tensor are mapped as follows:
Channel (Axis 0): The two channels distinguish the I/O operation type, with channel 0 representing read operations and channel 1 for write operations.
Height (Axis 1): The vertical position of a pixel corresponds to the I/O offset, which is normalized via min-max scaling to

fit the image height.
Width (Axis 2): The horizontal position corresponds to the I/O time, which is also normalized to fit the image width.
The pixel value at a specific `(channel, height, width)` coordinate is the accumulated sum of the sizes of all I/O accesses that fall into that spatio-temporal bin. Finally, the pixel values within each channel are normalized to a range of `[0, 1]`. Through this process, the entire sequence of spatio-temporal access patterns from the DXT log is encoded into a single, dense I/O image.

all component process data as batch but it mentioned only in some section.

4.3 Identity Verifier

The `Identity Verifier` serves a dual purpose within the `ScaleMon` architecture: ensuring routing integrity and providing early-stage anomaly detection. Primarily, its role is to validate the consistency between an application's claimed identity (e.g., binary filename) and its actual I/O behavior. This step is critical because the subsequent modules, `Inter-Mon` and `Intra-Mon`, are designed to monitor adherence to specific application profiles. If an execution stream is routed to an incorrect model due to a deceptive identity, the system could be misled, resulting in unreliable detection. Simultaneously, this module functions as an initial gatekeeper. By inspecting the intra-file level I/O behavior via I/O Images, it detects less stealthy attacks that exhibit patterns distinctively different from the claimed application's benign behavior. Identifying such overt anomalies at this stage allows `ScaleMon` to flag the execution immediately, bypassing both subsequent monitoring phases, `Inter-Mon` and `Intra-Mon`.

To implement this, we employ a lightweight CNN-based classifier (ResNet18) trained on benign I/O Images to distinguish between application types. This training can be performed offline using historical benign I/O Images accumulated by running `ScaleMon`. During detection, the module processes all I/O Images generated from an execution in a batch. For each image, it evaluates the classifier's confidence score for the claimed identity. If the confidence falls below a predefined threshold (0.6 in this work), it implies that the observed behavior does not align with the claim. If any single image in the batch is flagged as suspicious, the entire execution is treated as a potential masquerading attempt or anomaly, triggering an immediate alert.

4.4 Inter-Mon

The `Inter-Mon` module is responsible for scrutinizing the inter-file level behavior captured in the Execution Profile Vectors. It operates by evaluating the Job Profile Vector and the set of File Profile Vectors in parallel, using distinct models for each vector type. The system triggers an alert if any individual vector is flagged as anomalous. This parallel processing ensures that inter-file level deviations are captured simultaneously across different granularities.

To ensure robustness against zero-day attacks and independence from specific attack signatures, we employ one-class anomaly detection models. These models are trained exclusively on benign execution data, enabling them to identify any deviation from normal patterns as suspicious. Specifically, each model outputs an anomaly score for the input vector; if this score exceeds a predefined threshold, the execution is judged as an anomaly. In this work, the threshold is set at the 99th percentile of the anomaly scores observed in the benign training data, and this threshold can be adaptively updated during operation. The architecture is model-agnostic, supporting algorithms such as k-NN, LOF, Isolation Forest, and AutoEncoders, as evaluated in Section 5.2.2. Among these, we primarily adopt the Deep SVDD model for our evaluation due to its efficiency and performance.

4.5 Intra-Mon

The `Intra-Mon` module performs an intra-file level inspection of the I/O Images that have successfully passed the `Identity Verifier`. Its primary objective is to detect highly stealthy attacks that appear benign at the inter-file level and exhibit patterns deceptively similar to normal operations at the intra-file level. For efficient processing, all I/O Images associated with a single execution are evaluated in a batch. if any individual image within the batch is flagged as anomalous, the entire execution is immediately alerted as suspicious.

The detection process begins by capturing the graphical patterns of the I/O Image using a CNN embedder. We utilize a CNN pre-trained on the ImageNet dataset as a fixed, train-free feature extractor. The role of this embedder is to capture semantic graphical patterns and distill them into a compact one-dimensional vector of size *embed_dim*, which serves as the input for the anomaly detection model. The choice of the backbone model and the embedding dimension are configurable hyperparameters; in this work, we employ ResNet18 as the backbone and set the embedding dimension to 256. We evaluate the impact of these choices and select the optimal configuration in Section 5.2.2. Following feature extraction, one-class anomaly detection is conducted using the same methodology as `Inter-Mon`. As demonstrated in Section 5.2.2, this approach exhibits model-agnostic robustness, highlighting the superiority of our feature representation. Consistent with `Inter-Mon`, we primarily adopt the Deep SVDD model for evaluation in this work due to its efficiency and performance.

5 Evaluation

In this section, we conduct a comprehensive evaluation to answer the following key questions:

Q1. Effectiveness: How effectively does ScaleMon's multi-level approach detect a wide range of attacks?
Q2. Justification: Are our core design choices superior to reasonable alternatives?
Q3. Practicality: Is ScaleMon lightweight and scalable enough for real-world HPC deployment?

5.1 Experimental Setup

5.1.1 Datasets

To evaluate ScaleMon, we utilized two complementary datasets collected from the large-scale production HPC systems detailed in Table 5: a **Production Trace Dataset** and a **Fine-Grained I/O Dataset**. The Production Trace Dataset, containing standard Darshan logs, was used to specifically evaluate the inter-file level detection capabilities of Inter-Mon. The Fine-Grained I/O Dataset, containing detailed DXT traces, was used to evaluate the entire ScaleMon pipeline, including the image-based Identity Verifier and Intra-Mon.

Production Trace Dataset. The Production Trace Dataset consists of real-world Darshan logs collected from System A over one week in October 2023. This dataset captures benign I/O behavior from three widely used scientific simulation applications: VASP, GROMACS, and LAMMPS. To create anomalous samples for evaluation, we programmatically manipulated these benign logs to inject three types of inter-file level footprints, each corresponding to a different security goal:

- **Anomalous File Paths (Confidentiality):** We simulated a potential data breach by adding access to a file located at a high LCA distance (6–10) from other files in the execution.
- **Illegitimate Write Operations (Integrity):** We simulated data tampering by adding a write operation to a file that is normally read-only.
- **Anomalous File Counts (Availability):** We simulated a precursor to a Denial-of-Service attack by injecting a large number of spurious file accesses into a benign trace.

Table 3 provides a breakdown of the benign and anomalous samples in this dataset.

Fine-Grained I/O Dataset. The Fine-Grained I/O Dataset was generated on **System B** with DXT logging enabled. It comprises traces from h5bench_read and h5bench_write, VPIC-derived I/O benchmarks that mimic the application's real I/O interface, as well as from the LAMMPS ReaxFF simulation. Benign data were collected by running these applications under various configurations (1–8 nodes and 16–512 processes, using both independent and collective I/O) to capture diverse normal behaviors. We then synthesized attack samples by injecting three types of intra-file anomalous behavior through two distinct attack vectors (defined in Section 3.2). The specific anomalies are:

- **Full Sequential Scan (Confidentiality):** Simulating data exfiltration by performing a full sequential read of a file after normal I/O operations.
- **Full Sequential Overwrite (Integrity):** Simulating data tampering or destruction by performing a full sequential write over an entire file.
- **I/O Delay (Availability):** Simulating resource exploitation (e.g., cryptojacking) by introducing a significant pause in I/O activity while a hidden malicious process runs.

A summary of this dataset is presented in Table 4.

Table 3: Production Trace Dataset

Granularity	Dataset	Attack Type	VASP	GROMACS	LAMMPS
Accessed-File	Train	Benign (-)	96,596	9,400	43,806
Accessed-File	Test	Benign (-)	24,149	2,350	8,823
		Anomalous File Paths (Confidentiality)	15,309	1,746	5,511
		Illegitimate Operations (Integrity)	6,206	1,113	1,574
Execution	Train	Benign (-)	6,156	1,060	3,608
Execution	Test	Benign (-)	1,539	265	902
		Anomalous File Counts (Availability)	1,539	265	902

Table 4: Fine-Grained I/O Dataset

Dataset	Attack Type	h5bench_read	h5bench_write	LAMMPS_ReaxFF
Train	Benign (-)	345	389	186
Test	Benign (-)	87	97	46
	Full Sequential Scan (Confidentiality)	87	97	46
	Full Sequential Overwrite (Integrity)	87	97	46
	I/O Delay (Availability))	87	97	46

5.1.2 Implementation Details

All models were implemented using PyTorch and Scikit-learn, and were trained and evaluated on System B. Performance overheads were measured on a single GPU node using one NVIDIA A100 GPU (80GB). For classical one-class models, we used standard implementations with their default parameters. k in k-NN/LOF was heuristically set to the square root of the training set size, and the number of components for GMM was tested in a range from 1 to 8 with a diagonal covariance to prevent overfitting. Our deep learning models, including Autoencoders and Deep SVDD, were intentionally designed to be lightweight, consisting of only 4 to 6 fully connected layers with ReLU activations that compress the input to a latent space one-fourth of its original dimension. All deep learning models were trained for 50 epochs using the Adam optimizer.

Table 5: System Specifications.

Specification	System A		System B	
	KNL Node	CPU Node	GPU Node	CPU Node
Node Count	8,305	132	1,792	3,072
CPU	Intel Xeon Phi 7250 (68 cores)	2× Intel Xeon 6148 (20 cores)	1× AMD EPYC 7763 (64 cores)	2× AMD EPYC 7763 (64 cores)
Accelerator	-	-	4× NVIDIA A100 (40GB/80GB)	-
Memory	96GB DDR4	192GB DDR4	256GB DDR4	512GB DDR4
Interconnect	Intel OPA (100Gbps)		4× HPE Slingshot 11	1× HPE Slingshot 11
Storage (FS)	Lustre		Lustre	

5.1.3 Metrics

Our primary metric for detection performance is the **Area Under the ROC Curve (AUROC)**, a threshold-independent measure of a model's ability to distinguish between benign and anomalous samples. To provide insight into practical performance, we also report **Recall** (True Positive Rate) and **False Positive Rate (FPR)**. These threshold-dependent metrics are calculated using a fixed threshold for each model. Specifically, the threshold was set at the 99th percentile of the anomaly scores observed on the benign training set, corresponding to a theoretical 1% FPR on that data. We acknowledge that this standardized threshold may not be optimal for all scenarios and can be further tuned during operation. The broader implications and challenges of threshold selection are discussed in Section 6.

5.2 Effectivenes

5.2.1 Overall Detection Funnel

We evaluated `ScaleMon`'s end-to-end detection capability on a test set consisting of 46 benign executions and 276 synthesized attacks derived from the LAMMPS_ReaxFF application, spanning six attack types across inter-file and intra-file anomaly levels. As shown in Table 6, the detection funnel illustrates the hierarchical filtering process: the `Identity Verifier` first analyzed all attacks and immediately flagged 49 as anomalous due to mismatches between their claimed identity (bin file name in the Darshan log) and observed I/O behavior (I/O image). The remaining 227 attacks were then analyzed in parallel by application-specific `Inter-Mon` and `Intra-Mon` modules, which detected all remaining threats through inter-file and intra-file deviation analysis. Overall, `ScaleMon` successfully detected all 276 attacks with an FPR of 0.109, demonstrating the effectiveness of the proposed multi-stage design in providing comprehensive coverage across diverse attack types.

Table 6: Overall detection performance of `ScaleMon`.

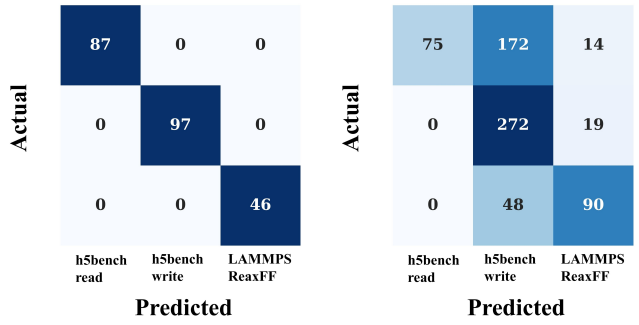
Detection Stage	# Detected	# Remaining	# False Positives
Initial State (Attacks)	–	276	–
Identity Verifier	49	227	0
Inter-Mon	138	89	0
Intra-Mon	89	138	5
Inter-Mon or Intra-Mon (Union)	227	0	5
ScaleMon (Total)	**276 (1.00)**	**0**	**5 (0.109)**

5.2.2 Component-wise Detection Performance

`Identity Verifier` We first evaluate the classification performance of the `Identity Verifier` on the Fine-Grained I/O Dataset. Figure 7 presents the confusion matrices for both benign and anomalous executions. As shown in Figure 7a, the verifier achieves perfect accuracy on benign data, correctly classifying every execution to its true application class. This result confirms that each application possesses a unique and stable I/O fingerprint that our model can effectively learn. In stark contrast, Figure 7b shows a significant number of misclassifications for the anomalous data. This is an expected and desirable outcome: the injected malicious patterns disrupt or dilute the application's intrinsic I/O fingerprint, causing the classifier to become uncertain or even assign the execution to the wrong class. The `Identity Verifier` leverages this misclassification as a powerful signal to detect masquerading or compromised executions at an early stage.

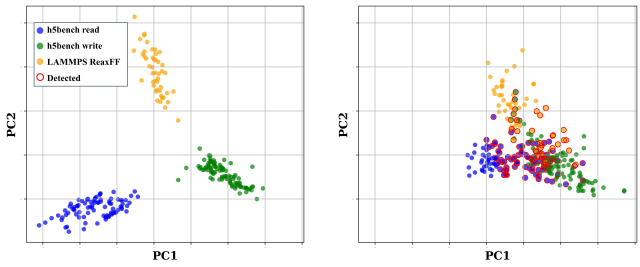
To provide a geometric intuition behind these classification results, Figure 8 visualizes the latent space embeddings from the final layer of our ResNet-18 classifier, reduced to two dimensions using PCA. The scatter plot of benign data (Figure 8a) clearly shows that embeddings from each application form tight, well-separated clusters. This visual evidence corroborates our core assumption that I/O images serve as highly discriminative fingerprints. Conversely, the embeddings from anomalous executions (Figure 8b) exhibit a collapsed cluster structure. Many of these points fall into ambiguous regions between the normal clusters, which corresponds to the misclassifications and low-confidence predictions observed in the confusion matrix. The red-bordered points in the figure, representing misclassified or low-confidence samples, are precisely those that the `Identity Verifier` flags as anomalous. This visualization confirms that our verifier effectively identifies attacks by recognizing when their behavioral fingerprints deviate from the clearly defined boundaries of normal behavior.

`Inter-Mon` Table 7 presents the AUROC scores for each model across all applications and inter-file attack types. The overall detection performance is exceptionally high, with most models achieving near-perfect AUROC scores (approaching 1.000) in the majority of scenarios. This consistently strong



(a) Normal Confusion Matrix (b) Anomaly Confusion Matrix

Figure 7: Classification Results of `Identity Verifier`



(a) Normal scatter plot (b) Anomaly scatter plot

Figure 8: Detection Performance of `Identity Verifier`

performance across diverse model architectures demonstrates the superior quality of our Execution Profile representation, which provides a rich, discriminative, and largely model-agnostic foundation for inter-file anomaly detection. This offers system administrators the flexibility to choose a model that best suits their operational needs. The main exception is Isolation Forest, which underperforms on Integrity (I) attacks. This is a known limitation of the algorithm; its axis-parallel partitioning struggles to detect anomalies defined by complex relationships between features [91]. Since the Integrity attack in our dataset is characterized by a subtle interplay between the operation type and file type features, Isolation Forest fails to effectively isolate it.

Table 8 details the practical performance of Deep SVDD, one of our top-performing models, using a fixed threshold set at the 99th percentile of the benign training data scores. The results are compelling. Except for the GROMACS Confidentiality scenario, the model achieves perfect recall (zero false negatives) for all other attack types while maintaining a low FPR of under 1.1%. For the GROMACS confidentiality threat, several anomalies are missed when using the 99th-percentile threshold. This behavior can be attributed to the inherent characteristics of GROMACS's benign I/O patterns, which occasionally exhibit larger LCA distances than those observed in other applications. As a result, the boundary between normal and anomalous behavior becomes less distinct.

Table 7: `Inter-Mon` AUROC across models

Model	VASP			GROMACS			LAMMPS		
	C	I	A	C	I	A	C	I	A
k-NN	1.000	0.998	1.000	0.994	0.981	1.000	1.000	1.000	1.000
LOF	0.995	1.000	0.971	0.969	0.999	0.965	0.987	0.991	0.999
GMM	1.000	0.813	1.000	1.000	0.999	1.000	1.000	1.000	1.000
Isolation Forest	0.991	0.791	0.969	0.956	0.479	0.826	0.986	0.761	0.973
One-Class SVM	1.000	0.996	1.000	0.939	0.666	1.000	1.000	0.644	1.000
Autoencoder	1.000	1.000	1.000	0.972	0.996	1.000	1.000	1.000	1.000
Deep SVDD	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Table 8: `Inter-Mon` detailed detection results.

Application	Threat	TP	TN	FP	FN	Recall	FPR
VASP	C	15309	23907	242	0	1.000	0.010
	I	6206	23907	242	0	1.000	0.010
	A	1539	1530	9	0	1.000	0.006
GROMACS	C	1741	2323	27	5	0.997	0.011
	I	1113	2323	27	0	1.000	0.011
	A	265	264	1	0	1.000	0.004
LAMMPS	C	5511	8788	35	0	1.000	0.004
	I	1574	8788	35	0	1.000	0.004
	A	902	893	9	0	1.000	0.010

Nevertheless, by lowering the detection threshold to the 95th percentile, perfect recall can also be achieved for this case, albeit at the cost of an increased false positive rate, which rises from 1.1% to 6.1%.

`Intra-Mon` Table 9 presents the AUROC performance of various one-class models for `Intra-Mon`, which leverages our I/O image representation. The results are outstanding, with most models achieving near-perfect AUROC scores on the `h5bench_read` workload and demonstrating strong performance across all other applications. While deep learning models like Autoencoder and DeepSVDD show excellent and robust performance (consistently above 0.97 AUROC), even classical methods such as k-NN and LOF perform remarkably well. This again confirms that our I/O image is a superior, highly discriminative representation that provides a largely model-agnostic foundation for intra-file anomaly detection. The performance of One-Class SVM and Isolation Forest degrades on more complex workloads, suggesting that their hyperplane or axis-parallel-based approaches are less suited for capturing the intricate, non-linear patterns present in our I/O image embeddings [28, 91].

Table 10 details the practical performance of Deep SVDD, selected for its robust performance, using a fixed threshold set at the 99th percentile of the benign training data scores. For the `h5bench` workloads, all attacks were successfully detected with a low FPR of 3.4%. Similarly, for `LAMMPS_ReaxFF`, all Confidentiality and Availability attacks were identified with only a single false alarm. A small number of false negatives were observed for `h5bench_write` across all attack types, and for the Integrity attack in `LAMMPS_ReaxFF`. However, we

Table 9: `Intra-Mon` AUROC across models

Model	h5bench read			h5bench write			LAMMPS ReaxFF		
	C	I	A	C	I	A	C	I	A
KNN	1.000	1.000	1.000	0.942	0.954	0.906	0.883	0.955	0.981
LOF	0.981	0.990	0.996	0.980	0.979	0.965	0.964	0.947	0.991
GMM	0.999	1.000	1.000	0.938	0.949	0.858	0.950	0.969	0.979
IsolationForest	0.999	1.000	1.000	0.880	0.888	0.767	0.958	0.965	0.982
OneClassSVM	0.909	0.962	0.967	0.853	0.900	0.522	0.657	0.893	0.966
Autoencoder	1.000	1.000	1.000	0.997	0.996	0.988	0.970	0.974	0.978
DeepSVDD	1.000	1.000	1.000	0.991	0.990	0.989	0.978	0.975	0.979

Table 10: `Intra-Mon` detailed detection results.

Application	Threat	TP	TN	FP	FN	Recall	FPR
h5bench read	C	87	84	3	0	1.000	0.034
	I	87	84	3	0	1.000	0.034
	A	87	84	3	0	1.000	0.034
h5bench write	C	96	87	10	1	0.990	0.103
	I	96	87	10	1	0.990	0.103
	A	96	87	10	1	0.990	0.103
LAMMPS ReaxFF	C	46	45	1	0	1.000	0.022
	I	42	45	1	4	0.913	0.022
	A	46	45	1	0	1.000	0.022

confirmed that these missed attacks could also be captured by lowering the detection threshold to the 95th percentile, which came at the cost of an increased FPR (from 10.3% to 19.6% for `h5bench_write` and from 2.2% to 10.9% for `LAMMPS_ReaxFF`). While these FPRs may seem rather high, they reflect the inherent trade-off in detecting extremely subtle anomalies, a challenge we discuss further in Section 6.

5.3 Justification

In this section, we provide a detailed justification for our core design choices, focusing on the Intra-Mon module which embodies our most novel contributions. We aim to empirically validate our I/O image representation and the specific model architecture chosen to analyze it.

5.3.1 The Superiority of the I/O Image Representation

To empirically validate our core design choice of using an I/O image representation, we compare it against strong baselines that directly process the raw, lengthy time-series data. For the baselines, we adopted a standard methodology for handling long sequences [**?**]: the I/O trace is segmented into fixed-size windows (e.g., 512 or 1024 tokens), and the maximum anomaly score from a sequence model (e.g., LSTM/Transformer Autoencoder) across all windows is taken as the final score. While these time-series baselines can achieve respectable performance in some scenarios (e.g., an AUROC of 0.984 for Integrity attacks in `h5bench read`), their effectiveness is inconsistent. Their performance often

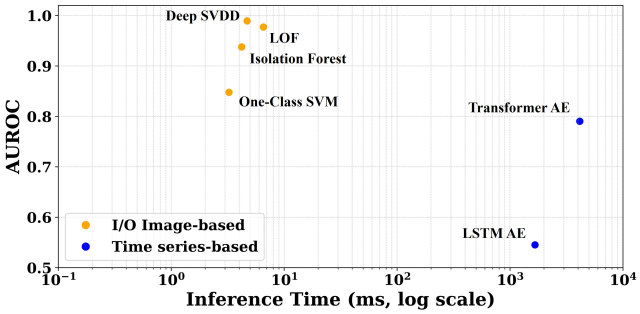


Figure 9: I/O Image vs Time series

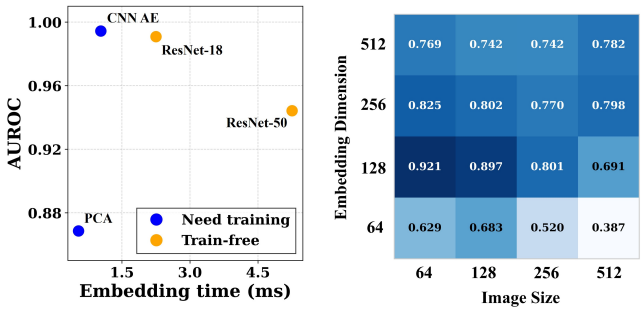
drops to near random chance (around 0.5 AUROC) when detecting attacks that require a global context, a direct consequence of their window-based, local-only view.

Figure 9 starkly illustrates the outcome, plotting the mean AUROC against mean inference time, averaged over all applications and attack types in our test set, for each approach. Models utilizing our I/O image representation exhibit both significantly higher mean accuracy and substantially lower inference times than the time-series baselines. This vast improvement stems from a fundamental advantage: by transforming the extremely long sequence into a single, fixed-size image, our approach holistically captures both local and global spatio-temporal patterns while drastically reducing the input data size.

5.3.2 Ablation Study of Model Components

The choice of embedder in `Intra-Mon` is critical, as it is responsible for capturing the semantic graphical patterns from I/O images, directly impacting both performance and overhead. Figure 10a compares the performance-overhead trade-off of several embedders. Both the custom-trained CNN Autoencoder and the pre-trained ResNet-18 demonstrate superior detection performance. While the CNN AE exhibits a slightly lower inference time, ResNet-18 offers a far more critical advantage: it is training-free. This eliminates the need for a separate, often lengthy, training phase for each application, making the entire framework more scalable and easier to deploy. Given that the inference overhead of ResNet-18 is still negligibly low (2.25 ms per image), its training-free nature makes it the clear choice. We further observed that increasing the backbone depth to ResNet-50 yielded no performance gains, suggesting that the essential patterns are sufficiently captured by the smaller ResNet-18 once projected into the target embedding dimension.

Next, we conducted a grid search to identify the optimal image size and embedding dimension. Figure 10b presents a heatmap of Recall − FPR (Youden's J statistic [94]), evaluated at the 99th-percentile threshold determined from the training data, for each image size–embedding dimension pair. The lower-right region of the heatmap shows lower scores in general, indicating that an embedding dimension that is too small relative to the I/O image size degrades detection performance



(a) Embedder comparison (b) Hyperparameter heatmap

Figure 10: Embedder and Hyperparameter Comparison

by introducing a representational bottleneck and discarding critical graphical information. Conversely, the upper region does not consistently yield higher scores, suggesting that simply increasing the embedding dimension does not necessarily improve detection performance. In practice, overly large embedding dimensions can lead to overfitting, increasing the distribution gap between training and test benign samples and thus raising the FPR. In our experiments, an image size of 64 with an embedding dimension of 128 provided a good balance. However, detecting highly subtle attack signatures may require larger images and higher embedding dimensions.

5.4 Practicality

A critical requirement for any security monitoring framework in HPC is minimal performance impact. In this section, we evaluate the practicality of `ScaleMon` by quantifying its computational overhead and analyzing its scalability with increasing data volume

5.4.1 Performance Overhead

We evaluated the performance overhead of `ScaleMon` on a System B GPU node, using a single NVIDIA A100 (40GB) GPU for all measurements. Table 11 summarizes the one-time training costs and the average per-execution detection overhead of each component. The total offline training time for all learning-based modules on the LAMMPS dataset is approximately 81 seconds, representing a modest one-time cost. During online detection, the end-to-end latency is dominated by the initial data processing stages. Specifically, the `Log Parser` incurs an average latency of 3.6 seconds per execution, primarily due to the overhead of invoking external Darshan parsing utilities, followed by 0.6 seconds for the `Fingerprint Generator`. In stark contrast, the core detection modules are highly efficient. The `Identity Verifier`, `Inter-Mon`, and `Intra-Mon` each require only 11–14 ms for inference, enabling low-latency monitoring. Furthermore, the peak GPU memory footprint during detection remains below 107 MB, demonstrating that `ScaleMon` imposes minimal pressure on critical accelerator resources.

Table 11: Performance overhead of `ScaleMon` components.

Component	Training Phase	Detection Phase (per Execution)	
	Total Time (s)	Avg. Time (ms)	Peak VRAM (MB)
`Log Parser`	N/A	3608.4	–
`Fingerprint Generator`	N/A	624.2	–
`Identity Verifier`	15.6	13.2	106.2
`Inter-Mon`	49.5	11.5	95.4
`Intra-Mon`	16.3	11.3	104.7
`ScaleMon`	**81.4**	**4268.7**	**106.2**

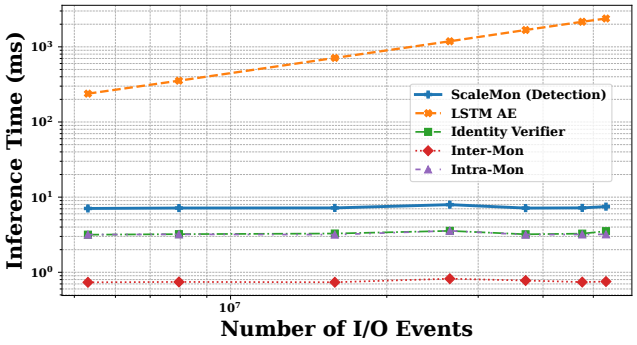


Figure 11: Scalability with Number of I/O Events

5.4.2 Scalability with Data Volume

We evaluated the scalability of `ScaleMon`'s detection modules (`Identity Verifier`, `Inter-Mon`, and `Intra-Mon`) by analyzing their inference latency across varying input log sizes. Figure 11 compares the processing time of `ScaleMon` against a LSTM-AE baseline. The results highlight a fundamental architectural advantage: while the baseline's overhead grows linearly with data volume due to the increasing number of sliding windows required for sequence analysis, `ScaleMon` exhibits a completely flat, constant-time performance curve ($O(1)$). This is achieved by our fingerprinting mechanism, which transforms the entire execution behavior into fixed-size profile vectors and I/O images. Consequently, the computational complexity of anomaly detection becomes effectively decoupled from the raw volume of the I/O log.

Furthermore, `ScaleMon` ensures scalability regarding the number of accessed files. While input volume scales linearly with file count, we leverage batch processing to infer fingerprints in parallel. Our experiments demonstrate the efficacy of this design: despite a 256× increase in concurrent files (from 1 to 256), the inference latency increased by factors of only 1.89× for the `Identity Verifier`, 1.36× for `Inter-Mon`, and 1.41× for `Intra-Mon` (total 1.62×). This significantly sub-linear growth confirms that `ScaleMon`'s parallelized architecture maintains predictable performance, preventing bottlenecks even in massive-scale scenarios involving hundreds of files.

6 Discussion and Limitations

While ScaleMon demonstrates promising results in detecting I/O-based anomalies with negligible overhead, we acknowledge limitations inherent to our design choices and experimental constraints.

Spatio-Temporal Resolution in I/O Image Compression Our approach converts variable-length I/O logs into fixed-size image tensors to efficiently capture both local and global patterns with $O(1)$ inference complexity. However, we acknowledge a trade-off in this compression: attacks manifesting as extremely short-lived temporal signals or modifications to a negligible number of bytes might be obscured during pixel binning. In such specific cases, analyzing the non-compressed raw data could be needed, and we leave the exploration of such a hybrid approach to future work.

False Positive Rate and Thresholding In our evaluation, the Intra-Mon module exhibited a False Positive Rate (FPR) of up to 19.6% in the worst-case scenario when calibrated to detect all attacks. We acknowledge that this rate could lead to alert fatigue in production environments. However, this result is largely a consequence of the limited training dataset used in our experiments, which failed to encompass the full diversity of benign behaviors encountered in the test set. In a real-world deployment, the repetitive nature of HPC workloads would provide a massive volume of benign execution logs, enabling the model to learn more generalized patterns and significantly improve robustness [1]. Furthermore, while this work utilized a simple static threshold, implementing advanced strategies—such as dynamic thresholding based on a separate validation set—can effectively mitigate false alarms without compromising detection capability. [29].

Reliance on Synthetic Data A significant challenge in HPC security is the absence of publicly available, labeled datasets containing real-world attacks. To address this, we rigorously generated a custom dataset by executing real scientific applications and benchmarks (e.g., LAMMPS, h5bench) under various configurations and systematically injecting attack patterns based on a comprehensive threat model. Although we strove to mimic realistic behaviors, we acknowledge that synthetic anomalies may not perfectly replicate the complexity of attacks orchestrated by sophisticated adversaries. Nevertheless, since ScaleMon adopts a one-class classification approach trained solely on benign behavior, our evaluation successfully demonstrates the system's fundamental capability to identify semantic deviations from benign patterns, regardless of the specific attack nuances. Furthermore, by open-sourcing our dataset, we aim to contribute a valuable resource to the community, fostering reproducibility and facilitating future research in HPC security. delete last sentece?

7 Related Works

Security in HPC.

Logging in large-scale systems. With the focus on low-performance overhead and scalability, many logging tools are designed and deployed in many leadership-scale HPCs. These tools can be categorized into three categories. 1) OS-native system log using *Syslog* [65, 95], 2) Focusesing on idivisual system resource or state inlcuding I/O (e.g., Darshan [79], Recorder [85], Reflector [2]) , Network (e.g., Bro [80]), and Job scheduling information (e.g., SLURM log [93], PBS log [5], and 3) Integrating distributed logs to create comprehensive overview [49, 67]. Our paper aligns with these studies by utilizing efficiently collected system logs for in-depth analysis. In contrast, we focus on detecting system anomalies through both efficient log collection and analysis. We achieve this by transforming large datasets into compact images that are representative enough to detect system and application anomalies induced by attacks, while maintaining scalability in large HPC systems.

Log-based HPC analysis and optimization. Many studies have attempted to analyze HPC systems using collected logs and optimize system performance. Other studies focused on utilizing logs to predict performance [49, 81] and analyze the main causes of bottlenecks [86, 92]. In terms of security, few studies have tried to learn from logs [22, 41, 64, 74]. However, these studies mainly focused on static application code analysis [22], encryption-based data security [41, 64], and network breach detection [74]. While some works [15, 61, 82, 87] have attempted to apply machine learning-based log anomaly detection in HPC, they either: 1) focused on specific subsets of logs, such as microservice logs [43, 61, 82], or 2) tested on small subsets or static analyses with test settings and previously accumulated test logs [15, 87]. Our paper aligns with these studies by detecting anomalies in system logs through log analysis. However, our study focuses on the strong high-performance guarantees in HPC systems to provide performance-efficient and scalable log processing and detection through image transformation and anomaly detection, enabling the processing of extremely large, often exascale, HPC logs.

8 Conclusion

In this paper, we presented ScaleMon, the first security framework to utilize widespread, HPC-native I/O logging tools for intrusion detection. ScaleMon treats I/O behavior as a unique fingerprint of application execution, providing deep, multi-level visibility with minimal overhead. By transforming raw I/O traces into lightweight fingerprints and images, our system effectively identifies anomalous behaviors at both inter-file and intra-file levels. Evaluation on production workloads shows that ScaleMon detects diverse attack vectors while meeting the strict performance requirements of HPC environments. Our work demonstrates that leveraging existing HPC I/O telemetry is a practical and powerful approach to protecting HPC systems against compromised applications.

References

[1] Al Jallad, K., Aljnidi, M., and Desouki, M. S. Anomaly detection optimization using big data and deep learning to reduce false-positive. *Journal of Big Data 7*, 1 (2020), 68.

[2] Al-Mamun, A., Liu, J., Li, T., Koziol, Q., Zhai, Z., Qian, J., Shen, H., and Zhao, D. Reflector: a fine-grained i/o tracker for hpc systems. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (2020), pp. 427–428.

[3] Aloul, F., Zahidi, S., and El-Hajj, W. Two factor authentication using mobile phones. In *2009 IEEE/ACS international conference on computer systems and applications* (2009), IEEE, pp. 641–644.

[4] ALSaleem, B. O., and Alshoshan, A. I. Multi-factor authentication to systems login. In *2021 National Computing Colleges Conference (NCCC)* (2021), IEEE, pp. 1–4.

[5] Altair Engineering Inc. *PBS Professional 2021.1.2 Administrator's Guide*, 2021. Version 2021.1, Available at: https://www.altair.com/pbs-professional/.

[6] Bhargava, R., and Troester, K. Amd next generation" zen 4" core and 4 th gen amd epyc™ server cpus. *IEEE Micro* (2024).

[7] Bilot, T., Jiang, B., Li, Z., El Madhoun, N., Al Agha, K., Zouaoui, A., and Pasquier, T. Sometimes simpler is better: A comprehensive analysis of {State-of-the-Art}{Provenance-Based} intrusion detection systems. In *34th USENIX Security Symposium (USENIX Security 25)* (2025), pp. 7193–7212.

[8] Burford, A., Calder, A., Carlson, D., Chapman, B., Coskun, F., Curtis, T., Feldman, C., Harrison, R., Kang, Y., Michalowicz, B., et al. Ookami: Deployment and initial experiences. In *Practice and Experience in Advanced Research Computing 2021: Evolution Across All Dimensions*. 2021, pp. 1–8.

[9] Cao, P. Jupyter notebook attacks taxonomy: Ransomware, data exfiltration, and security misconfiguration. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2024), IEEE, pp. 750–754.

[10] Cao, P., Kalbarczyk, Z., and Iyer, R. K. Security testbed for preempting attacks against supercomputing infrastructure. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2024), IEEE, pp. 1781–1788.

[11] Cappello, F., Constantinescu, E., Hovland, P., Peterka, T., Phillips, C., Snir, M., and Wild, S. Improving the trust in results of numerical simulations and scientific data analytics. Tech. rep., Argonne National Lab.(ANL), Argonne, IL (United States), 2015.

[12] Cheng, Z., Lv, Q., Liang, J., Wang, Y., Sun, D., Pasquier, T., and Han, X. Kairos: Practical intrusion detection and investigation using whole-system provenance. In *2024 IEEE Symposium on Security and Privacy (SP)* (2024), IEEE, pp. 3533–3551.

[13] Choquette, J., Gandhi, W., Giroux, O., Stam, N., and Krashinsky, R. Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro 41*, 2 (2021), 29–35.

[14] Chunduri, S., Parker, S., Balaji, P., Harms, K., and Kumaran, K. Characterization of mpi usage on a production supercomputer. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis* (2018), IEEE, pp. 386–400.

[15] Cinque, M., Della Corte, R., and Pecchia, A. Micro2vec: Anomaly detection in microservices systems by mining numeric representations of computer logs. *Journal of Network and Computer Applications 208* (2022), 103515.

[16] Copos, B., and Peisert, S. Catch me if you can: Using power analysis to identify hpc activity. *arXiv preprint arXiv:2005.03135* (2020).

[17] Costan, V., and Devadas, S. Intel sgx explained. *Cryptology ePrint Archive* (2016).

[18] Crichigno, J., Bou-Harb, E., and Ghani, N. A comprehensive tutorial on science dmz. *IEEE Communications Surveys & Tutorials 21*, 2 (2018), 2041–2078.

[19] Dai, D., Chen, Y., Carns, P., Jenkins, J., and Ross, R. Lightweight provenance service for high-performance computing. In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)* (2017), IEEE, pp. 117–129.

[20] Dart, E., Rotman, L., Tierney, B., Hester, M., and Zurawski, J. The science dmz: A network design pattern for data-intensive science. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (2013), pp. 1–10.

[21] De Sensi, D., Di Girolamo, S., McMahon, K. H., Roweth, D., and Hoefler, T. An in-depth analysis of the slingshot interconnect. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (2020), IEEE, pp. 1–14.

[22] DeMasi, O., Samak, T., and Bailey, D. H. Identifying hpc codes via performance logs and machine learning. In *Proceedings of the first workshop on Changing landscapes in HPC security* (2013), pp. 23–30.

[23] Dhanuskodi, G., Guha, S., Krishnan, V., Manjunatha, A., O'Connor, M., Nertney, R., and Rogers, P. Creating the first confidential gpus: The team at nvidia brings confidentiality and integrity to user code and data for accelerated computing. *Queue 21*, 4 (2023), 68–93.

[24] Di Pietro, R., Lombardi, F., and Villani, A. Cuda leaks: Information leakage in gpu architectures. *arXiv preprint arXiv:1305.7383* (2013).

[25] Du, M., Li, F., Zheng, G., and Srikumar, V. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security* (2017), pp. 1285–1298.

[26] Elia, R., Granata, D., Rak, M., et al. Systematic threat modelling of high-performance computing systems: The v: Hpccri case study. In *CLOSER* (2024), pp. 327–337.

[27] Elwasif, W., Godoy, W., Hagerty, N., Harris, J. A., Hernandez, O., Joo, B., Kent, P., Lebrun-Grandie, D., Maccarthy, E., Melesse Vergara, V., et al. Application experiences on a gpu-accelerated arm-based hpc testbed. In *Proceedings of the HPC Asia 2023 Workshops* (2023), pp. 35–49.

[28] Erfani, S. M., Rajasegarar, S., Karunasekera, S., and Leckie, C. High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning. *Pattern Recognition 58* (2016), 121–134.

[29] Ghafouri, A., Abbas, W., Laszka, A., Vorobeychik, Y., and Koutsoukos, X. Optimal thresholds for anomaly-based intrusion detection in dynamical environments. In *International Conference on Decision and Game Theory for Security* (2016), Springer, pp. 415–434.

[30] Giannakou, A., Hazen, D., Enders, B., Ramakrishnan, L., and Wright, N. J. Understanding data movement patterns in hpc: A nersc case study. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis* (2024), IEEE, pp. 1–17.

[31] Gomes, F., and Correia, M. Cryptojacking detection with cpu usage metrics. In *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)* (2020), IEEE, pp. 1–10.

[32] Grooms, X., Rollins, R., and Rumpca, C. sflow monitoring for security and reliability. Tech. rep., Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), 2019.

[33] Guo, Y., Chandramouli, R., Wofford, L., Gregg, R., Key, G., Clark, A., Hinton, C., Prout, A., Reuther, A., Adamson, R., et al. High-performance computing (hpc) security: Architecture, threat analysis, and security posture. Tech. rep., National Institute of Standards and Technology, 2023.

[34] GUO, Y., CHANDRAMOULI, R., WOFFORD, L., GREGG, R., KEY, G., CLARK, A., HINTON, C., PROUT, A., REUTHER, A., ADAMSON, R., ET AL. High-performance computing security architecture, threat analysis, and security posture.

[35] GUZ, Z., LI, H., SHAYESTEH, A., AND BALAKRISHNAN, V. Nvme-over-fabrics performance characterization and the path to low-overhead flash disaggregation. In *Proceedings of the 10th ACM International Systems and Storage Conference* (2017), pp. 1–9.

[36] HAN, R., BYNA, S., TANG, H., DONG, B., AND ZHENG, M. Prov-io: An i/o-centric provenance framework for scientific data on hpc systems. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing* (2022), pp. 213–226.

[37] HAN, R., ZHENG, M., BYNA, S., TANG, H., DONG, B., DAI, D., CHEN, Y., KIM, D., HASSOUN, J., AND THORSLEY, D. Prov-io ϴI++: A cross-platform provenance framework for scientific data on hpc systems. *IEEE Transactions on Parallel and Distributed Systems 35*, 5 (2024), 844–861.

[38] HE, Z., HU, G., AND LEE, R. New models for understanding and reasoning about speculative execution attacks. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (2021), IEEE, pp. 40–53.

[39] HER, J., KIM, J., KIM, J., AND LEE, S. An in-depth analysis of ebpf-based system security tools in cloud-native environments. *IEEE Access* (2025).

[40] HERTEN, A. Many cores, many models: Gpu programming model vs. vendor compatibility overview. In *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis* (2023), pp. 1019–1026.

[41] HICKMAN, M., FULP, D., BASEMAN, E., BLANCHARD, S., GREENBERG, H., JONES, W., AND DEBARDELEBEN, N. Enhancing hpc system log analysis by identifying message origin in source code. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)* (2018), IEEE, pp. 100–105.

[42] HOSEK, P., AND CADAR, C. Varan the unbelievable: An efficient n-version execution framework. *ACM SIGARCH Computer Architecture News 43*, 1 (2015), 339–353.

[43] HUANG, H., LUO, W., WANG, Y., ZHOU, Y., AND HUANG, W. Logctbl: a hybrid deep learning model for log-based anomaly detection. *The Journal of Supercomputing 81*, 2 (2025), 448.

[44] ISLAM, N. S., LU, X., WASI-UR RAHMAN, M., SHANKAR, D., AND PANDA, D. K. Triple-h: A hybrid approach to accelerate hdfs on hpc clusters with heterogeneous storage architecture. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (2015), IEEE, pp. 101–110.

[45] JIA, Z., XIONG, Y., NAN, Y., ZHANG, Y., ZHAO, J., AND WEN, M. {MAGIC}: Detecting advanced persistent threats via masked graph representation learning. In *33rd USENIX Security Symposium (USENIX Security 24)* (2024), pp. 5197–5214.

[46] JIANG, B., BILOT, T., EL MADHOUN, N., AL AGHA, K., ZOUAOUI, A., IQBAL, S., HAN, X., AND PASQUIER, T. Orthrus: Achieving high quality of attribution in provenance-based intrusion detection systems. In *Security Symposium (USENIX Sec'25). USENIX* (2025).

[47] JUMPER, J., EVANS, R., PRITZEL, A., GREEN, T., FIGURNOV, M., RONNEBERGER, O., TUNYASUVUNAKOOL, K., BATES, R., ŽÍDEK, A., POTAPENKO, A., ET AL. Highly accurate protein structure prediction with alphafold. *nature 596*, 7873 (2021), 583–589.

[48] KAPLAN, L., AND HARRELL, J. Cray compute node linux. *Operating Systems for Supercomputers and High Performance Computing* (2019), 99–120.

[49] KIM, S., SIM, A., WU, K., BYNA, S., SON, Y., AND EOM, H. Towards hpc i/o performance prediction through large-scale log analysis. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing* (2020), pp. 77–88.

[50] KO, C., RUSCHITZKA, M., AND LEVITT, K. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In *Proceedings. 1997 IEEE symposium on security and privacy (Cat. No. 97CB36097)* (1997), IEEE, pp. 175–187.

[51] KOLEINI, S., SHIRAZ, I., AND PAHLEVANZADEH, B. High performance computing (hpc) system security: Threats and vulnerabilities, challenges and solutions.

[52] KONING, R., BURAGLIO, N., DE LAAT, C., AND GROSSO, P. Coreflow: Enriching bro security events using network traffic monitoring data. *Future Generation Computer Systems 79* (2018), 235–242.

[53] KORANGA, R. S., RAI, A. K., KUMAR, S., AND KUMAR, H. Comparative analysis of ml algorithms for detecting intrusion using the nsl-kdd dataset. In *2025 3rd International Conference on Disruptive Technologies (ICDT)* (2025), IEEE, pp. 763–766.

[54] KUVAISKII, D., STAVRAKAKIS, D., QIN, K., XING, C., BHATOTIA, P., AND VIJ, M. Gramine-tdx: A lightweight os kernel for confidential vms. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (2024), pp. 4598–4612.

[55] LADISA, P., PLATE, H., MARTINEZ, M., AND BARAIS, O. Sok: Taxonomy of attacks on open-source software supply chains. In *2023 IEEE Symposium on Security and Privacy (SP)* (2023), IEEE, pp. 1509–1526.

[56] LAGUNA, I., MARSHALL, R., MOHROR, K., RUEFENACHT, M., SKJELLUM, A., AND SULTANA, N. A large-scale study of mpi usage in open-source hpc applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2019), pp. 1–14.

[57] LANGE, J. R., GORDON, N., AND GAINES, B. Low overhead security isolation using lightweight kernels and tees. In *2021 SC Workshops Supplementary Proceedings (SCWS)* (2021), IEEE, pp. 42–49.

[58] LEAL, E. A., WRIGHT-HAMOR, C., MANZANO, J. B., MULTARI, N. J., BARKER, K. J., MANZ, D. O., AND MING, J. Assessing risk in high performance computing attacks. In *ICISSP* (2023), pp. 793–803.

[59] LEE, J.-K., HONG, T., AND LI, G. Traffic and overhead analysis of applied pre-filtering acl firewall on hpc service network. *Journal of Communications and Networks 23*, 3 (2021), 192–200.

[60] LI, Y., MIAO, R., KIM, C., AND YU, M. {FlowRadar}: A better {NetFlow} for data centers. In *13th USENIX symposium on networked systems design and implementation (NSDI 16)* (2016), pp. 311–324.

[61] LUO, Z., HOU, T., NGUYEN, T. T., ZENG, H., AND LU, Z. Log analytics in hpc: A data-driven reinforcement learning framework. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (2020), IEEE, pp. 550–555.

[62] MEHNAZ, S., AND BERTINO, E. Ghostbuster: A fine-grained approach for anomaly detection in file system accesses. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy* (2017), pp. 3–14.

[63] MILUTINOVIĆ, V., AZER, E. S., YOSHIMOTO, K., KLIMECK, G., DJORDJEVIC, M., KOTLAR, M., BOJOVIC, M., MILADINOVIC, B., KOROLIJA, N., STANKOVIC, S., ET AL. The ultimate dataflow for ultimate supercomputers-on-a-chip, for scientific computing, geo physics, complex mathematics, and information processing. In *2021 10th Mediterranean Conference on Embedded Computing (MECO)* (2021), IEEE, pp. 1–6.

[64] NOLTE, H., SPICHER, N., RUSSEL, A., EHLERS, T., KREY, S., KREFTING, D., AND KUNKEL, J. Secure hpc: A workflow providing a secure partition on an hpc system. *Future Generation Computer Systems 141* (2023), 677–691.

[65] OLINER, A., AND STEARLEY, J. What supercomputers say: A study of five system logs. In *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)* (2007), IEEE, pp. 575–584.

[66] OSBORN, S. Mandatory access control and role-based access control revisited. In *Proceedings of the second ACM workshop on Role-based access control* (1997), pp. 31–40.

[67] PARK, B. H., HUKERIKAR, S., ADAMSON, R., AND ENGELMANN, C. Big data meets hpc log analytics: Scalable approach to understanding systems at extreme scale. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)* (2017), IEEE, pp. 758–765.

[68] PARK, J.-W., WOO, J., AND HONG, T. Myksc: Disaggregated containerized supercomputer platform. In *International Conference on Web Services* (2023), Springer, pp. 83–91.

[69] PASQUIER, T., HAN, X., GOLDSTEIN, M., MOYER, T., EYERS, D., SELTZER, M., AND BACON, J. Practical whole-system provenance capture. In *Proceedings of the 2017 symposium on cloud computing* (2017), pp. 405–418.

[70] PAXSON, V. Bro: a system for detecting network intruders in real-time. *Computer networks 31*, 23-24 (1999), 2435–2463.

[71] PEISERT, S. Security in high-performance computing environments. *Communications of the ACM 60*, 9 (2017), 72–80.

[72] PFISTER, G. F. An introduction to the infiniband architecture. *High performance mass storage and parallel I/O 42*, 617-632 (2001), 102.

[73] PINTO, S., AND SANTOS, N. Demystifying arm trustzone: A comprehensive survey. *ACM computing surveys (CSUR) 51*, 6 (2019), 1–36.

[74] PROUT, A., ARCAND, W., BESTOR, D., BERGERON, B., BYUN, C., GADEPALLY, V., HUBBELL, M., HOULE, M., JONES, M., MICHALEAS, P., ET AL. Enhancing hpc security with a user-based firewall. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)* (2016), IEEE, pp. 1–4.

[75] REHMAN, M. U., AHMADI, H., AND HASSAN, W. U. Flash: A comprehensive approach to intrusion detection via provenance graph representation learning. In *2024 IEEE Symposium on Security and Privacy (SP)* (2024), IEEE, pp. 3552–3570.

[76] SANDHU, R., AND MUNAWER, Q. How to do discretionary access control using roles. In *Proceedings of the third ACM workshop on Role-based access control* (1998), pp. 47–54.

[77] SEV-SNP, A. Strengthening vm isolation with integrity protection and more. *White Paper, January 53*, 2020 (2020), 1450–1465.

[78] SINHA, P., GULIANI, A., JAIN, R., TRAN, B., SINCLAIR, M. D., AND VENKATARAMAN, S. Not all gpus are created equal: characterizing variability in large-scale, accelerator-rich systems. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis* (2022), IEEE, pp. 01–15.

[79] SNYDER, S., CARNS, P., HARMS, K., ROSS, R., LOCKWOOD, G. K., AND WRIGHT, N. J. Modular hpc i/o characterization with darshan. In *2016 5th workshop on extreme-scale programming tools (ESPT)* (2016), IEEE, pp. 9–17.

[80] SOMMER, R. Bro: An open source network intrusion detection system. In *Security, E-learning, E-Services, 17. DFN-Arbeitstagung über Kommunikationsnetze* (2003), Gesellschaft für Informatik eV, pp. 273–288.

[81] SUNG, D. K., SON, Y., SIM, A., WU, K., BYNA, S., TANG, H., EOM, H., KIM, C., AND KIM, S. A2fl: autonomous and adaptive file layout in hpc through real-time access pattern analysis. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2024), IEEE, pp. 506–518.

[82] TAN, R., AND LI, Z. Maad: A distributed anomaly detection architecture for microservices systems. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2024), IEEE, pp. 1009–1021.

[83] VAN DER SPOEL, D., LINDAHL, E., HESS, B., GROENHOF, G., MARK, A. E., AND BERENDSEN, H. J. Gromacs: fast, flexible, and free. *Journal of computational chemistry 26*, 16 (2005), 1701–1718.

[84] WALLACE, D. Compute node linux: New frontiers in compute node operating systems. *Cray User Group* (2007).

[85] WANG, C., SUN, J., SNIR, M., MOHROR, K., AND GONSIOROWSKI, E. Recorder 2.0: Efficient parallel i/o tracing and analysis. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2020), IEEE, pp. 1–8.

[86] WANG, T., BYNA, S., LOCKWOOD, G. K., SNYDER, S., CARNS, P., KIM, S., AND WRIGHT, N. J. A zoom-in analysis of i/o logs to detect root causes of i/o performance bottlenecks. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)* (2019), IEEE, pp. 102–111.

[87] WANG, X., CAO, Q., WANG, Q., CAO, Z., ZHANG, X., AND WANG, P. Robust log anomaly detection based on contrastive learning and multi-scale mass. *The Journal of Supercomputing 78*, 16 (2022), 17491–17512.

[88] WHALEN, S., PEISERT, S., AND BISHOP, M. Multiclass classification of distributed memory parallel computations. *Pattern Recognition Letters 34*, 3 (2013), 322–329.

[89] WISNIEWSKI, R. W., INGLETT, T., KEPPEL, P., MURTY, R., AND RIESEN, R. mos: An architecture for extreme-scale operating systems. In *Proceedings of the 4th International Workshop on Runtime and Operating Systems for Supercomputers* (2014), pp. 1–8.

[90] XU, C., SNYDER, S., VENKATESAN, V., CARNS, P., KULKARNI, O., BYNA, S., SISNEROS, R., AND CHADALAVADA, K. Dxt: Darshan extended tracing. Tech. rep., Argonne National Lab.(ANL), Argonne, IL (United States), 2017.

[91] XU, H., PANG, G., WANG, Y., AND WANG, Y. Deep isolation forest for anomaly detection. *IEEE Transactions on Knowledge and Data Engineering 35*, 12 (2023), 12591–12604.

[92] YANG, B., WEI, H., ZHU, W., ZHANG, Y., LIU, W., AND XUE, W. Full lifecycle data analysis on a large-scale and leadership supercomputer: what can we learn from it? In *2024 USENIX Annual Technical Conference (USENIX ATC 24)* (2024), pp. 917–933.

[93] YOO, A. B., JETTE, M. A., AND GRONDONA, M. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing* (2003), Springer, pp. 44–60.

[94] YOUDEN, W. J. Index for rating diagnostic tests. *Cancer 3*, 1 (1950), 32–35.

[95] ZHANG, S., LIU, Y., MENG, W., BU, J., YANG, S., SUN, Y., PEI, D., XU, J., ZHANG, Y., SONG, L., ET AL. Efficient and robust syslog parsing for network devices in datacenter networks. *IEEE access 8* (2020), 30245–30261.

[96] ZHANG, W., ALMGREN, A., BECKNER, V., BELL, J., BLASCHKE, J., CHAN, C., DAY, M., FRIESEN, B., GOTT, K., GRAVES, D., ET AL. Amrex: a framework for block-structured adaptive mesh refinement. *The Journal of Open Source Software 4*, 37 (2019), 1370.

[97] ZHOU, H., AND GU, G. Cerberus: Enabling efficient and effective in-network monitoring on programmable switches. In *2024 IEEE Symposium on Security and Privacy (SP)* (2024), IEEE, pp. 4424–4439.

[98] ZHU, Z., GU, R., PAN, C., LI, Y., ZHU, B., AND LI, J. Cpu and network traffic anomaly detection method for cloud data center. In *Proceedings of the 1st International Conference on Advanced Information Science and System* (2019), pp. 1–7.