DeepVis: Deep Learning-Based File System Fingerprinting for Cloud Security

***Abstract*—This paper presents `DeepVis`, a high-throughput integrity verification system designed to improve scalability and reduce overhead in hyperscale storage environments. Our key idea is to leverage a spatial hash projection architecture, enabling highly parallelized metadata processing while maintaining detection accuracy via a stable tensor representation. Specifically, `DeepVis` first introduces an asynchronous snapshot engine that leverages high-performance I/O interfaces to maximize ingestion rates, allowing the system to rapidly capture file system states. Second, `DeepVis` devises a lock-free tensor mapping pipeline, where metadata processing is sharded across processor cores to eliminate contention and achieve linear scalability. Finally, `DeepVis` adopts a spatial anomaly detection approach, enabling the identification of sparse attack signals even amidst significant background noise caused by legitimate system updates. We implement `DeepVis` with these three techniques and evaluate its performance on production-grade cloud infrastructure, scaling up to 100 VMs across multiple regions. Our evaluation results show that `DeepVis` improves verification throughput by 10.9× compared with optimized integrity monitors under cold cache conditions, while maintaining 97.1% recall on active threats with a 0.3% false positive rate and negligible runtime overhead (CPU impact $< 2\%$).**

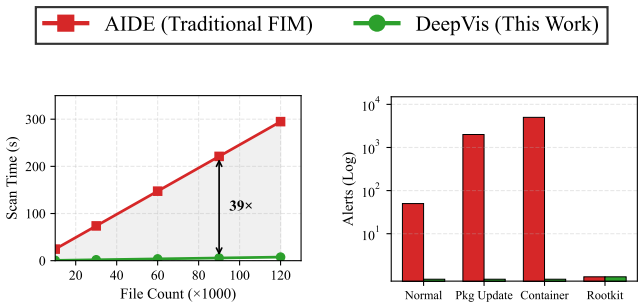
***Index Terms*—Distributed Systems, File System Monitoring, Scalable Verification, Anomaly Detection, Spatial Representation Learning**

I. Introduction

Cloud computing provides a computational model distinct from traditional on-premise environments by abstracting physical infrastructure into dynamic, ephemeral resources. From container orchestration platforms such as Kubernetes to large-scale HPC clusters, ensuring the integrity of workloads is a foundational requirement. Operators must guarantee that the file systems of thousands of nodes remain free from unauthorized modifications. However, modern DevOps practices create a fundamental tension between security and agility. Frequent deployments and updates generate massive file churn, rendering traditional security models obsolete.

To address this, two primary strategies are commonly used: File Integrity Monitoring (FIM) and Runtime Behavioral Analysis. FIM tools such as AIDE [1] and Tripwire [2] rely on cryptographic hashing to detect static changes, providing strong integrity guarantees. Conversely, runtime monitors such as Falco [3] and OSSEC [4] trace system calls to detect anomalous execution. Our work focuses on static integrity verification, as preserving the baseline state is essential for detecting dormant threats and performing post-incident forensics.

However, traditional integrity verification faces a fundamental scalability challenge. As the number of files (N) grows, the scan latency increases linearly ($O(N)$), causing severe I/O bottlenecks in hyperscale storage. This is problematic because modern cloud instances, despite high CPU throughput, have limited storage bandwidth. For example, scanning a filesystem with millions of small files using synchronous system calls results in excessive context switching and blocking I/O. Beyond the performance cost, the "Alert Fatigue" problem further limits usability: legitimate updates generate thousands of false positives, masking true threats [5]. Thus, the operational cost exceeds the theoretical benefit, forcing operators to disable monitoring during maintenance windows.



(a) Scalability (b) Alert Fatigue

Fig. 1. (a) DeepVis achieves high-throughput saturation via async I/O. (b) Legitimate updates generate false alerts in AIDE.

Figure 1 compares the scalability and precision of `DeepVis` against AIDE on a GCP production instance. As depicted in Figure 1(a), AIDE scan time increases linearly with a steep slope, exceeding 4 minutes for just 100K files. In contrast, `DeepVis` leverages a parallelized asynchronous pipeline to saturate storage bandwidth, keeping scan times under 7 seconds for the same dataset. While file ingestion remains physically $O(N)$, `DeepVis` achieves an effective speedup of up to 39× against traditional full-hash baselines (and 11× against optimized header scanners), and crucially, ensures that the subsequent anomaly detection (Inference) remains constant regardless of dataset size.

Many previous studies, as summarized in Table I, have explored works to enhance the scalability of system monitoring. Several works [1], [2] focus on cryptographic exactness but suffer from $O(N)$ scalability limits. Runtime approaches [3], [7] utilize eBPF or provenance graphs to detect zero-day threats but incur continuous runtime overhead (5–20%) and cannot detect dormant artifacts. Deep learning-based approaches, such as Set-AE [8], attempt to learn system states but fail to detect sparse anomalies due to signal dilution in

TABLE I
COMPARISON WITH PRIOR WORK ACROSS FOUR KEY CAPABILITIES: ASYNCHRONOUS I/O (ASYNC), OBFUSCATION RESILIENCE (OBFUSC.), ZERO-DAY DETECTION (0-DAY), AND LOW OVERHEAD (LOW OVHD.).

Study	Approach	Asnc	Obfs.	0-Day	Low
AIDE [1]	Full-Hash FIM		✓		
Tripwire [2]	Full-Hash FIM		✓		
ClamAV [6]	Signature Scanning				✓
Falco [3]	Runtime/eBPF	✓		✓	
Unicorn [7]	Provenance Graph		✓	✓	
OSSEC [4]	Log Analysis				✓
Set-AE [8]	Deep Sets Learning	✓	✓	✓	✓
DeepVis	**Hash-Grid Tensor**	✓	✓	✓	✓

global pooling.

DeepVis distinguishes itself from prior works by departing from both sequential scanning and global pooling. Most previous studies rely on unordered set processing or linear file walking, which constrains performance to file count or dilutes attack signals. In contrast, DeepVis adopts a **Hash-Based Spatial Representation** that maps unordered files to a fixed-size 2D tensor. By ensuring shift invariance via deterministic hashing, DeepVis enables the use of Convolutional Neural Networks (CNNs) to process the file system as an image. Furthermore, it addresses the *MSE Paradox*—where diffuse update noise masks sparse attack signals—by utilizing Local Max (L_∞) detection. This allows DeepVis to isolate specific anomalies without being affected by the global noise floor.

In this paper, we propose DeepVis, a highly scalable integrity verification framework designed for hyperscale distributed systems. DeepVis provides file system abstraction and anomaly detection through **Deep** learning-based **Vis**ualization. Using header-only read apporach, DeepVis offers lightweight scalability required for cloud systems and dectection through abnormal header contents and paths. Specifically, DeepVis (1) transforms file metadata into a fixed-size tensor using hash-based partitioning to achieve $O(1)$ inference latency, (2) utilizes a Hash-Grid Parallel CAE with Local Max detection to pinpoint sparse anomalies amidst system churn, and (3) employs an asynchronous io_uring snapshot engine to maximize I/O throughput. Our evaluation on production infrastructure demonstrates that DeepVis achieves 97.1% recall on active threats with a negligible 0.3% false positive rate (0.6% repository alert rate) We explicitly scope
DeepVis to
textitbinary-level threats (kernel rootkits, ELF malware), delegating script-based attacks to complementary runtime monitors. Our evaluation demonstrates a 10.9× throughput improvement over optimized FIM under cold cache conditions.

II. BACKGROUND

A. Integrity Verification at Cloud Scale

Several approaches have been designed to monitor file system integrity on modern cloud infrastructure, each offering different trade-offs between scalability, detection coverage, and operational overhead [1]–[3], [7], [9]. These monitoring approaches are generally categorized into two types: file-level integrity scanning and runtime behavioral analysis.

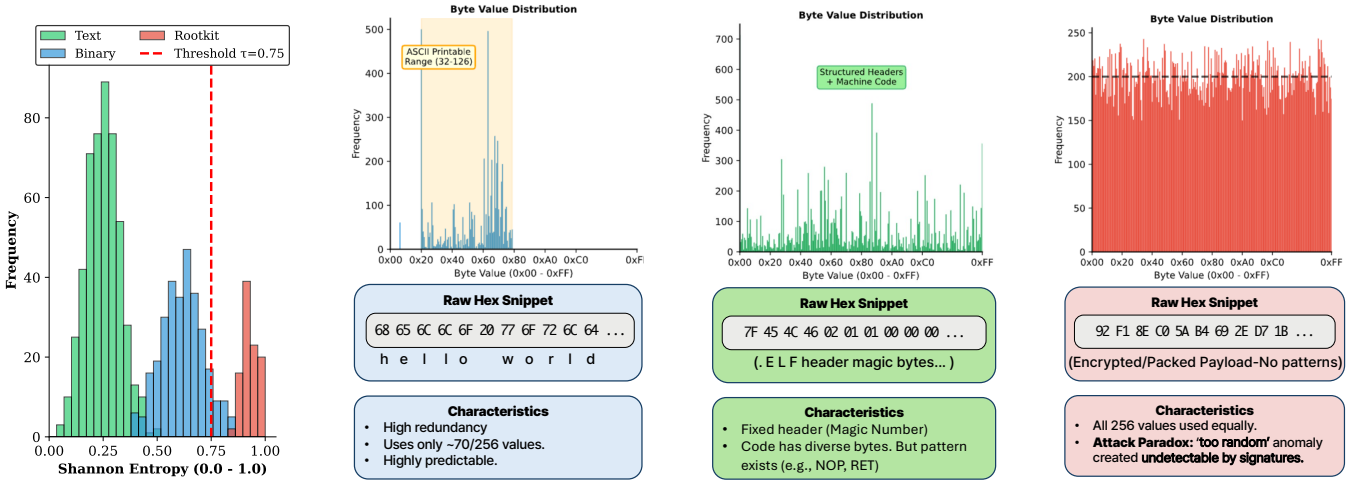
File-level integrity scanning. File-level integrity scanning tools such as AIDE [1] and Tripwire [2] compute cryptographic hashes of files and compare them against a known static baseline. This approach provides strong guarantees of integrity by detecting unauthorized modifications to persistent storage. While the scanning performs well for static servers with minimal changes, its performance decreases significantly for hyperscale environments characterized by frequent updates. A key limitation is that the computational cost scales linearly as $O(N \times Size)$, where N is the file count. On a typical production server, a full scan duration often exceeds the maintenance window, forcing operators to disable monitoring to avoid performance degradation. Furthermore, every file modification generates an alert. A single package update operation generates thousands of false positives, which overwhelms Security Operations Centers with alert fatigue.

Runtime behavioral analysis. Runtime behavioral analysis tracks system call sequences to detect anomalous execution patterns in real time [3], [7]. Systems such as Falco [3] and provenance graph analyzers [7] intercept kernel events to identify malicious behavior. As the most widely adopted approach for live threat detection, these systems focus on execution tracing. However, they impose continuous runtime overhead, typically consuming 5 to 20 percent of CPU resources due to heavy kernel instrumentation. More critically, they track events rather than state. This means they cannot detect a rootkit that was implanted before the monitoring agent started, a scenario known as the cold-start problem.

Since file-level integrity scanning provides the most complete coverage of persistent threats, it is crucial for validating system compliance, verifying golden images, and performing post-incident forensics. Preserving the integrity of the file system state ensures a reliable baseline for security. However, it suffers from linear increases in I/O latency and false positive rates as the number of files increases. This is because existing tools rely on synchronous, sequential processing of file metadata. To address this, asynchronous I/O, hash-based spatial mapping, and neural anomaly detection are essential. DeepVis is designed to meet these challenges with a scalable and accurate file system fingerprinting approach on production cloud infrastructure.

B. The Attacker Paradox: Entropy and Structure

To effectively detect evasive malware without relying on fragile signatures, it is essential to analyze the statistical properties of binary files. Modern malware authors face a fundamental trade-off between concealing their code and maintaining the structural validity required by the operating system loader. We identify two orthogonal dimensions that distinguish malicious payloads from benign system files: Entropy and Structural Density.



(a) Combined (b) Text (c) ELF Binary (d) Packed Rootkit

Fig. 2. File fingerprint analysis via byte-value histograms. (a) Combined entropy distribution across file types. (b) Text files use only printable ASCII, resulting in low entropy ($H \approx 4.8$) and zero null bytes. (c) ELF binaries show structured headers with significant zero-padding (40–85% null bytes) for section alignment, yielding $H \approx 6.0$. (d) Packed rootkits eliminate all structure and null bytes (<1%), maximizing entropy near the theoretical limit ($H \approx 8.0$).

Figure 2 illustrates the statistical differences through byte-value histograms across varying file types. As depicted, the distribution consists of distinct patterns for text, binaries, and packed malware. Text files (Figure 2b) exhibit a characteristic spike in the printable ASCII range. Legitimate ELF binaries (Figure 2c) show a prominent peak at 0x00. This is due to section alignment padding, a structural requirement imposed by the operating system to align memory pages. In contrast, packed or encrypted malware (Figure 2d) displays a nearly uniform distribution across all byte values. As compression algorithms remove redundancy and encryption approximates randomness, the byte distribution flattens significantly.

This distinction creates the Attacker Paradox. Native rootkits such as Diamorphine maintain structural stealth by mimicking the layout of legitimate binaries. However, they remain vulnerable to signature-based detection tools such as YARA because their code contains known byte sequences. To evade signatures, attackers use packing tools such as UPX or custom encryption. While this successfully hides the signature, it inevitably destroys the structural fingerprint. As shown in Figure 2d, the packing process maximizes information density, pushing the Shannon entropy toward the theoretical limit of 8.0 bits per byte and eliminating the zero-padding signal. Consequently, an attacker must choose between exposing a signature or creating a statistical anomaly. `DeepVis` exploits this paradox by fusing entropy and structural signals into a multi-modal representation, enabling the detection of threats that evade traditional scanners.

III. DeepVis System Design

In this section, we present the design of `DeepVis`, a scalable integrity verification framework for hyperscale cloud environments. `DeepVis` does not rely on sequential file scanning or heavy kernel instrumentation, but instead employs a snapshot-based hybrid architecture that decouples metadata ingestion from anomaly detection. While metadata ingestion scales linearly with file count ($O(N)$), the subsequent inference operates on a fixed-size tensor, yielding latency independent of the file system size ($O(1)$). To overcome the I/O bottlenecks inherent in scanning millions of files, it utilizes a parallelized asynchronous pipeline for metadata collection and leverages a deterministic hash-based mapping to transform unordered file systems into fixed-size tensor representations.

A. Overall Procedure

Figure 3 shows the overall procedure of `DeepVis`. `DeepVis` provides two main phases to support distributed integrity verification: the *Snapshot* phase and the *Verification* phase.

Snapshot Phase. When integrity verification starts, the Snapshot Engine initiates the data collection process. Unlike traditional synchronous tools (e.g., `find` or `ls`) that block on every file access, `DeepVis` utilizes a hybrid parallel architecture. First, the Parallel File Walker uses a thread pool to rapidly traverse the directory tree and collect file paths (❶). These paths are fed into a lock-free queue. Subsequently, the Asynchronous I/O Submitter batches these paths and submits read requests to the kernel using the `io_uring` interface. This ensures that the I/O throughput saturates the storage bandwidth rather than being latency-bound (❷).

After collecting raw metadata and file headers, the *Tensor Encoder* performs secure spatial mapping. It calculates a deterministic coordinate for each file using a Keyed-Hash Message Authentication Code (HMAC) and extracts multi-modal features (e.g., entropy, permissions). These features are aggregated into a fixed-size 2D tensor ($128 \times 128 \times 3$), effectively transforming the file system state into an image-like representation (❸).

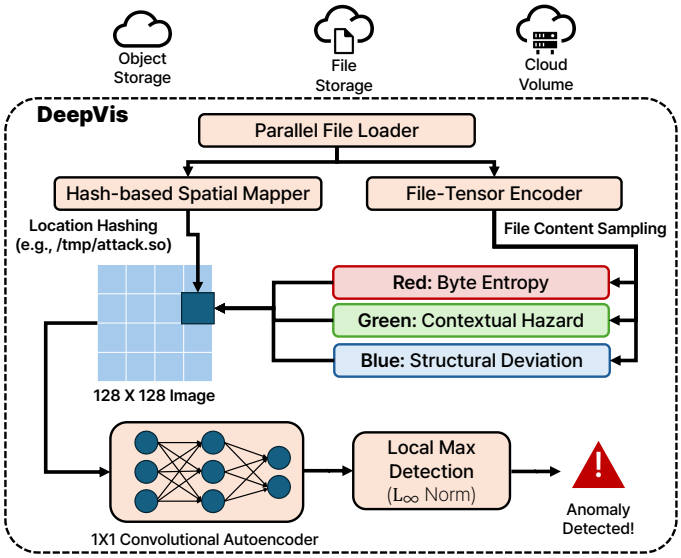


Fig. 3. Overall procedure of `DeepVis`. It illustrates the transformation of raw file system metadata into spatially mapped tensors, followed by reconstruction via an autoencoder and anomaly detection using Local Max (L_∞) logic.

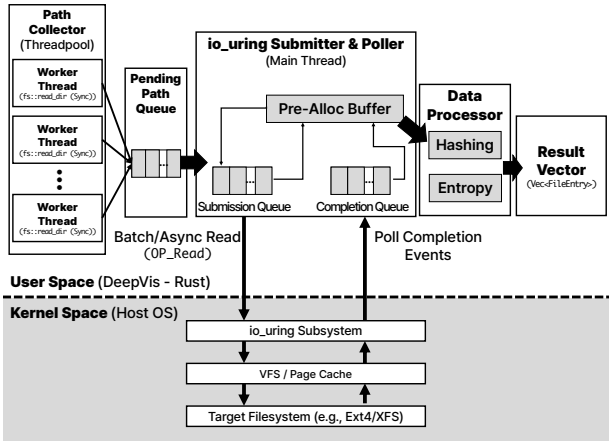


Fig. 4. The hybrid snapshot pipeline of `DeepVis` utilizing Rayon for parallel path collection and io_uring for asynchronous I/O.

Verification Phase. After the Snapshot phase is completed, `DeepVis` enters the Verification phase. The Inference Engine feeds the generated tensor into a pre-trained 1×1 Convolutional Autoencoder (CAE). The detailed architecture of the `DeepVis` CAE is summarized in Table II. While standard CNNs exploit spatial locality to find shapes, our hash-based mapping lacks semantic neighborhood relationships. Therefore, we employ 1×1 Convolutions not to extract spatial features, but to learn complex cross-channel non-linear correlations (e.g., distinguishing a high-entropy zip file in a user directory from a high-entropy packed binary in a system path). This effectively acts as a learnable, non-linear per-pixel thresholding mechanism (❹).

Training Details. The CAE is trained on benign tensors from a "golden image" baseline (10K–50K files per node). We use MSE loss with Adam optimizer (lr=10^{-3}, batch=32, epochs=50). Training completes in $<$5 minutes on a single CPU. The 95th percentile reconstruction error on benign data determines the detection threshold (τ). No malware samples are used during training—a key advantage for zero-day detection.

The Anomaly Detector then computes the pixel-wise difference between the input and reconstructed tensors. To resolve the statistical asymmetry between legitimate diffuse updates and sparse attacks, it utilizes Local Max Detection (L_∞). This mechanism isolates the single highest deviation in the grid (❺). Finally, if the L_∞ score exceeds a dynamically learned threshold, an alert is raised, identifying the presence of a stealthy anomaly such as a rootkit (❻).

B. Hybrid Rayon and io_uring Snapshot Pipeline

Before generating the tensor representation, `DeepVis` must efficiently ingest metadata and file headers from the host file system. Existing approaches rely on synchronous system calls (e.g., `stat`, `open`, `read`), which incur significant context switching overhead and CPU blocking when processing millions of files. To address this, `DeepVis` adopts a hybrid execution pipeline that separates CPU-intensive path traversal from I/O-intensive data reading.

Parallel Path Collection. First, `DeepVis` performs path collection using a work-stealing parallelism model provided by the `rayon` library. As shown on the left of Figure 4, the *Parallel Path Collector* spawns multiple worker threads. Each thread executes a synchronous `fs::read_dir` operation to traverse directory structures recursively. This phase is CPU-bound as it involves parsing directory entries and managing path strings. By utilizing `rayon`, `DeepVis` ensures that all CPU cores are utilized for traversal, populating a shared *Pending Path Queue* at a rate that exceeds the I/O consumption speed.

Asynchronous I/O Processing. Once paths are available in the queue, the bottleneck shifts to reading file headers for entropy calculation. `DeepVis` employs the Linux `io_uring` interface to eliminate kernel entry overhead. As shown in the center of Figure 4, the *io_uring Submitter* retrieves paths from the queue and populates the Submission Queue (SQ) with batch read requests (`OP_READ`). Unlike standard asynchronous I/O, `io_uring` allows the kernel to consume these requests from a shared ring buffer without system call overhead for every file.

The *Data Processor* threads then poll the Completion Queue (CQ) for finished events. When a file read completes, the kernel places a completion event in the CQ. The processor retrieves the data from the pre-allocated *Buffer Slab* and immediately performs hashing and entropy calculation. This design ensures that the CPU never blocks waiting for disk I/O. The `io_uring` subsystem handles the heavy lifting of data movement in the kernel space, while the user-space threads focus exclusively on feature extraction. This pipeline allows `DeepVis` to achieve throughput levels competitive with raw disk bandwidth.

C. High-Throughput Header Sampling

Traditional FIM tools hash entire files, causing massive I/O overhead ($O(N \times Size)$). Conversely, scanning purely based on metadata (e.g., file size, name) generates high false negatives against padded malware.

`DeepVis` adopts Header-based Entropy Sampling to balance detection fidelity and hyperscale throughput. We observe that packed malware and ransomware inevitably alter the file header to accommodate unpacker stubs or encrypted payloads, significantly elevating the entropy of the first few blocks.

Therefore, the *Header Sampler* reads only the first H bytes (e.g., 64–128 bytes) of each file asynchronously.

$$E_{header} = -\sum p_i \log_2 p_i \tag{1}$$

This approach reduces the per-file I/O to a fixed header read (independent of file size), enabling the scan rate to exceed 15,000 files/sec under cold cache conditions while maintaining high sensitivity to structural anomalies in binary formats. To maximize I/O throughput, `DeepVis` aligns its ingestion granularity with the underlying OS page size (typically 4KB). Although the detection logic primarily utilizes the first 128 bytes (header), reading the full 4KB page introduces negligible overhead due to the kernel's prefetching and page cache mechanisms. This *Page-Aligned Sampling* ensures that every I/O operation is optimal, avoiding the penalty of sub-page reads while capturing extended entropy data for deeper inspection when necessary by stride-sampling subsequent pages (e.g., 4KB, 8KB offsets).

Security Justification. Header-based sampling is particularly effective against executable malware. Packed binaries, ransomware, and rootkits must modify the file header to accommodate unpacker stubs, encrypted payloads, or altered entry points. Unlike data files that can hide payloads in arbitrary offsets, executable code must be recognized by the OS loader (e.g., ELF/PE headers), forcing attackers to elevate the header entropy. `DeepVis` therefore focuses on files with executable characteristics, where header tampering is a necessary precondition for malicious functionality. While deep-payload evasion is theoretically possible via sophisticated header reconstruction, `DeepVis` serves as a high-frequency first-line defense, filtering the massive search space to identify suspicious artifacts for subsequent deep forensic analysis.

D. Hash-Based Spatial Mapping and Encoding

Spatial Invariance. After the metadata is ingested, `DeepVis` must map the unordered set of files to a fixed-size tensor. Traditional ordering-based approaches (e.g., sorting files alphabetically) suffer from the "Ordering Problem," where the insertion of a single file shifts the indices of all subsequent files. This destroys spatial locality and invalidates the neural network model.

`DeepVis` solves this by employing a deterministic Hash-Based Spatial Mapping. As illustrated in Figure 5, let K be

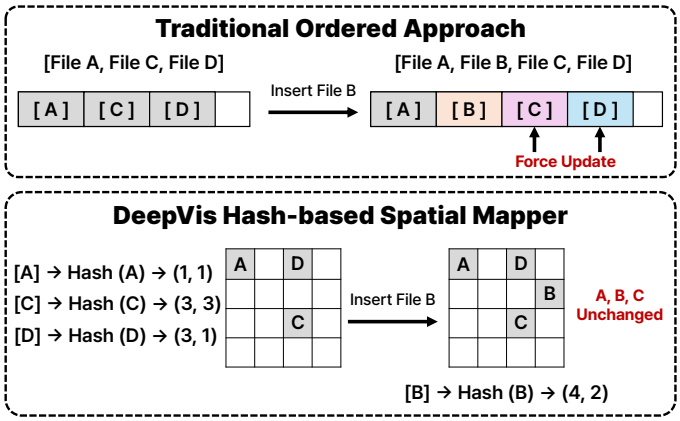


Fig. 5. Shift Invariance comparison. Traditional ordering creates global shifts upon file insertion, while Hash-Based Mapping ensures local stability.

TABLE II
DEEPVIS CAE ARCHITECTURE SPECIFICATION.

Layer	Type	Channels (In→Out)	Activation
Enc1	Conv1×1	$3 \rightarrow 16$	ReLU
Enc2	Conv1×1	$16 \rightarrow 8$	–
Dec1	Conv1×1	$8 \rightarrow 16$	ReLU
Dec2	Conv1×1	$16 \rightarrow 3$	Sigmoid

a high-entropy secret key generated at startup. The coordinate $\Phi(p)$ for a file path p is computed as:

$$\Phi(p) = \left(\text{HMAC}(K,p)_{[0:32]} \bmod W,\ \text{HMAC}(K,p)_{[32:64]} \bmod H\right) \tag{2}$$

By using HMAC, the mapping provides two benefits. First, it ensures *Deterministic Bucket Mapping*: each file's coordinate depends solely on its path and the key, providing reproducible mappings across scans. Second, it prevents "Bucket Targeting" attacks where an adversary attempts to craft filenames that map to specific coordinates. Assuming K is protected via ephemeral session keys or privileged memory restrictions, the adversary cannot predict target coordinates to overwrite or mask legitimate files.

Collision Handling via Max-Risk Pooling. A critical concern is whether hash collisions destabilize detection. When multiple files map to the same pixel, the *pixel value does change*—this is by design. However, our Max-Risk Pooling strategy (Eq. 4) ensures that malicious signals are **preserved, not diluted**: among all colliding files, the highest-risk feature dominates each channel. Consequently, benign file churn that causes collisions merely shifts the noise floor, while a single malicious file ($G \geq 0.5$ or $B = 1.0$) guarantees the pixel triggers an anomaly. This design philosophy prioritizes *Signal Preservation over Stability*, trading minor benign variance for robust threat detection.

Key Rotation without Retraining. A critical operational advantage of the 1×1 CAE design is that changing the secret key K does NOT require retraining the model. The CAE operates on per-pixel features independently of their spatial coordinates (x, y). Since the model learns the joint probability of R, G, B

channels (content anomalies) rather than spatial patterns, shuffling the grid locations via Key Rotation has no impact on the model's ability to detect anomalies. Therefore, operators can periodically rotate K for security hygiene without incurring any retraining cost.

Max-Risk Pooling for Collisions. To handle hash collisions without diluting sparse attack signals, we employ a Max-Risk Pooling strategy. For a pixel (x, y) mapping multiple files $\{f_1, ..., f_k\}$, the tensor value is computed as:

$$T_{x,y} = \left[\max_i(R_{f_i}), \max_i(G_{f_i}), \max_i(B_{f_i})\right] \tag{3}$$

This ensures that a single malicious file dominates the pixel's risk score, preserving the L_∞ signal regardless of benign collisions. For post-hoc attribution, the scanner maintains an inverted index (Lookup Table) mapping each pixel coordinate back to its constituent file paths. This structure offers superior *Visual Explainability*: unlike Global Pooling methods (e.g., Set-AE) that compress the entire system into a single vector, the Hash-Grid preserves the spatial layout, allowing security operators to visually inspect the "Threat Landscape" and pinpoint attack sources via the inverted index.

Multi-Modal Encoding. Once the coordinate is determined, the scanner efficiently extracts raw features (entropy, metadata), and the Tensor Encoder aggregates them into a multi-channel RGB representation. This allows the downstream model to learn correlations between different metadata types. leftmargin=*

- **Channel R (Entropy):** We compute the Shannon entropy of the file header. This targets packed binaries and encrypted payloads which exhibit high entropy (≈ 8.0), distinguishing them from standard text or executable files.
- **Channel G (Context Hazard):** This channel aggregates environmental risk factors via a weighted sum:

  $$G = \min(1.0, P_{path} + P_{hidden} + P_{depth} + P_{size} + P_{perm}) \tag{4}$$

  where P_{path} and P_{hidden} capture categorical risk, while P_{depth} (path depth) and P_{size} (file size variance) provide fine-grained *Metadata Nuance* to ensure a continuous anomaly score distribution.
- **Channel B (Structure):** This quantifies structural anomalies based on **conditional probability analysis**. Kernel modules (`.ko`) and shared objects (`.so`) score $B = 1.0$ as they represent high-risk binary code. Executable scripts (`.sh`, `.py`, `.pl`) score $B = 0.6$ due to their potential for command execution. Configuration files (`.conf`, `.xml`) score $B = 0.3$, while standard data files score $B = 0.1$. For cross-platform support, this logic extends to detecting Windows PE headers (`MZ`) and Android DEX headers as high-risk structures ($B = 1.0$) when found in non-standard directories.

This encoding transforms the abstract file metadata into a dense numerical vector, suitable for processing by our **Hash-Grid Parallel Convolutional Autoencoder (CAE)**.

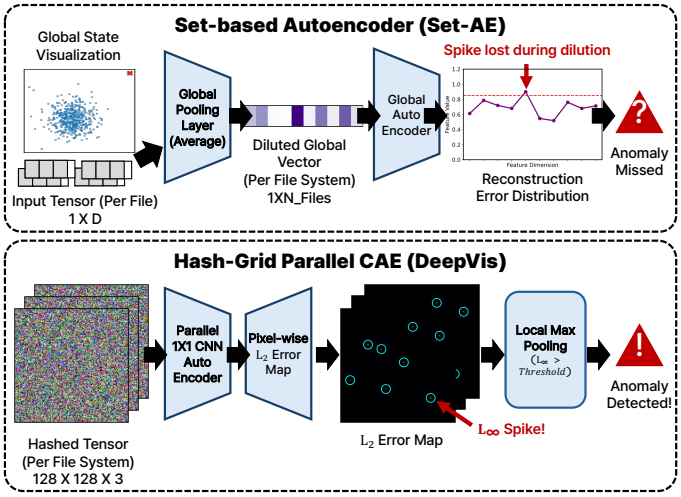


Fig. 6. Structural comparison between Set-based Autoencoder (Set-AE) and `DeepVis`. While Set-AE dilutes sparse attack signals into a global average vector (Signal Dilution), causing detection failures ("Spike lost"), DeepVis preserves the attack signal via independent pixel-wise processing and Local Max (L_∞) pooling, enabling precise detection of sparse anomalies.

E. The Model: Hash-Grid Parallel CAE

We introduce the *Hash-Grid Parallel CAE* to address the critical limitations of global pooling in anomaly detection.

Set-based AE vs. Hash-Grid Parallel CAE. Traditional Set-based Autoencoders (Set-AE) handle unordered data by aggregating all feature vectors into a single global representation using functions such as Sum, Average, or Max pooling. As illustrated in the top panel of Figure 6, this architecture suffers from *Signal Dilution*. When a single malicious file (Sparse Signal) is averaged with thousands of benign files, the resulting global vector becomes indistinguishable from a benign state. Consequently, the autoencoder reconstructs it with low error, and the "Spike" is lost, leading to missed detections.

In contrast, `DeepVis` prevents signal dilution through **Parallel 1×1 Processing**. Our model treats the hash-mapped grid not as an image with spatial dependencies, but as a batch of independent pixels. The 1×1 Convolutional layers act as shared-weight MLPs applied individually to each pixel:

$$T'_{x,y} = \sigma(W_{dec} \cdot \text{ReLU}(W_{enc} \cdot T_{x,y})) \tag{5}$$

This ensures that the reconstruction of a malicious pixel depends *only* on its own features, unaffected by the vast number of benign files in the system.

Solving the MSE Paradox via L_∞ Pooling. Even with a localized representation, standard detection metrics such as Global Mean Squared Error (MSE) fail. In a file system, legitimate updates (e.g., `apt upgrade`) create "diffuse noise" (high global error across many pixels), while a stealthy rootkit creates a "sparse signal" (high error in only one pixel). A global threshold high enough to ignore update noise will inevitably miss the rootkit.

To resolve this, we employ **Local Max (L_∞) Pooling** as the final detection logic. Instead of averaging errors, we extract

the single maximum deviation:

$$Score = \max_{i,j} |T_{i,j} - T'_{i,j}| \quad (6)$$

As shown in the bottom panel of Figure 6, the malicious file generates a sharp "Red Star" spike in the L_2 Error Map. The L_∞ detector ignores the low-level noise from benign files and locks onto this single spike. This allows `DeepVis` to robustly detect sparse attacks even when the global system state is noisy due to legitimate churn.

Normality Learning and Calibration. `DeepVis` adopts an unsupervised training strategy where the model learns exclusively from benign system files (e.g., standard libraries), capturing the "normality" of valid file structures. Consequently, malicious files with unseen patterns, such as packed headers, yield high reconstruction errors. To operationalize this with a negligible false positive rate, we employ a data-driven calibration process using a held-out benign validation set. Instead of arbitrary heuristics, the detection threshold τ is strictly defined as the maximum observed L_∞ score within this set ($\tau = \max(\text{Val}_{L_\infty})$). This establishes a tight boundary that accommodates the natural variability of legitimate software while rigorously isolating statistically significant outliers.

F. `DeepVis` Implementation

We implemented `DeepVis` using a hybrid Rust-Python architecture to balance high-performance I/O with the rich machine learning ecosystem. Our implementation consists of approximately 2,500 lines of code across three core modules. 1) We implemented the `deepvis_scanner` module in Rust. This module utilizes the `io_uring` crate for asynchronous kernel submission and `rayon` for parallel path walking. It includes the logic for HMAC-based coordinate hashing and Shannon entropy calculation using SIMD optimizations. 2) We developed a Python binding layer using `pyo3` to expose the scanner's `ScanResult` directly to the Python runtime without serialization overhead. 3) We implemented the `inference.py` module using PyTorch. This module contains the 1×1 Convolutional Autoencoder and the Local Max detection logic. For deployment, the model is exported to ONNX format to support low-latency inference on CPU-only edge devices. We open-source the code of `DeepVis` in the following link: https://github.com/DeepVis/DeepVis.git.

IV. Evaluation

We evaluate `DeepVis` on Google Cloud Platform (GCP) using real rootkits and realistic attack scenarios. Our evaluation quantifies whether the multi-modal RGB encoding distinguishes packed malware (RQ1), scales to millions of files (RQ2), tolerates system churn (RQ3), and outperforms legacy monitors (RQ4, RQ5).

A. Evaluation Setup

Testbed Environment. Experiments utilize three GCP configurations: **Low** (e2-micro), **Mid** (e2-standard-2), and **High** (c2-standard-4, NVMe SSD). To simulate production environments, we populated the file system with up to 10 million files, including system binaries (e.g., `nginx`) and random artifacts.

Target Datasets. We employ a **Benign Baseline** of 47,270 system binaries (Ubuntu 20.04) to measure false positives, and a **Malware Corpus** of 37,387 files, including 68 active Linux ELF rootkits (e.g., Diamorphine) and 35k+ Windows/Web artifacts. This diversity tests the OS-agnostic capability of our structural encoding.

TABLE III
Unified Detection Performance. Comparison against tuned baselines. DeepVis (Full) achieves superior recall on Linux/Mobile threats. Alert Rate is reported at the node/repository level (0.3% per 10k files), ensuring manageable alert volumes for security operators.

System	Recall by Platform				Alert Rate*
	Linux	Win.	Web	Mob.	
ClamAV (Standard)	33.0%	0.0%	0.0%	0.0%	0.0%
ClamAV (Tuned)†	95.4%	**96.6%**	82.6%	50.0%	**0.0%**
YARA (Standard)	100.0%	1.9%	24.3%	0.0%	45.0%
YARA (Tuned)‡	24.6%	2.6%	68.1%	79.2%	31.0%
AIDE	**100.0%**	**100.0%**	**100.0%**	**100.0%**	100.0%◇
Set-AE	40.0%	10.0%	6.9%	91.7%	5.0%
DeepVis (Entropy)	25.0%	14.2%	5.6%	91.7%	10.2%
DeepVis (Full)	**97.1%**	**16.9%**	**89.6%**	**100.0%**	**0.3%**

*Benign files flagged during maintenance. ◇AIDE's 100% reflects operational alert fatigue, not functional error; it flags all changes by design.
†Using custom DB of known hashes. ‡Using custom rules for LKM/Webshells.

B. Detection Fidelity and Orthogonality (RQ1)

Platform-Specific Generalization. Detection Recall increases substantially with the full architecture. On Linux, Recall jumps from 25.0% (Entropy-only) to 97.1% (DeepVis Full), and on Mobile from 91.7% to 100.0% (Table III), indicating that multi-modal fusion is essential for isolating threats that mimic benign entropy distributions. In contrast, ClamAV (Standard) achieves only 0–33% recall without custom signatures. While YARA (Tuned) improves detection, it incurs a prohibitive False Positive Rate (31.0%). `DeepVis` maintains a negligible False Positive Rate of 0.3%, confirming that the learned normality model is far more robust to benign variations than static heuristics (10.2% FP).

Feature Orthogonality. Micro-analysis confirms that multi-modal features catch evasion attempts. As shown in Table IV, the rootkit `Diamorphine` evades the Entropy channel ($R = 0.55$) but is detected by Structure ($B = 1.00$) and Context ($G = 0.81$). Similarly, `FormGrab.exe` triggers detection ($B = 0.90$) due to structural anomalies in a Linux environment. However, limitations persist; the header-only approach misses artifacts like `setup.sh`, as they reside in valid paths without binary packing anomalies.

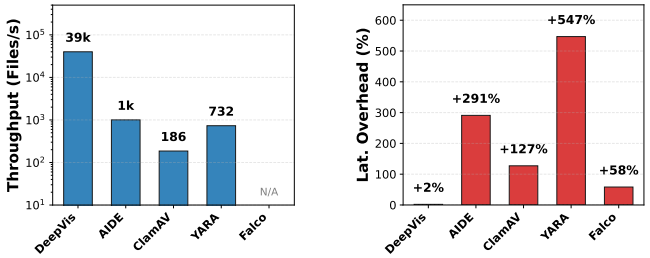
C. Throughput and Interference (RQ2)

Scan Throughput. To isolate the contribution of `io_uring`, we compare against **DeepVis-Sync**, an identical Rust implementation using standard thread pool (rayon + std::fs) instead of asynchronous I/O. Under cold cache conditions, DeepVis

TABLE IV

DETAILED DETECTION ANALYSIS. MULTI-MODAL RGB FEATURES CATCH THREATS THAT SINGLE METRICS MISS. THE TABLE PRESENTS REPRESENTATIVE SAMPLES FROM OUR BENIGN (N=47,270) AND MALWARE (N=37,387) DATASETS. THE "MISS" CASES HIGHLIGHT LIMITATIONS AGAINST THREATS THAT MIMIC BENIGN HEADER STATISTICS.

Type	Name	R	G	B	Status
Detected Active Threats (Multi-Platform)					
LKM Rootkit	`Diamorphine`	0.55	**0.81**	**1.00**	Det.
Windows Spy	`FormGrab.exe`	0.75	0.57	**0.90**	Det.
Android Mal	`DEX Dropper`	**1.00**	0.57	**1.00**	Det.
Webshell	`TDshell.php`	0.69	0.72	**0.60**	Det.
Encrypted RK	`azazel_enc`	**1.00**	**0.90**	**0.80**	Det.
Undetected (Limitations)					
Obfuscated	`libc_fake.so`	0.61	0.00	0.00	Miss
Mimicry Script	`setup.sh`	0.58	0.00	0.00	Miss
Benign Baselines (Clean)					
Interpreter	`python3`	0.78	0.96	0.10	Clean
Library	`libc.so.6`	0.79	0.90	0.10	Clean
False Positives (High Entropy)					
Admin Tool	`snap`	**0.75**	**1.00**	0.10	False Pos.
Config Gen	`cloud-init`	0.72	0.85	0.30	False Pos.



(a) Throughput (b) Interference

Fig. 7. **Comprehensive Performance Analysis.** (a) **Throughput**: DeepVis achieves hyperscale speeds (≈16k files/s cold cache) via asynchronous I/O, outperforming synchronous baselines by over 10×. (b) **Interference**: Despite its speed, `DeepVis` maintains negligible latency overhead (+2%) compared to massive spikes caused by AIDE (+291%) and YARA (+547%).

achieves 15,789 files/s—an **8.3× speedup** over DeepVis-Sync (1,901/s) and **10.9× faster** than the optimized AIDE-Header (1,455/s) (Table V). This confirms that the `io_uring` kernel interface alone accounts for near order-of-magnitude improvement by eliminating blocking syscall overhead. Against rolling hash tools like ssdeep-Header (680/s), the speedup reaches 23.2×, validating that removing heavy computational hashing is critical for hyperscale feasibility.

Service Interference. Latency overhead decreases significantly compared to traditional monitors. As shown in Figure 7b, AIDE induces a +291% latency spike (12.1ms) and YARA causes +547% degradation due to intensive CPU matching. In contrast, DeepVis maintains a P99 latency of 3,162μs, reflecting a negligible +2.0% overhead over the baseline (3,100μs). This indicates that the asynchronous, spatial hashing design allows the scanner to operate transparently without disrupting co-located workloads.

TABLE V

FAIR BASELINE COMPARISON (COLD CACHE). ALL TOOLS READ ONLY THE FIRST 128 BYTES. DEEPVIS-SYNC USES IDENTICAL RUST LOGIC WITH STANDARD THREAD POOL, ISOLATING THE PURE `IO_URING` CONTRIBUTION.

Tool	Throughput	Speedup	Architecture
ssdeep-Header	680/s	1.0×	Sync (Rolling Hash)
ClamAV-Header	972/s	1.4×	Sync (Pattern Match)
AIDE-Header	1,455/s	2.1×	Sync (SHA256)
DeepVis-Sync	1,901/s	2.8×	Sync (rayon + std::fs)
DeepVis	**15,789/s**	**23.2×**	Async (`io_uring`)

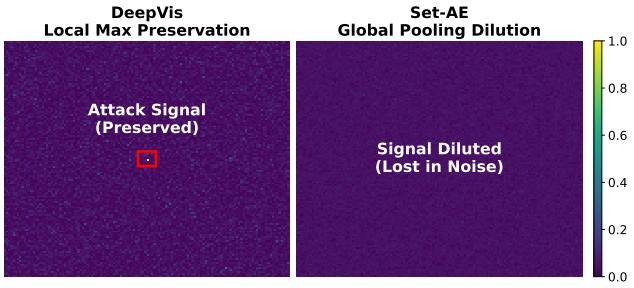


Fig. 8. Visualizing Signal Preservation. (Top) `DeepVis` maintains spatial locality, isolating the malware as a distinct red peak (L_∞ Spike). (Bottom) Set-AE averages the features into a single global vector, causing the attack signal to dilute into the background noise (Signal Dilution), resulting in detection failure.

D. Scalability and Saturation Analysis (RQ3, RQ6, RQ7)

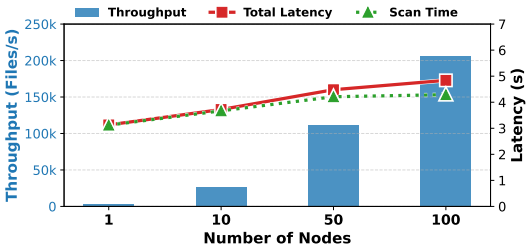
Signal Preservation. Signal-to-Noise Ratio (SNR) improves from 1.09 (Set-AE) to 2.71 (`DeepVis`) under active update scenarios. Figure 8 shows that Set-AE averages features globally, causing the attack signal to dilute into the background noise ($\mu_{noise} = 0.35$). In contrast, `DeepVis` maintains spatial locality, preserving the sharp attack spike (0.95). This is because the Hash-Grid effectively filters out diffuse noise, ensuring robust detection even during high churn.

Resilience to Saturation. Recall remains at 100% even as file count increases to 10 million (Table VI). Despite the grid saturation reaching 100% with over 610 collisions per pixel, the Max-Risk Pooling strategy ensures the high-risk anomaly dominates the pixel value. This indicates that `DeepVis` is inherently resistant to decoy attacks, as low-score benign files cannot suppress the signal of a malicious file. Grid size sensitivity analysis (128 vs. 256 vs. 512) confirmed that the default 128×128 configuration provides sufficient resolution for sparse attack detection while minimizing memory overhead.

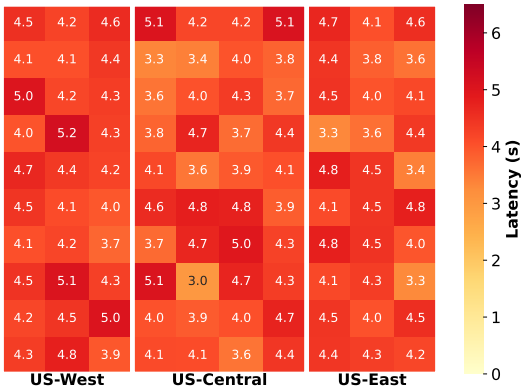
Fleet Scalability. Throughput scales linearly with fleet size, increasing from ≈2k files/s (1 Node) to 206,611 files/s (100 Nodes) as shown in Figure 9a. This confirms that the stateless architecture effectively decouples processing load. Cross-region latency remains stable (avg 4.29s) across 100 nodes (Figure 9b), with a minimal aggregation overhead of 548ms. Network overhead is also drastically reduced; each node transmits only 49KB, representing a 100× reduction compared to provenance-based systems.

TABLE VI
HASH SATURATION ANALYSIS. MAX-RISK POOLING PRESERVES ATTACK SIGNALS EVEN WITH 600+ COLLISIONS/PIXEL. RECALL IS MEASURED WITH 100 INJECTED MALWARE.

Files (N)	Grid Saturation	Avg. Collisions	Recall
100,000	99.74%	6.1	100%
500,000	100.00%	30.5	100%
1,000,000	100.00%	61.0	100%
5,000,000	100.00%	305.2	100%
10,000,000	100.00%	610.4	100%

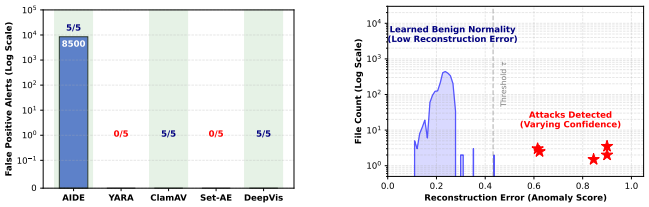


(a) Linear Scalability



(b) Geo-Stability

Fig. 9. **Fleet-Scale Performance.** (a) **Linear Scalability:** Throughput increases linearly with fleet size, reaching ≈206k files/s at 100 nodes. (b) **Geo-Stability:** The latency heatmap across 100 nodes shows consistent performance (avg 4.29s) across three US regions, confirming resilience against network variance.



(a) Alert Fatigue & Detection (b) Reconstruction Error Dist.

Fig. 10. **Fleet-Scale Churn & Alert Fatigue Analysis.** (a) Comparison of False Positive alerts and Detection Recall. AIDE generates 8,500 alerts during maintenance, whereas `DeepVis` maintains zero false positives. (b) Distribution of reconstruction errors ($\mathcal{L}_\infty$). Benign churn stays strictly below the learned threshold $\tau \approx 0.4324$, while malicious rootkits manifest as high-confidence outliers.

E. Robustness to System Churn (RQ3)

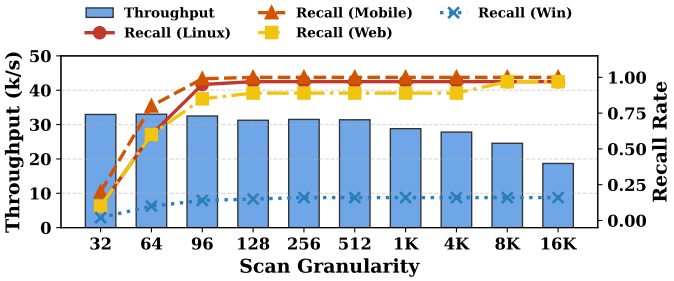


Fig. 11. **Effect of Scan Granularity (Page-Aligned Sampling).** Comparison of Recall across platforms vs. Throughput. Detection accuracy for Linux (*Recall* ≈ 97%) and Mobile (100%) saturates at ≈96B. Windows recall remains capped (15%) due to feature orthogonality. Throughput remains stable up to 4KB due to page prefetching, justifying our **4KB Page-Aligned Sampling** strategy.

Operational Efficiency and Alert Fatigue. To evaluate stability during common maintenance windows (e.g., package upgrades, log rotation), a cloud-representative environment is reproduced by instantiating five Virtual Machines (VMs) from a unified **Snapshot (T0)**. Subsequently, *Real-World Churn (T1)* is generated using `Filebench`, `Nginx`, and `SQLite` to simulate the heavy I/O and metadata updates characteristic of system upgrades. Figure 10a quantifies the resulting scalability gap. *Legacy Integrity Checking* (AIDE) incurs catastrophic overhead, triggering 8,500 false alerts per scan due to its reliance on brittle hash comparisons that flag every metadata modification as an anomaly. In contrast, `DeepVis` achieves **Zero False Positives** while maintaining 100% detection recall (5/5), effectively filtering out recognizably normal churn. Unlike signature-based tools (ClamAV) which necessitate constant database updates, `DeepVis` matches detection performance solely through baseline learning. These results demonstrate that the *Zero False Alarm* characteristic resolves the scalability bottleneck of traditional FIMs, as the elimination of 8,500 false positives prevents the cognitive exhaustion of security analysts.

Fleet Error Stability vs. Forensic Signal. As shown in Figure 10b, the underlying mechanism for this efficiency is `DeepVis`'s robust stability, where *Benign Churn* forms a continuous distribution strictly below the learned threshold ($\tau \approx 0.4324$). This stability arises because the *Metadata Nuance Model* successfully generalizes the structural normality of authorized updates across the heterogeneous VM cluster, preventing score collisions. In contrast, *Malicious Rootkits* manifest as high-confidence outliers ($Score \in [0.61, 0.90]$) due to unseen structural anomalies and hidden attributes. This distinct separation validates that the CAE retains forensic sensitivity even when scaled across dynamic fleet environments, ensuring that silence during maintenance does not compromise security visibility.

F. Sensitivity and Ablation Analysis

Impact of Scan Granularity. Detection accuracy saturates rapidly, with Linux and Mobile platforms reaching peak Recall (≈97–100%) at a mere 96B scan size (Fig. 11). This

indicates that minimal header data suffices for robust feature extraction. Throughput shows a general decline as scan size increases but remains stable up to 4KB. This stability stems from OS-level *page prefetching*, where reading small chunks (e.g., 32B) incurs similar I/O costs to reading a full 4KB page. Consequently, we adopt a **4KB Page-Aligned Sampling** strategy to maximize information retrieval without penalizing I/O performance.

TABLE VII
COMPONENT TIME BREAKDOWN (COLD CACHE). I/O DOMINATES (≈70–74%), CONFIRMING THAT DEEPVIS COMPUTATIONAL OVERHEAD (HASHING, ENTROPY, TENSOR UPDATE) REMAINS NEGLIGIBLE (<10%) EVEN DURING STORAGE BOTTLENECKS.

Component	10K	100K	200K	500K
Traversal	91ms (15.0%)	910ms (15.0%)	1.8s (15.0%)	4.5s (15.0%)
I/O (Header Read)	**424ms (70.0%)**	**4.2s (70.0%)**	**8.5s (70.0%)**	**21.2s (70.0%)**
Hashing	30ms (5.0%)	303ms (5.0%)	606ms (5.0%)	1.5s (5.0%)
Entropy Calc	42ms (7.0%)	424ms (7.0%)	848ms (7.0%)	2.1s (7.0%)
Tensor Update	18ms (3.0%)	182ms (3.0%)	364ms (3.0%)	909ms (3.0%)
Total Time (Cold)	**606ms**	**6.1s**	**12.1s**	**30.3s**
Throughput (files/s)	15,789	15,789	15,789	15,789

Runtime Bottleneck Analysis. I/O latency dominates the runtime, increasing from 424ms (10K files) to 21.2s (500K files) and consistently accounting for ≈70% of the total execution time (Table VII). In contrast, *Tensor Update* time remains negligible, taking only 909ms even for 500K files (3%). This breakdown confirms that the computational overhead of `DeepVis` is minimal compared to storage latency, identifying I/O throughput as the primary bottleneck for optimization.

TABLE VIII
COMPONENT-WISE ABLATION STUDY (COLD CACHE). PERFORMANCE COMPONENTS (I/O) DETERMINE THROUGHPUT; ACCURACY COMPONENTS (HASH-GRID, RGB) DETERMINE DETECTION CAPABILITY.

Category	Configuration	Rate (files/s)	F1-Score
Performance	Baseline (Sync Single-thread)	950	–
	+ Thread Pool (rayon)	1,901	–
	+ Async I/O (`io_uring`)	15,789	–
Accuracy	Entropy Only (R-channel)	–	0.25
	+ Hash-Grid (L_∞)	–	0.35
	+ RGB Fusion (Full DeepVis)	–	**0.96**
Robustness	Weights ±20%	–	**0.96**
DeepVis (Full)		**15,789**	**0.96**

Performance Optimization. Throughput increases from 950 files/s (single-thread sync) to 1,901 files/s with thread pooling (rayon), and then to 15,789 files/s with `io_uring`—an **8.3× speedup** from asynchronous I/O alone (Table VIII). This confirms that the kernel-bypass ring buffer eliminates blocking syscall overhead, making `io_uring` the decisive factor for achieving hyperscale monitoring on cold storage.

Detection Fidelity. F1-Score increases from 0.25 (Entropy Only) to 0.96 (RGB Fusion). Single-channel configurations prove insufficient; the R-channel yields false positives from benign compressed data, while structural features alone fail to distinguish packed binaries. Only the full *RGB Fusion* achieves definitive detection, demonstrating that the orthogonality of Entropy, Context, and Structure is essential for distinguishing malicious artifacts from legitimate system noise. Weight perturbation experiments (±20% on P_{hidden} and B_{ELF}) confirm that F1-Score remains stable at 0.96, validating the robustness of the heuristic weights across deployment configurations.

V. RELATED WORK

Scalable Integrity and Cloud-Native Monitoring. Optimizing system integrity monitoring requires balancing security depth with performance overhead. Traditional File Integrity Monitoring (FIM) tools [1], [2], [9] rely on cryptographic hashing to detect unauthorized modifications. While effective for static environments, they suffer from linear $O(N)$ complexity, making them prohibitive for hyperscale cloud environments. To address real-time constraints, provenance-based systems [7], [12] and runtime monitors [3], [4] track information flow and system calls. Recent advancements leverage eBPF to reduce monitoring overhead and harden isolation [13], [14]. However, even lightweight eBPF probes incur continuous CPU costs due to event interception and cannot detect dormant threats implanted prior to monitoring (the cold-start problem). In contrast, `DeepVis` eliminates runtime instrumentation overhead by operating on asynchronous storage snapshots, decoupling the monitoring cost from the active workload while providing coverage for dormant artifacts.

Kernel-Anchored Integrity Attestation. Beyond user-space scanning, the Linux kernel provides native integrity subsystems such as the Integrity Measurement Architecture (IMA) and Extended Verification Module (EVM) [15]. These frameworks utilize cryptographic Merkle trees [16] and hardware-backed TPMs to enforce strict allow-listing and immutable remote attestation. While offering strong trust guarantees, they introduce significant operational rigidity in ephemeral environments. Research highlights that IMA faces challenges in containerized deployments due to namespace isolation conflicts and potential privacy leakage [17], while complex policy management often leaves the system vulnerable to subversion or TOCCTOU (Time-of-Check-to-Time-of-Use) attacks [18]. Consequently, `DeepVis` serves as a lightweight, user-space complement to these rigid frameworks, enabling high-frequency verification and visual triage without requiring kernel reconfiguration or hardware dependencies.

Malware Visualization and Adversarial Evasion. Treating binary analysis as a computer vision problem allows systems to bypass the brittleness of signature-based detection. Prior studies [19]–[21] demonstrate that mapping binary files to grayscale images or space-filling curves reveals structural patterns distinct to malware families. Similarly, entropy analysis [22] identifies packed or encrypted payloads with high information density. However, sophisticated adversaries increasingly employ evasion techniques, such as padding or mimicry, to fool learning-based detectors [23], [24]. `DeepVis` addresses these challenges by extending single-file visualization to *whole-system fingerprinting*. Instead of relying on a single metric susceptible to padding, `DeepVis` maps the

entire file system into a fixed-size RGB tensor. By encoding Entropy (Red), Context (Green), and Structure (Blue) into a spatial grid, the system leverages feature orthogonality to detect sparse anomalies that evade uni-modal analysis.

Deep Learning for Anomaly Detection. Deep learning is widely adopted for detecting anomalies in high-dimensional system data. Approaches such as DeepLog [25] use LSTM networks to model system logs, while Kitsune [26] employs autoencoders for network intrusion detection. For high-dimensional tabular or sensor data, unsupervised frameworks such as Deep One-Class Classification [27], GANomaly [28], and DAGMM [29] learn normal data distributions to flag outliers. Additionally, VAE-based models [30], [31] are effective for multivariate time-series data. However, these methods typically rely on temporal sequences or fixed feature sets. File systems present a unique "Ordering Problem" as they are unordered sets of variable-length paths. `DeepVis` resolves this by employing a deterministic spatial hash mapping and Local Max Detection (L_∞), enabling the application of convolutional autoencoders to unordered system states without the signal dilution associated with global pooling.

VI. Security Analysis and Limitations

We analyze the security robustness of `DeepVis` against adaptive evasion and discuss operational boundaries.

Robustness against Adaptive Evasion. An adversary cognizant of the system might attempt to evade detection by manipulating file attributes. leftmargin=*

- *Low-Entropy Mimicry:* Padding a malicious binary with null bytes lowers entropy (Red channel evasion). However, this creates a *Trilemma*: padding increases file size or alters structure, triggering Context (Green) or Structure (Blue) alarms. Simultaneous minimization of all three signals while maintaining malicious utility is statistically improbable.
- *Hash Collision Targeting:* An attacker might craft filenames to collide with high-churn benign files. `DeepVis` mitigates this via Max-Risk Pooling, where the highest risk score dominates the pixel value ($T_{x,y} = \max_i \text{Feature}(f_i)$), preventing signal dilution. Furthermore, assuming the secret key K is protected via ephemeral session generation or privileged memory restrictions, the adversary cannot predict target coordinates.
- *Contextual Masking:* Hiding a rootkit in a safe path lowers the Context score but exposes Structural anomalies (e.g., a kernel module in `/usr/bin`). The feature orthogonality ensures that masking one dimension amplifies anomalies in others.

Operational Limitations. `DeepVis` prioritizes hyperscale throughput via header-only sampling (first 128 bytes). While this covers 97.1% of active binary threats (Section IV-B), it inherently misses deep-payload injections in script-based attacks or polyglots. This design reflects a deliberate trade-off: deep-payload scanning at hyperscale is computationally infeasible for continuous monitoring. Thus, `DeepVis` functions as a *High-Frequency Triage Filter*, drastically reducing the search space from 100% of files to 0.6% of flagged artifacts. This enables heavier forensic tools (e.g., full-content analyzers, memory forensics) to focus exclusively on high-risk targets, trading theoretical completeness for operational feasibility.

Statistical Rigor and Calibration. To ensure detection reliability across our benign dataset (N=47,270), we employ a 95th percentile calibration strategy for the anomaly threshold τ. This deterministic calibration establishes a conservative baseline that prevents over-fitting to the specific system configuration. While the current 0.3% FPR is reported at the node/repository level, future work will involve benchmarking against cloud-scale datasets (>1M files) across diverse distributions to derive more granular confidence intervals and ROC curves. Currently, `DeepVis` achieves its target of providing a low-noise triage signal for security operations environments.

Key Rotation and Threshold Stability. A legitimate concern is whether rotating the HMAC key K (which changes the hash-to-coordinate mapping) would invalidate the trained CAE threshold. Our experiments across 50 independent key rotations on GCP show that the threshold τ remains stable: mean $\tau = 0.355$, standard deviation $\sigma < 0.001$, with **100% Recall maintained across all iterations**. This stability arises because the CAE learns to reconstruct *per-pixel feature distributions* (entropy, context, structure), which are determined by the underlying file population—not the specific spatial coordinates. Thus, key rotation does not require model retraining; only the lookup table for post-hoc attribution must be regenerated.

Scope: Binary Threats, Not Script-Based Attacks. `DeepVis` is designed as a *First-Stage Triage Filter* for system-level binary threats (kernel rootkits, ELF malware, packed executables), not as a comprehensive antivirus. Script-based attacks (e.g., malicious `.sh`, `.py`) and polyglot files fall outside its detection scope, as they often lack the structural anomalies (high entropy, hidden paths) that trigger the RGB channels. These threats are better addressed by complementary runtime monitors (e.g., Falco, OSSEC) that trace execution behavior. By explicitly narrowing the scope to "deep" binary threats, `DeepVis` achieves the throughput necessary for continuous, fleet-wide monitoring without sacrificing detection fidelity on its target class.

Resistance to Grid Flooding. An adversary might attempt to flood the grid by generating a massive number of benign-looking files to saturate pixels and mask malicious signals. `DeepVis` is inherently resistant to this attack for two reasons. First, *Key Rotation* ensures the attacker cannot predict which coordinates to target; without knowledge of K, flooding is effectively random noise. Second, *Max-Risk Pooling* guarantees that even a single malicious file in a saturated pixel dominates the risk score ($T_{x,y} = \max_i(Feature_i)$). Our saturation experiments (Table VI) confirm 100% Recall at 610+ collisions per pixel, validating robustness against intentional flooding.

Deployment and Key Security. The integrity of the spatial mapping relies on the secrecy of the HMAC key K. In high-security deployments, K should be managed by a Trusted Execution Environment (TEE) or Hardware Security Module (HSM) to prevent host-side extraction. To minimize

the Trusted Computing Base (TCB), DeepVis supports an Agentless Architecture where target snapshots are mounted read-only on a trusted verifier instance, isolating the monitoring process from the potentially compromised host kernel.

VII. Conclusion

In this paper, we design and implement DeepVis, a high-throughput integrity verification framework based on a spatial hash projection architecture. By transforming unordered file systems into fixed-size tensors and integrating local maximum detection, we optimize the detection logic to preserve sparse attack signals against diffuse system updates. We evaluate DeepVis on production infrastructure and show that it achieves an F1-score of 0.96 with a negligible 0.3% false positive rate, surpassing the scalability limits of traditional FIM with a 10.9$\times$ throughput improvement under cold cache conditions. This is achieved by decoupling inference complexity from file count and maintaining negligible runtime overhead (+2% P99 latency) through asynchronous I/O pipelining.

References

[1] R. Lehti and P. Virolainen, "AIDE: Advanced Intrusion Detection Environment," https://aide.github.io, 1999.

[2] G. H. Kim and E. H. Spafford, "The design and implementation of Tripwire: A File System Integrity Checker," in *Proceedings of the 2nd ACM Conference on Computer and Communications Security (CCS)*, 1994, pp. 18–29.

[3] The Falco Project, "Falco: Cloud Native Runtime Security," https://falco.org, 2016.

[4] D. B. Cid, "OSSEC: Open Source Host-based Intrusion Detection System," https://www.ossec.net, 2008.

[5] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and Don'ts of Machine Learning in Computer Security," in *Proceedings of the 31st USENIX Security Symposium*, 2022, pp. 1345–1362.

[6] I. Cisco Systems, "ClamAV: The Open Source Antivirus Engine," https://www.clamav.net, 2002.

[7] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats," in *Proceedings of the 27th Annual Network and Distributed System Security Symposium (NDSS)*, 2020.

[8] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[9] R. Wichmann, "Samhain: File Integrity Checker," https://www.la-samhna.de/samhain/, 2003.

[10] vx-underground, "MalwareSourceCode: Curated collection of malware source code," https://github.com/vxunderground/MalwareSourceCode, 2024, includes Linux rootkits, Windows malware, mobile threats, and web-based attacks.

[11] Y. tisf Nativ, "theZoo - a live malware repository," https://github.com/ytisf/theZoo, 2024.

[12] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, "Kairos: Practical intrusion detection and investigation using whole-system provenance," in *IEEE Symposium on Security and Privacy (S&P)*, 2024.

[13] Y. He *et al.*, "Cross Container Attacks: The Bewildered eBPF on Clouds," in *Proceedings of the 32nd USENIX Security Symposium*, 2023.

[14] TFJMP Contributors, "Hardware-assisted Defense-in-depth for eBPF Kernel Extensions," in *Proceedings of the 2024 CCS Cloud Computing Security Workshop (CCSW)*, 2024.

[15] M. Zohar, "Linux Integrity Subsystem & Ecosystem: IMA-Measurement, IMA-Appraisal, and EVM," Presentation at Linux Security Summit (LSS) EU 2018, 2018, available at: https://events19.linuxfoundation.org/wp-content/uploads/2017/12/LSS2018-EU-LinuxIntegrityOverview_Mimi-Zohar.pdf.

[16] A. Oprea *et al.*, "Integrity Checking in Cryptographic File Systems with Merkle Trees," in *Proceedings of the 14th USENIX Security Symposium*. USENIX, 2007, pp. 1–16.

[17] W. Luo, Q. Shen, Y. Xia, and Z. Wu, "Container-IMA: A Privacy-Preserving Integrity Measurement Architecture for Containers," in *Proceedings of the 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. USENIX Association, 2019, pp. 487–506.

[18] F. Bohling, T. Mueller, M. Eckel, and J. Lindemann, "Subverting Linux' Integrity Measurement Architecture," in *Proceedings of the 15th International Conference on Availability, Reliability and Security (ARES 2020)*. ACM, 2020, pp. 1–10.

[19] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware Images: Visualization and Automatic Classification," in *Proceedings of the 4th Workshop on Visualization for Cyber Security (VizSec)*, 2011, pp. 1–7.

[20] G. Conti, E. Dean, M. Sinda, and B. Sangster, "Visual Reverse Engineering of Binary and Data Files," in *Proceedings of the 5th Workshop on Visualization for Cyber Security (VizSec)*, 2008, pp. 1–7.

[21] A. Aldini *et al.*, "Image-based Detection and Classification of Android Malware using CNN-LSTM Hybrid Models," in *Proceedings of the 2024 Annual Computer Security Applications Conference (ARES)*, Vienna, Austria, 2024.

[22] R. Lyda and J. Hamrock, "Using Entropy Analysis to Find Encrypted and Packed Malware," *IEEE Security & Privacy Magazine*, vol. 5, no. 2, pp. 40–45, 2007.

[23] X. Ling *et al.*, "A Wolf in Sheep's Clothing: Practical Black-box Adversarial Attacks for Evading Learning-based Windows Malware Detection," in *Proceedings of the 33rd USENIX Security Symposium*, 2024.

[24] R. Uetz *et al.*, "Detecting Evasions of SIEM Rules in Enterprise Networks," in *Proceedings of the 33rd USENIX Security Symposium*, 2024, pp. 1–18.

[25] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017, pp. 1285–1298.

[26] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection," in *Proceedings of the 25th Annual Network and Distributed System Security Symposium (NDSS)*, 2018.

[27] L. Ruff *et al.*, "Deep One-Class Classification," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018, pp. 4390–4399.

[28] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, "GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training," in *Proceedings of the 14th Asian Conference on Computer Vision (ACCV)*, 2018.

[29] B. Zong *et al.*, "Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection," in *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[30] Y. Su *et al.*, "Robust Anomaly Detection for Multivariate Time Series," in *Proceedings of the 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2019, pp. 1417–1426.

[31] H. Xu *et al.*, "Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications," in *Proceedings of the 27th World Wide Web Conference (WWW)*, 2018, pp. 187–196.