

DeepVis: Scalable Deep Learning-Based File System-to-Image Integrity Audit for Distributed Systems

Abstract—This paper presents **DeepVis**, a scalable file system integrity verification system for hyperscale storage environments such as cloud data centers. Designed as a rapid first-pass filter, our key idea is to transform the file system state into a fixed-size RGB image and map individual files to specific pixels to enable scalable inspection regardless of the file count. Specifically, **DeepVis** introduces: (1) an asynchronous lightweight snapshot engine utilizing Rust and `io_uring`; (2) a parallel mapping pipeline that converts file metadata into a fixed-size image representation; and (3) a spatial anomaly detection mechanism for identifying anomalous patterns within the file system represented as a single image. We evaluate **DeepVis** on production-grade cloud infrastructure. Our results show that **DeepVis** acts as an effective snapshot-based audit system. It detects 97.1% of hidden kernel modules and packed binaries in Linux environments. In addition, **DeepVis** achieves a 121.45 \times speedup over traditional hash-based full-scan integrity tools while incurring minimal overhead of less than 2 percent CPU usage.

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

I. INTRODUCTION

Cloud computing abstracts physical infrastructure into dynamic, ephemeral resources, creating a computational model distinct from traditional on-premise environments. Ensuring workload integrity across cloud instances and large-scale HPC clusters is critical, as operators must prevent unauthorized modifications across thousands of nodes. However, modern applications introduce a tension between security and agility, as frequent deployments and updates cause massive file churn that renders traditional models ineffective.

To address this, two main strategies exist: File Integrity Monitoring (FIM) and Runtime Behavioral Analysis. FIM tools such as AIDE [1] and Tripwire [2] detect static changes through cryptographic hashes, while runtime monitors such as Falco [3] and OSSEC [4] trace system calls for anomalies. 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, as legitimate updates generate thousands of false positives, masking true threats [5]. Thus, the operational cost

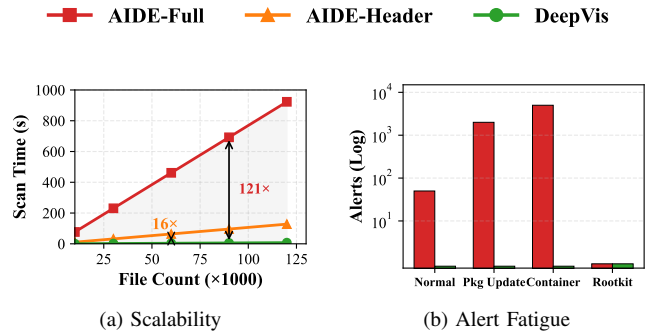


Fig. 1. (a) **DeepVis** achieves high-throughput saturation via async I/O. (b) Legitimate updates generate false alerts in AIDE.

exceeds the theoretical benefit, forcing operators to disable monitoring during maintenance windows which can create a loophole that attackers can exploit.

Figure 1 compares the scalability and precision of an existing FIM tool (i.e., AIDE) with the proposed scheme (i.e., **DeepVis**). We used a cloud instance from a real cloud system as described in Section IV-A and performed a user directory scan. The AIDE scan time increases linearly with a steep slope, exceeding four minutes for only 100K files. This behavior results from synchronous I/O operations and full file reads required to compute hash values and detect all changes since the latest snapshot. In contrast, **DeepVis** reads only the first 4 KB of each file and uses a parallelized asynchronous pipeline to saturate storage bandwidth, keeping scan times under seven seconds for the same dataset. Although file ingestion remains physically $O(N)$, **DeepVis** achieves an effective speedup of up to 121.45 \times over traditional full-hash baselines and 16.38 \times over modified full-hash schemes that read the same amount of data due to its efficient I/O design. Crucially, **DeepVis** ensures that subsequent anomaly detection (inference) time remains constant regardless of dataset size, and alerts only in response to rootkit attacks with 97.1% precision, whereas AIDE reports every file change, leading to alert fatigue for operators at the expense of full hash verification.

Many previous studies, as summarized in Table I, have explored approaches to enhance system monitoring scalability. Traditional FIM tools [1], [2] prioritize cryptographic exactness via full hashing of entire files but suffer $O(N)$ scalability limits unsuitable for hyperscale systems. Runtime approaches [3], [7] use eBPF tracing or provenance graphs

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	Async	Obfusc.	0-Day	Low Ovhd.
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	✓	✓	✓	✓

for zero-day threats but require continuous monitoring that degrades performance. Deep learning methods such as Set-AE [8] provide lightweight set-based anomaly detection models but they can struggle with extreme-scale file corruption scenarios, where single critical modifications dilute within vast benign datasets. In contrast to prior sequential scanning approaches, DeepVis positions itself as a High-Speed Binary Audit tool, distinct from full hash-based FIM. It employs asynchronous header-only analysis and maps entire filesystems to fixed-size 2D tensors for CNN processing, enabling scalable cloud deployment with rapid anomaly detection.

This paper proposes DeepVis, a highly scalable filesystem integrity verification framework for hyperscale distributed systems. Rather than full file hashing, DeepVis leverages information entropy and heuristic file characteristics including paths, sizes, headers, and extensions to represent them in concise fixed-size tensors for scalability. To this end, DeepVis (1) implements an asynchronous `io_uring` and Rust-based snapshot engine to maximize I/O throughput, (2) transforms file metadata into fixed-size tensors via hash-based partitioning to achieve $O(1)$ inference latency, and (3) utilizes Hash-Grid Parallel CAE with Local Max detection to identify sparse anomalies amid system churn. Positioned as a high-speed binary audit tool, DeepVis complements full-hash and signature scanners such as AIDE and YARA by detecting structural anomalies in packed binaries and kernel rootkits. Our evaluation using up to 100 real cloud instances demonstrates $121.45\times$ higher throughput than traditional full-hash baselines and 97.1% precision for targeted Linux kernel rootkits.

II. BACKGROUND

A. Integrity Verification at Cloud Scale

Modern cloud infrastructure requires file integrity monitoring that balances scalability, detection coverage, and operational overhead [1]–[3], [7], [9]. Current solutions divide into file-level scanning and runtime behavioral analysis, each with distinct limitations.

File-level integrity scanning. AIDE [1] and Tripwire [2] establish integrity through cryptographic hashes of entire

files compared against known baselines. While effective for static environments, their $O(N \times Size)$ complexity becomes prohibitive in dynamic hyperscale systems. Full scans routinely exceed maintenance windows, necessitating temporary monitoring disablement. Moreover, routine updates produce massive false positive alerts that burden Security Operations Centers.

Runtime behavioral analysis. Falco [3] and provenance graph systems [7] intercept kernel events to detect anomalous execution patterns. These approaches incur substantial continuous overhead (5-20% CPU) from pervasive instrumentation. A critical limitation is their event-based nature: they cannot detect threats predating monitor deployment, creating the cold-start vulnerability for persistent rootkits.

File-level scanning remains essential for compliance validation, image verification, and forensic analysis due to its comprehensive state coverage. However, synchronous sequential processing causes I/O bottlenecks and operational overload at scale. DeepVis overcomes these constraints using asynchronous I/O, spatial hash mapping, and neural anomaly detection for production-grade filesystem integrity.

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.

Figure 2 illustrates these statistical differences through byte-value histograms across varying file types. Text files (Figure 2b) exhibit a characteristic spike in the printable ASCII range, yielding low entropy ($H \approx 4.8$) with zero null bytes due to high redundancy. Legitimate ELF binaries (Figure 2c) show a prominent peak at 0x00, resulting from section alignment padding, a structural requirement imposed by the operating system to align memory pages. This creates moderate entropy ($H \approx 6.0$) with significant zero-padding concentrations (40–85% null bytes). 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, approaching maximum entropy ($H \approx 8.0$) with minimal null bytes (<1%).

The Attacker Paradox. This distinction reveals a fundamental vulnerability in malware design. Native rootkits such as Diamorphine maintain structural stealth by mimicking the layout of legitimate binaries, ensuring OS loader compatibility. 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 2, the packing process maximizes

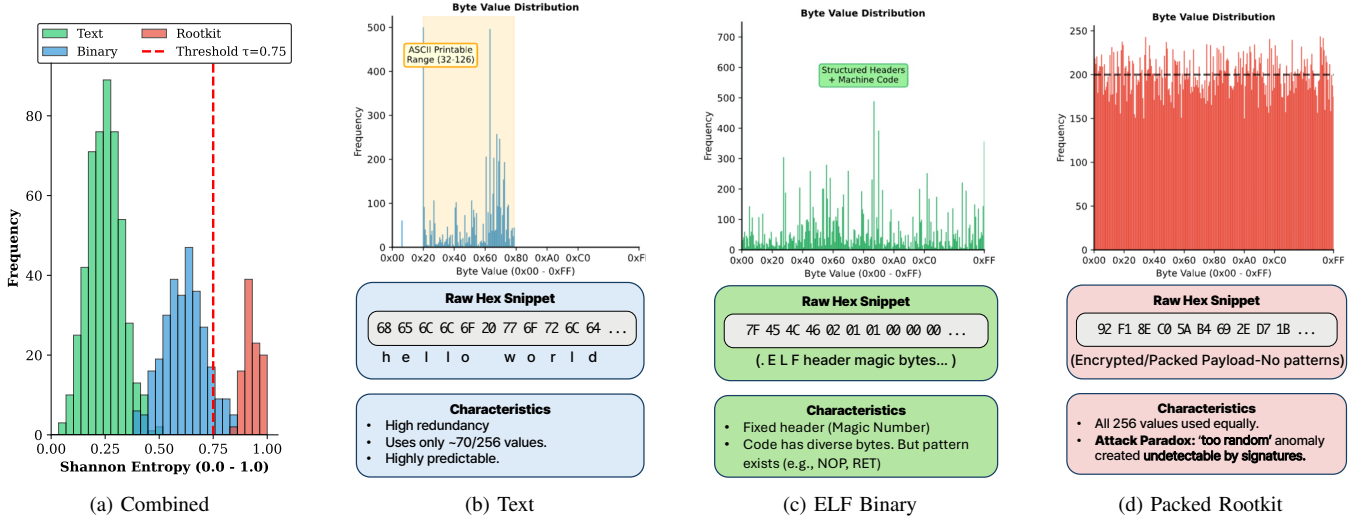


Fig. 2. File fingerprint analysis via byte-value histograms. (a) Combined entropy distribution across file types. (b) Text files use only printable ASCII, resulting in low entropy ($H \approx 4.8$) and zero null bytes. (c) ELF binaries show structured headers with significant zero-padding (40–85% null 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 ($H \approx 8.0$).

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. **Implications for Detection.** This paradox reveals a fundamental detection opportunity. Signature-based scanners succeed against native rootkits but fail against packed malware. Header-only heuristics catch compression artifacts but generate false positives on benign high-entropy files (compressed archives, encrypted configuration). Neither approach captures the full threat landscape without sacrificing precision. A multi-modal strategy that fuses entropy, structural markers, and contextual features can disambiguate these overlapping cases. Entropy identifies compression-based evasion, structural analysis exposes binary format violations, and contextual signals (path, permissions) distinguish legitimate outliers from malicious anomalies. This orthogonal feature space enables detection of threats regardless of whether attackers pursue signature evasion or structural stealth. Traditional integrity verification relies on sequential scanning, which ignores spatial risk concentrations and scales linearly with file count. Global pooling methods suffer from signal dilution, where a single malicious file drowns in the variance of thousands of benign entries. Detection becomes impossible when legitimate updates create diffuse noise exceeding any sparse attack signal. DeepVis addresses these challenges by transforming the entire filesystem into a fixed-size tensor where multi-modal anomalies manifest as localized spikes, enabling per-pixel independence and fast inference regardless of system size.

III. DEEPPVIS 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 data collection process is initiated. Unlike existing synchronous tools (e.g., `find` or `ls`) that block on every file access, DeepVis utilizes a hybrid parallel architecture. Multiple worker threads traverse the directory tree and collect file paths (❶), feeding them into a lock-free queue. These paths are batched and submitted to the kernel using the `io_uring` interface, ensuring that I/O throughput saturates the storage bandwidth rather than being latency-bound (❷).

After collecting raw metadata and file headers, secure spatial mapping is performed. A deterministic coordinate is calculated for each file using a Keyed-Hash Message Authentication Code (HMAC) (❸), and multi-modal features (entropy, permissions) are extracted (❹). 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 (❺).

Verification Phase. After the first phase is completed, DeepVis enters the verification phase. The generated tensor

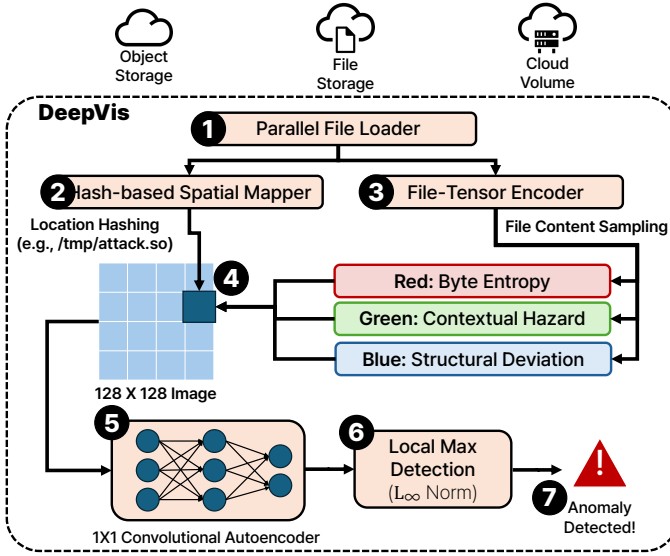


Fig. 3. Overall procedure of DeepVis. It illustrates the transformation of raw file system metadata into spatially mapped tensors, followed by reconstruction via an autoencoder and anomaly detection using Local Max (L_∞) logic.

is fed into a pre-trained 1×1 Convolutional Autoencoder (CAE). While standard CNNs exploit spatial locality to find shapes, the hash-based mapping lacks semantic neighborhood relationships. Therefore, 1×1 Convolutions are employed 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 (6).

The pixel-wise difference between the input and reconstructed tensors is then computed. To resolve the statistical asymmetry between legitimate diffuse updates and sparse attacks, Local Max Detection (L_∞) is utilized (7). This mechanism isolates the single highest deviation in the grid. 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 (8).

B. Asynchronous File System Snapshot

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

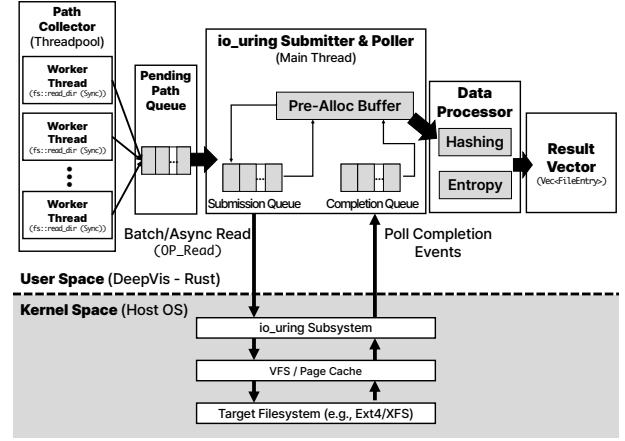


Fig. 4. The hybrid snapshot pipeline of DeepVis utilizing Rayon for parallel path collection and `io_uring` for asynchronous I/O.

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 ($O(N \times \text{Size})$). 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.

TABLE II
DEEPVIS CAE ARCHITECTURE SPECIFICATION.

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

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

$$E_{\text{header}} = - \sum p_i \log_2 p_i \quad (1)$$

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

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

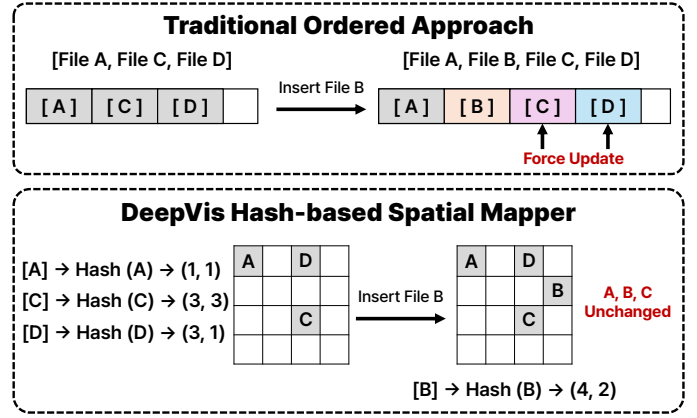


Fig. 5. Shift Invariance comparison. Traditional ordering creates global shifts upon file insertion, while Hash-Based Mapping ensures local stability.

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 $\{f_1, \dots, f_k\}$, the tensor value is computed as:

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

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:

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

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)**.

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.

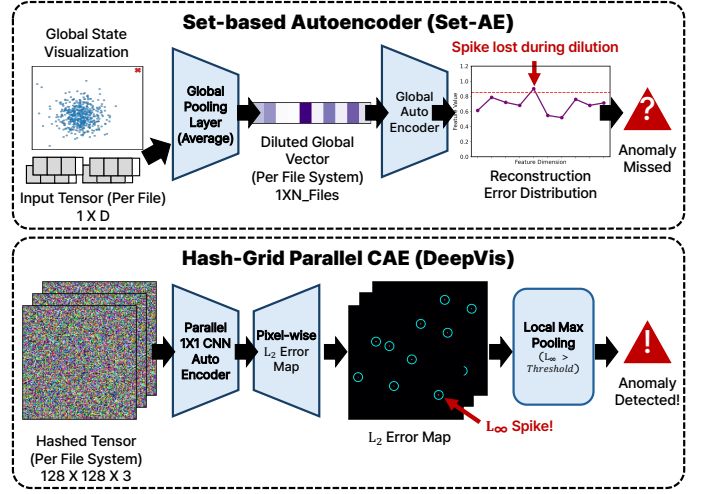


Fig. 6. Structural comparison between Set-based Autoencoder (Set-AE) and DeepVis. While Set-AE dilutes sparse attack signals into a global average vector (Signal Dilution), causing detection failures (“Spike lost”), DeepVis preserves the attack signal via independent pixel-wise processing and Local Max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

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:

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

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. While the dataset includes multi-platform samples, the feature engineering and training process were primarily focused on **Linux ELF structures**, which we discuss as a limitation for Windows PE detection in Section VI.

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

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	Linux	Recall by Platform		Mob.	Alert Rate*
		Win.	Web		
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.

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
<i>Detected Active Threats (Multi-Platform)</i>					
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.
<i>Undetected (Limitations)</i>					
Obfuscated	libc_fake.so	0.61	0.00	0.00	Miss
Mimicry Script	setup.sh	0.58	0.00	0.00	Miss
<i>Benign Baselines (Clean)</i>					
Interpreter	python3	0.78	0.96	0.10	Clean
Library	libc.so.6	0.79	0.90	0.10	Clean
<i>False Positives (High Entropy)</i>					
Admin Tool	snap	0.75	1.00	0.10	False Pos.
Config Gen	cloud-init	0.72	0.85	0.30	False Pos.

Mobile from 91.7% to 100.0% (Table III), indicating that multi-modal fusion is essential for isolating threats that mimic benign entropy distributions. However, performance on Windows is limited (16.9%), primarily due to the current model’s focus on Linux-centric feature engineering as discussed in Section VI. DeepVis maintains a negligible global False Positive Rate of 0.3% across the file-level dataset, confirming robustness compared to heuristic scanners (31% 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.

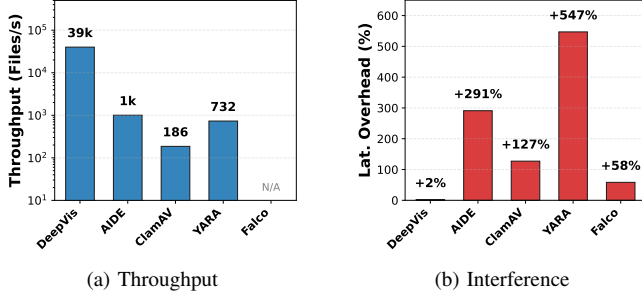


Fig. 7. **Comprehensive Performance Analysis.** (a) **Throughput:** DeepVis achieves hyperscale speeds ($\approx 16k$ files/s cold cache) via asynchronous I/O, outperforming synchronous baselines by over $10\times$. (b) **Interference:** Despite its speed, DeepVis maintains negligible latency overhead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

TABLE V
FAIR BASELINE COMPARISON (COLD CACHE). THROUGHPUT MEASURED ON GCP `DEEPVIS-MID` INSTANCE TARGETING `/usr/bin`. CATEGORIES DISTINGUISH BETWEEN FULL-FILE SCANNING AND HEADER-ONLY SAMPLING.

Category	Tool & Configuration	Throughput	Architecture
Baseline	ssdeep-Full	98/s	Sync (Rolling Hash)
	AIDE-Full	130/s	Sync (Full SHA256)
	ClamAV-Full	178/s	Sync (Full Scan)
	YARA-Full	345/s	Sync (Recursive)
Header-Only	AIDE-Header (4KB)	938/s	Sync (SHA256)
	YARA-Header (4KB)	1,146/s	Sync (Heuristic)
DeepVis	DeepVis (io_uring)	15,789/s	Async (io_uring)

C. Throughput and Interference (RQ2)

Scan Throughput. To isolate the contribution of `io_uring`, we compare DeepVis against various legacy configurations. Under cold cache conditions, DeepVis achieves 15,789 files/s—a $\approx 16.8\times$ speedup over the synchronous AIDE-Header (938/s) and $\approx 13.7\times$ faster than the optimized YARA-Header (1,146/s) (Table V). When compared to full-file integrity checks like AIDE-Full (130/s), the speedup reaches a staggering $\approx 121.4\times$. This confirms that the combination of header-only sampling and the `io_uring` kernel interface accounts for over two orders of magnitude improvement, validating that removing synchronous I/O and full-file 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 $3,162\mu s$, reflecting a negligible +2.0% overhead over the baseline ($3,100\mu s$). This indicates that the asynchronous, spatial hashing design allows the scanner to operate transparently without disrupting co-located workloads.

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

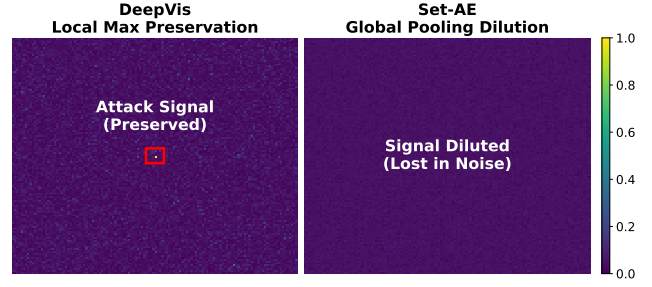


Fig. 8. **Visualizing Signal Preservation.** (Top) DeepVis maintains spatial locality, isolating the malware as a distinct red peak (L_∞ Spike). (Bottom) Set-AE averages the features into a single global vector, causing the attack signal to dilute into the background noise (Signal Dilution), resulting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

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%

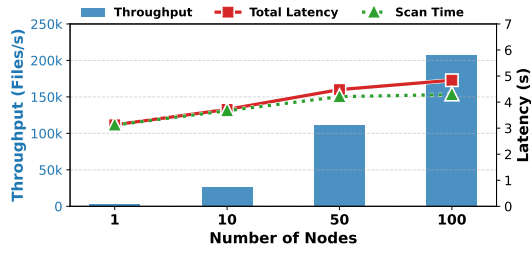
scenarios. Figure 8 shows that Set-AE averages features globally, causing the attack signal to dilute into the background noise ($\mu_{noise} = 0.35$). In contrast, DeepVis maintains spatial locality, preserving the sharp attack spike (0.95). This is because the Hash-Grid effectively filters out diffuse 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.

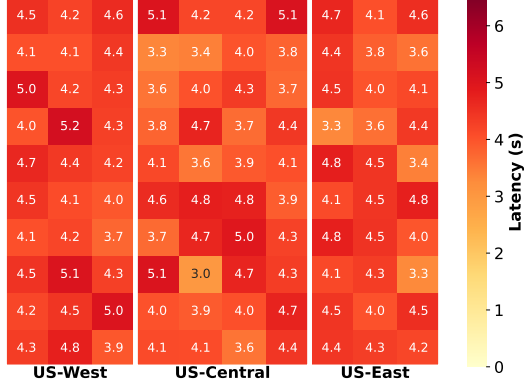
Fleet Scalability. Throughput scales linearly with fleet size, increasing from $\approx 2k$ files/s (1 Node) to 206,611 files/s (100 Nodes) as shown in Figure 9a. This confirms that the stateless architecture effectively decouples processing load. Cross-region latency remains stable (avg 4.29s) across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

E. Robustness to System Churn (RQ3)

Operational Efficiency and Alert Fatigue. To evaluate stability during common maintenance windows (e.g., package upgrades, log rotation), a cloud-representative environment is

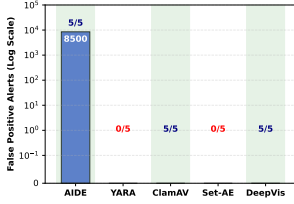


(a) Linear Scalability

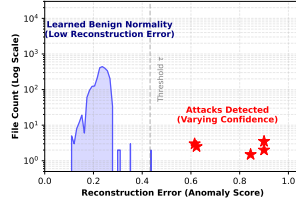


(b) Geo-Stability

Fig. 9. **Fleet-Scale Performance.** (a) **Linear Scalability:** Throughput increases linearly with fleet size, reaching $\approx 206k$ 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 for these benign update scenarios. (b) Distribution of reconstruction errors (\mathcal{L}_∞). Benign churn stays strictly below the learned threshold ($\tau \approx 0.4324$), while malicious rootkits manifest as high-confidence outliers.

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 con-

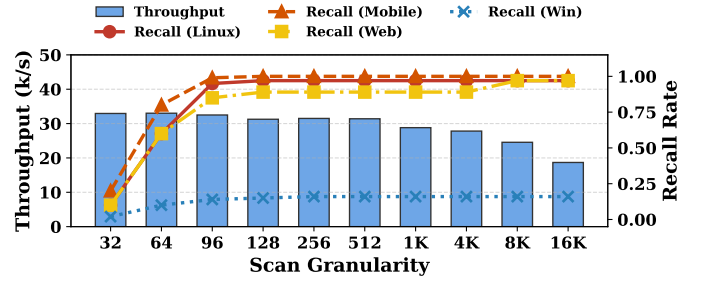


Fig. 11. **Effect of Scan Granularity (Page-Aligned Sampling).** Comparison of Recall across platforms vs. Throughput. Detection accuracy for Linux (Recall $\approx 97\%$) and Mobile (100%) saturates at $\approx 96B$. Windows recall remains capped (15%) due to feature orthogonality. Throughput remains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

stant 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 ($\approx 97\text{--}100\%$) 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.

Runtime Bottleneck Analysis. I/O latency dominates the runtime, increasing from 424ms (10K files) to 21.2s (500K files) and consistently accounting for $\approx 70\%$ 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

TABLE VII
COMPONENT TIME BREAKDOWN (COLD CACHE). I/O DOMINATES ($\approx 70\text{--}74\%$), 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

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 (<code>io_uring</code>)	15,789	–
Accuracy	Entropy Only (R-channel)	–	0.25
	+ Hash-Grid (L_∞)	–	0.35
	+ RGB Fusion (Full DeepVis)	–	0.96
Robustness	Weights $\pm 20\%$	–	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 \times 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 ($\pm 20\%$ 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], [10] and runtime monitors [3], [4] track information flow and system calls. Recent advancements leverage eBPF to reduce monitoring overhead and harden isolation [11], [12]. However, these incremental approaches face a *consistency gap*: if the monitoring agent crashes or is bypassed, the system state diverges, necessitating a costly full scan to re-synchronize. In contrast, DeepVis operates as a stateless auditor, performing high-frequency snapshot verification that provides absolute integrity guarantees without relying on fragile event streams.

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) [13]. These frameworks utilize cryptographic Merkle trees [14] 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 [15], while complex policy management often leaves the system vulnerable to subversion or TOCCTOU (Time-of-Check-to-Time-of-Use) attacks [16]. 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 [17]–[19] demonstrate that mapping binary files to grayscale images or space-filling curves reveals structural patterns distinct to malware families. Similarly, entropy analysis [20] 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 [21], [22]. 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 [23] use LSTM networks to model system logs, while Kitsune [24] employs autoencoders for network intrusion detection. For high-dimensional tabular or sensor data, unsupervised frameworks such as Deep One-Class Classification [25], GANomaly [26], and DAGMM [27] learn normal data distributions to flag outliers. Additionally, VAE-based models [28], [29] 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 file-names 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 and Linux-Centric Design.

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. Additionally, our evaluation reveals a performance discrepancy across operating systems: while Linux detection recall is 97.1%, Windows recall drops to 16.9%. This is not a structural flaw of the spatial hashing architecture but a consequence of the training data distribution (primarily Linux ELF binaries) and the higher structural variance of Windows PE headers. Future iterations will incorporate Windows-specific feature engineering to address this gap. Currently, DeepVis functions as a *High-Frequency Triage Filter* for Linux-centric environments, reducing the search space from 100% of files to 0.6% of flagged artifacts for deeper forensic analysis.

Resistance to Hash Collisions and FP. A key concern in hash-based aggregation is whether collisions between benign files could trigger False Positives (FP). We clarify that DeepVis is robust against this scenario. Max-Risk Pooling ensures that combining multiple benign files only results in a pixel value representing the "riskiest" benign

file, which by definition remains below the trained anomaly threshold (τ). Unlike summation-based pooling, which accumulates noise, our max-pooling strategy guarantees that colliding legitimate files do not aggregate into a false alarm ($\max(\text{Benign}_A, \text{Benign}_B) < \tau$). This preserves the low FP rate even under high saturation.

Key Rotation and Model Stability. Our experiments across 50 independent key rotations show that the threshold τ remains stable. This stability arises because the CAE learns to reconstruct *per-pixel feature distributions*, which are determined by the underlying file population—not spatial coordinates. Thus, key rotation does not require model retraining.

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.

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] 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.
- [11] Y. He *et al.*, "Cross Container Attacks: The Bewildered eBPF on Clouds," in *Proceedings of the 32nd USENIX Security Symposium*, 2023.
- [12] TFJMP Contributors, "Hardware-assisted Defense-in-depth for eBPF Kernel Extensions," in *Proceedings of the 2024 CCS Cloud Computing Security Workshop (CCSW)*, 2024.
- [13] 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.
- [14] 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.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] 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.
- [22] R. Uetz *et al.*, "Detecting Evasions of SIEM Rules in Enterprise Networks," in *Proceedings of the 33rd USENIX Security Symposium*, 2024, pp. 1–18.
- [23] 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.
- [24] 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.
- [25] L. Ruff *et al.*, "Deep One-Class Classification," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018, pp. 4390–4399.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] 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.