

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

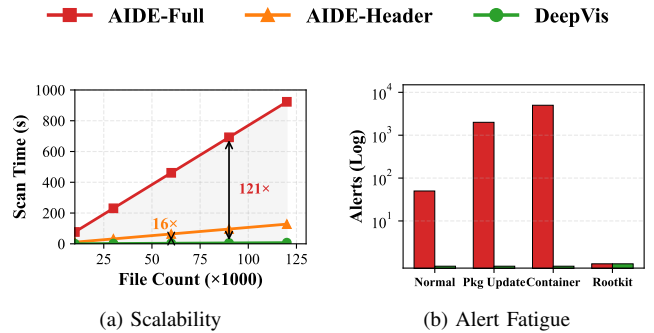


Fig. 1. (a) **DeepVis** achieves high-throughput saturation via async I/O. (b) Legitimate updates generate false alerts in AIDE.

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 $O(N \times \text{Size})$ 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 (Executable and Linkable Format) binaries (Figure 2c) display characteristic 0x00 peaks resulting from operating system requirements for section alignment padding. The ELF structure enforces 4KB page alignment for distinct sections such as `.text` (code) and `.data` (variables). Compilers therefore insert null-byte padding (0x00) to satisfy these boundaries, producing moderate entropy ($H \approx 6.0$) and 40–85% null byte concentration. 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. Signature evasion prompts attackers to employ



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 ($H \approx 8.0$).

UPX (Ultimate Packer for eXecutables), an open-source tool that compresses executables by 50–70% across Windows, Linux, and macOS formats. UPX prepends decryption stubs to compressed ELF binaries, effectively concealing signatures. However, this process eliminates section alignment padding and produces uniform byte distributions, obliterating the structural fingerprint of legitimate files.

As shown in Figure 2, 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.

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. 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 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

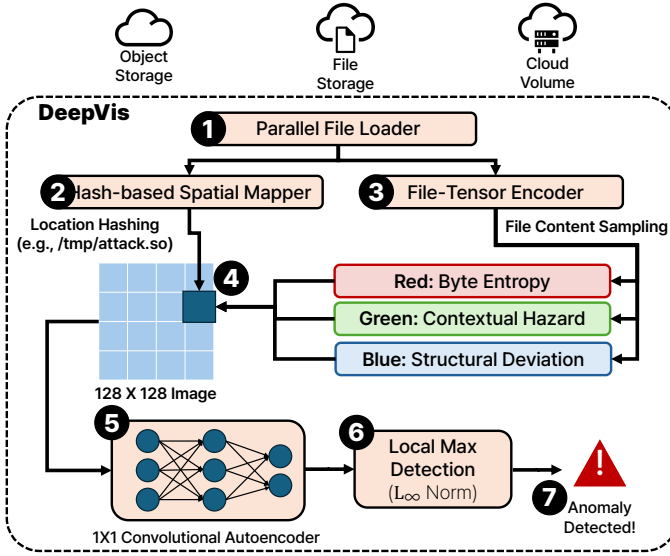


Fig. 3. Overall procedure of DeepVis. It illustrates the transformation of raw file system metadata into spatially mapped tensors, followed by reconstruction via an autoencoder and anomaly detection using Local Max (L_∞) logic.

for each file using a Keyed-Hash Message Authentication Code (HMAC) (③), and multi-modal features (entropy, permissions) are extracted (④). 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 (⑤).

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

Before generating the tensor representation, DeepVis processes metadata and file headers across thousands of cloud instances. Existing synchronous system calls (e.g., `stat`, `open`, `read`) cause context switching overhead and CPU blocking at scale. Network-attached storage in cloud environments exacerbates I/O latency. DeepVis overcomes this with a hybrid pipeline separating CPU-bound path traversal from I/O-bound data reading.

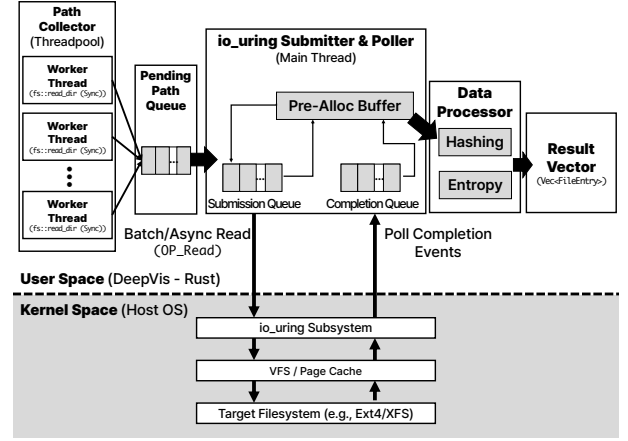


Fig. 4. Hybrid snapshot pipeline of DeepVis using Rayon for parallel path collection and `io_uring` for asynchronous I/O.

Parallel Path Collection. DeepVis uses work-stealing parallelism from the Rust `rayon` library for path collection. As shown on the left of Figure 4, the *Path Collector Threadpool* spawns worker threads that execute synchronous `fs::read_dir` operations recursively. This CPU-bound phase parses directory entries and builds path strings across all cores, filling the *Pending Path Queue* faster than I/O consumption.

Asynchronous I/O Processing. With paths enqueued, reading file headers becomes the bottleneck. DeepVis employs Linux `io_uring` to eliminate per-file system call overhead. As shown in Figure 4, the *io_uring Submitter* batches paths into Submission Queue (SQ) read requests (`OP_READ`). Unlike traditional async I/O, `io_uring` uses shared ring buffers for kernel-user communication. *Data Processor* threads poll the Completion Queue (CQ) for finished reads. Completed events trigger data retrieval from pre-allocated *Buffer Slabs*, followed by immediate hashing and entropy calculation. CPU threads never block on disk I/O. The kernel handles data movement while user-space processes features. This achieves throughput competitive with raw disk bandwidth.

C. Header Sampling

Traditional FIM tools hash entire files, causing massive I/O overhead ($O(N \times \text{Size})$). Conversely, metadata-only scanning (e.g., file size, name) produces high false negatives against padded malware. To balance these extremes, DeepVis adopts header-based entropy sampling.

As discussed in Section II, packed malware and ransomware inevitably modify file headers to accommodate unpacker stubs or encrypted payloads, significantly increasing entropy in the first few blocks. To detect this, DeepVis reads only the first 4KB of each file asynchronously. File systems use 4KB block size, so sub-4KB requests incur full 4KB I/O padded with zeros, while larger reads require additional requests per file. As Linux headers reside in the first 128 bytes, evaluation results show 4KB suffices for most malicious files.



Fig. 5. Shift invariance comparison. Top: Traditional ordered approach creates catastrophic global index shifts when File D inserts between B and C. Bottom: Hash-based mapping ensures local stability—only affected pixels change positions independently.

Reading the first 4KB enables header sampling to excel against executable malware requiring loader compatibility (ELF/PE). Packed binaries, ransomware, and rootkits must alter headers for unpackers or encrypted payloads, unlike data files hiding payloads at arbitrary offsets. This 4KB page-aligned approach reduces per-file I/O independent of file size while retaining binary format anomaly sensitivity, providing high-frequency first-line defense.

D. Hash-Based Spatial Mapping

Spatial Invariance. After asynchronously reading metadata and 4KB header blocks, DeepVis must transform thousands of unordered files into a fixed-size tensor for neural processing. Traditional ordering-based approaches (e.g., alphabetical sorting by path or directory tree) suffer from the Ordering Problem. Inserting a single new file shifts the indices of all subsequent files, destroying spatial locality across scans and invalidating trained neural network models. To overcome this, DeepVis employs a deterministic hash-based spatial mapping to project each file onto a fixed-size grid.

Figure 5 illustrates the comparison between the traditional ordered approach and the hash-based mapping employed by DeepVis. As depicted in the upper panel, inserting File B forces updates to the indices of files C and D. This cascading shift becomes critically expensive in large-scale cloud systems containing millions of files. In contrast, the lower panel demonstrates how DeepVis calculates stable coordinates $\Phi(p)$ for each file path p . By utilizing a high-entropy secret key K generated at startup, the system maps files onto a fixed-size 128×128 tensor. This process generates a uniform representation of the file system state and facilitates the visualization of system health.

To derive coordinates for each file, DeepVis employs the first 32 bits of the HMAC output modulo 128 for the x-coordinate and the subsequent 32 bits (bits 32 to 64) modulo 128 for the y-coordinate. The utilization of HMAC to establish stable coordinates provides two critical benefits.

First, deterministic bucket mapping ensures that coordinates depend solely on the file path and the secret key, producing reproducible 128×128 grids across successive scans. Second, cryptographic key K defeats targeted mapping attacks. Adversaries cannot craft filenames to reach specific low-risk coordinates. Ephemeral keys and privileged memory access restrict K extraction.

Multi-Modal RGB Encoding Within the hash-mapped coordinate space, DeepVis encodes each pixel through three risk channels: Red, Green, and Blue. These represent file characteristics for image-based visualization. The channels leverage malware feature orthogonality, ensuring evasion of one channel increases risk in others.

leftmargin=*

- **R (Entropy)** The Shannon entropy of the 4KB file header exploits the inherent trade-off in malware design. Legitimate ELF binaries incorporate zero-padding for section alignment, resulting in low entropy, whereas packed malware employs high information density to obfuscate signatures. This channel effectively distinguishes packed threats from standard system executables.
- **G (Context Hazard) as Spatial Prior.** To eliminate heuristic ambiguity, spatial risk is quantified using a deterministic mapping derived from the Filesystem Hierarchy Standard (FHS). A risk function $G(p) \in [0, 1]$ is defined for a file path p :

$$G(p) = \min \left(1.0, \sum_{k \in \mathcal{K}_{\text{FHS}}} w_k \cdot \mathbb{I}(p \in \text{Zone}_k) + \alpha \cdot \text{Depth}(p) \right) \quad (1)$$

where \mathcal{K}_{FHS} represents standard directory roles (e.g., volatile memory `/dev/shm` [$w = 0.9$] vs. static binaries `/usr/bin` [$w = 0.1$]). These specific values maximize the separation between immutable system paths and writable attack surfaces. By anchoring weights w_k to the FHS violation probability rather than arbitrary tuning, this channel acts as a hard attention mechanism, forcing the model to prioritize structural anomalies in locations deemed volatile by OS standards.

- **B (Structure) as Compliance Distance.** This channel quantifies the distance from specification compliance rather than employing arbitrary header parsing. Kernel modules (`.ko`) and shared objects (`.so`) residing outside system-privileged paths (e.g., `/lib/modules`) are assigned $B = 1.0$. This metric is not a heuristic but a direct boolean constraint reflecting OS loader policies: relocatable code in user-writable paths is inherently anomalous. This compresses complex loader logic into an $O(1)$ lookup.

This RGB encoding transforms the abstract file system state into a dense numerical tensor $T \in \mathbb{R}^{128 \times 128 \times 3}$. This transformation enables the downstream Hash-Grid Parallel CAE to learn complex cross-channel correlations. Unlike simple linear thresholding, which fails to capture conditional dependencies (e.g., high entropy is permissible in static binaries but anomalous in temporary paths), the CAE learns the non-linear



Fig. 6. Set-AE vs. Hash-Grid CAE. Top: Set-AE global pooling dilutes single malicious signal among benign files (Spike Lost). Bottom: Hash-Grid CAE processes 16K pixels independently; L_∞ pooling captures isolated spikes.

manifold of valid structural states. The scanner maintains an inverted index that maps each pixel coordinate back to its constituent file paths. This ensures that operators can attribute the violation to a specific file once an anomaly is detected.

Mapping Robustness and Security. Since the grid size remains fixed while cloud systems scale to massive file counts, hash collisions become inevitable. DeepVis addresses both natural collisions and adversarial targeting by projecting files onto a fixed 128×128 grid and applying max-risk pooling, where each pixel retains the maximum feature value across all colliding files per RGB channel. A single high-risk file therefore determines the pixel value, ensuring that malicious signals dominate despite benign collisions. To prevent bucket targeting attacks, where adversaries engineer filenames to land in specific coordinates, the system uses a high-entropy secret key K and periodically rotates it, shuffling the entire grid without retraining because downstream 1×1 convolutions are spatially agnostic.

E. Hash-Grid Parallel CAE

Signal Preservation via Parallel Processing. To ensure robust detection in hyperscale environments, DeepVis fundamentally alters the aggregation strategy by abandoning global pooling in favor of a parallelized pixel-wise architecture. As illustrated in the top panel of Figure 6, traditional models such as Set-AE aggregate features into a single global vector. This architectural choice results in signal dilution, where the distinct feature vector of a single compromised file is aggregated with an increasing volume of benign metadata. In a hyperscale scenario where $N_{benign} \gg N_{malicious}$, the attack spike drowns in the statistical variance of millions of benign files, rendering detection impossible. In contrast, as shown in the bottom of Figure 6, the Hash-Grid Parallel CAE processes the 128×128 spatial grid as independent feature channels. By utilizing 1×1 convolutions, the model maintains strict isolation

between pixels throughout the encoding and decoding phases. This design prevents error propagation from benign regions to malicious ones, ensuring that the structural footprint of a rootkit remains mathematically distinct regardless of the total filesystem size.

Pixel-Wise Reconstruction and Error Map Generation.

The core of the detection pipeline is the transformation of the raw system snapshot into a fine-grained anomaly map without cross-pixel contamination. The input tensor is fed into a 1×1 Convolutional Autoencoder which effectively functions as a massive array of shared Multi-Layer Perceptrons (MLP) operating simultaneously on all 16,384 pixels. The encoder utilizes point-wise layers (Enc: $3 \rightarrow 16 \rightarrow 8$) to compress the RGB channels into a latent representation that captures the manifold of valid file states, while the decoder ($8 \rightarrow 16 \rightarrow 3$) attempts to reconstruct the original input. Crucially, because the convolution kernel size is 1×1 , the reconstruction of any given pixel depends solely on its own depth-wise features (Entropy, Context, Structure) and is uninfluenced by its neighbors. The system then computes the element-wise difference between the input snapshot and the reconstructed output to generate a Pixel-wise L_2 Error Map. This map explicitly visualizes the deviation intensity for every coordinate in the grid, isolating specific files that violate the learned structural norms of the fleet.

Solving the MSE Paradox with L_∞ Scoring. The final detection phase addresses the MSE Paradox where legitimate system updates generate high global error while stealthy attacks generate low global error. Production environments are characterized by frequent legitimate updates such as package upgrades, which introduce diffuse noise across the grid. Using global Mean Squared Error (MSE) metrics in this context misclassifies benign churn as anomalous because the cumulative error of thousands of legitimate updates exceeds the error of a single stealthy rootkit. DeepVis resolves this by applying Maximum Deviation (L_∞) pooling over the generated error map. By calculating the score as $Score = \max_{i,j} |T_{i,j} - T'_{i,j}|$, the system ignores the cumulative sum of low-magnitude noise and focuses exclusively on the single highest anomaly peak. As shown in the bottom of Figure 6, this strategy captures the L_∞ spike generated by the malicious file while filtering out the background noise of valid system updates. This approach decouples detection sensitivity from the volume of benign churn, allowing the system to isolate the most distinct structural violation within the file system.

Anomaly Decision and Unsupervised Calibration. To distinguish malicious anomalies from benign variations, DeepVis establishes a decision boundary based on the reconstruction capability of the CAE. The system defines an anomaly condition as $Score > \tau$, where τ represents the maximum reconstruction error observed in a clean environment. To derive this threshold, we perform an offline calibration using a validation dataset consisting exclusively of benign Linux ELF binaries. The system processes each benign file to generate a tensor, passes it through the trained CAE, and calculates the L_∞ score from the resulting error map. The threshold τ is then set to the

highest L_∞ score recorded across this validation corpus. This method effectively measures the worst-case structural variance of legitimate software that the model can reconstruct. While detection on other platforms such as Windows or Android would require learning their specific binary formats (e.g., PE or DEX), this study limits its scope to Linux to strictly evaluate efficacy against server-grade rootkits. Consequently, during online monitoring, any input producing a residual error exceeding this calibrated maximum indicates a data pattern that deviates from the learned characteristics of the Linux ELF format, triggering a security alert.

F. DeepVis Implementation

DeepVis employs a hybrid Rust-Python architecture to reconcile the conflicting requirements of low-level I/O throughput and high-level deep learning flexibility. The high-performance core is implemented as the `deepvis_scanner` Rust module, which orchestrates asynchronous filesystem operations to minimize kernel-to-user context switching. This module leverages the Linux `io_uring` interface with a submission queue depth of 512, enabling the batching of 4KB header reads to saturate storage bandwidth. To maintain pipeline saturation, the `rayon` library manages a work-stealing thread pool for parallel path traversal, ensuring that metadata ingestion fills the pending queue faster than the I/O subsystem can consume it. Within this native layer, the system executes computationally intensive feature engineering tasks—including Shannon entropy calculation, HMAC-based coordinate generation, and max-risk tensor construction ($3 \times 128 \times 128$)—effectively offloading overhead from the Python interpreter.

To interface with the anomaly detection logic, `pyo3` bindings expose the `DeepVisScanner` class, providing a direct Foreign Function Interface (FFI) to the underlying Rust engine. This abstraction enables zero-copy data transfer of the constructed tensors via methods such as `scan_to_tensor()`, facilitating immediate integration with PyTorch inference engines or ONNX runtimes on CPU-constrained edge devices. Furthermore, the implementation incorporates operational heuristics to ensure stability in production environments, such as the automatic exclusion of volatile pseudo-file systems (e.g., `/proc`, `/sys`, `/dev`) and the synchronization of CPU-bound path discovery with kernel-level data movement. This design guarantees that the memory-safe Rust backend handles all data-intensive operations while the Python frontend retains flexibility for model deployment and threshold management.

IV. EVALUATION

This section evaluates DeepVis on Google Cloud Platform (GCP) using real rootkits and realistic attack scenarios. The 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, the file system is populated with up to 10 million files, including system binaries (e.g., `nginx`) and random artifacts.

The experimental setup employs a *Benign Baseline* of 47,270 system binaries (Ubuntu 20.04) to measure false positives, and a *Malware Corpus* of 37,387 files. The malware corpus includes 68 functional Linux kernel rootkit samples derived from five major families: Diamorphine [10], Rep-tile [11], and variants from public malware repositories [12], [13], compiled against various kernel versions to create distinct binary signatures. This corpus is supplemented by 35k+ Windows/Web artifacts as cross-platform controls. To strictly validate the structural analysis engine, feature engineering targets the Linux ELF format. Windows PE artifacts serve as a control group to test cross-platform feature orthogonality, as discussed in Section VI.

Context Hazard (G) Weights. For reproducibility, the Context channel employs a weighted path taxonomy based on the Filesystem Hierarchy Standard (FHS). System binaries in standard paths (e.g., `/usr/bin`) receive $w = 0.1$, user directories receive $w = 0.3$, temporary paths (`/tmp`) receive $w = 0.6$, and volatile memory locations (`/dev/shm`) receive $w = 0.9$. Hidden files add $+0.2$ and deep nesting (>5 levels) adds $+0.1$ per level. All scores are normalized to $[0, 1]$ before RGB fusion. These static weights ensure deterministic scoring across deployment environments.

Baselines. Traditional monitors include AIDE [1], YARA [14], ClamAV [6], and ssdeep [15]. For deep learning baselines, we evaluate MalConv [16], a 1D-CNN operating on raw byte sequences (up to 2MB); Byte-plot CNN [17], which visualizes binaries as grayscale images for 2D-CNN classification; and Pack-ALM [18], a Transformer-based architecture for packed malware detection.

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 II), 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 focus on Linux-centric feature engineering as discussed in Section VI. DeepVis maintains a raw anomaly rate of 0.3% across the static dataset. Crucially, these anomalies are structural outliers (e.g., highly packed binaries) rather than erroneous security alerts, as confirmed by the zero operational false positives in the churn analysis (Section IV-E). The low Alert Rate (0.3%) on the benign baseline corresponds to a Precision of approximately 99.7%. Combined with the 97.1% Recall on Linux threats, this yields the F1-Score of 0.96 reported in the ablation study (Table VII). Compared to deep learning baselines, DeepVis outperforms Byte-plot CNN (29.2%) and

TABLE II

UNIFIED DETECTION PERFORMANCE. COMPARISON AGAINST TUNED BASELINES AND MODERN DL MODELS. DEEPVIS (FULL) ACHIEVES SUPERIOR RECALL ON LINUX/MOBILE THREATS WHILE ENSURING ZERO OPERATIONAL FPS. ALERT RATE IS REPORTED AT THE NODE/REPOSITORY LEVEL (0.3% PER 10K FILES).

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%
Byte-plot CNN	29.2%	9.8%	52.1%	95.8%	0.5%
MalConv	72.8%	55.0%	91.0%	100.0%	0.8%
Pack-ALM	49.8%	22.2%	74.3%	100.0%	1.2%
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. [◊]The 100% rate of AIDE 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.

Pack-ALM (49.8%) on Linux while maintaining a lower Alert Rate than MalConv (0.8%). This demonstrates that lightweight structural features can match or exceed the detection capability of heavyweight neural architectures on the target platform.

Feature Orthogonality. Micro-analysis confirms that multi-modal features catch evasion attempts. As shown in Table III, 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. DeepVis is designed specifically to detect structural anomalies (packing, encryption, ELF manipulation) typical of binary rootkits, not semantic anomalies in text-based scripts. Script-based attacks such as `setup.sh` typically rely on standard interpreters (e.g., `/bin/bash`), which DeepVis monitors for structural tampering, or leave filesystem traces in high-risk paths that the Context channel captures. This explicit scope trade-off is necessary to achieve the 15k+ files/s throughput required for hourly fleet-wide scans.

C. Throughput and Interference (RQ2)

Scan Throughput. To explicitly isolate the architectural benefits of `io_uring` from I/O reduction, we constructed Optimized Synthetic Baselines (AIDE-Header, YARA-Header). These configurations represent a theoretical best-case scenario for legacy tools restricted to header scanning. Under cold cache conditions, DeepVis achieves 15,789 files/s, representing a $16.8\times$ speedup over the synchronous AIDE-Header (938/s) and $13.7\times$ faster than the optimized YARA-Header (1,146/s) (Table IV). When compared to full-file integrity checks like AIDE-Full (130/s), the speedup reaches $121.4\times$. This confirms that the combination of header-only sampling and the `io_uring` kernel interface accounts for over two orders of magnitude improvement. This throughput enables hourly scanning cycles that were previously computationally infeasible with synchronous blocking I/O.

TABLE III

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.

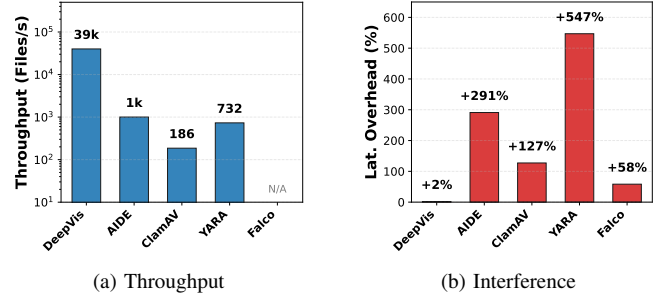


Fig. 7. **Comprehensive Performance Analysis.** (a) **Throughput:** DeepVis achieves hyperscale speeds ($\approx 16k$ files/s cold cache) via asynchronous I/O, outperforming synchronous baselines by over $10\times$. (b) **Interference:** Despite its speed, DeepVis maintains negligible latency overhead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

Deep learning models exhibit even more severe bottlenecks. MalConv requires approximately 1 second per file for inference, while Transformer-based architectures like Pack-ALM exceed 2 seconds per file. Scanning a modern Linux distribution (500k files) with Pack-ALM would take approximately 11 days, whereas DeepVis completes the same task in under 32 seconds. This throughput difference (approximately $30,000\times$) is not merely a matter of optimization but a fundamental architectural shift: by extracting highly representative structural features from the file header, DeepVis moves the complexity from $O(N)$ model execution to $O(1)$ metadata verification, making it the only viable candidate for continuous, high-fidelity system scanning.

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 base-

TABLE IV
FAIR BASELINE COMPARISON (COLD CACHE). THROUGHPUT
MEASURED ON GCP DEEPLIS-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)
Deep Learning	Byte-plot CNN	130/s	2D-CNN (Img)
	MalConv	1.1/s	1D-CNN (2MB)
	Pack-ALM	0.5/s	Transformer
DeepVis	DeepVis (io_uring)	15,789/s	Async (io_uring)

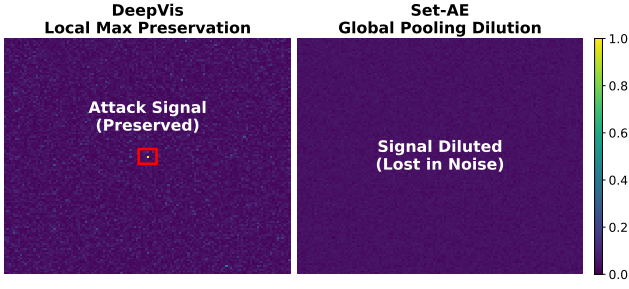


Fig. 8. **Visualizing Signal Preservation.** (Top) DeepVis maintains spatial locality, isolating the malware as a distinct red peak (L_∞ Spike). (Bottom) Set-AE averages the features into a single global vector, causing the attack signal to dilute into the background noise (Signal Dilution), resulting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

line (3,100 μ s). This indicates that the asynchronous, spatial hashing design allows the scanner to operate transparently without disrupting co-located workloads.

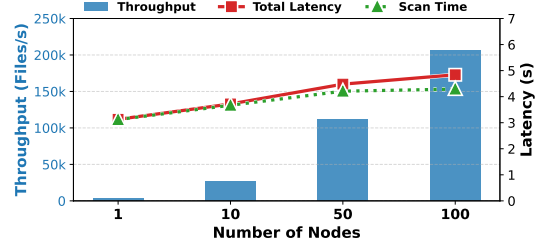
D. Scalability and Saturation Analysis (RQ3, RQ6, RQ7)

Signal Preservation. The parallel architecture prevents the variance of benign files from impacting the reconstruction error of the malicious target. 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). The Hash-Grid effectively filters out diffuse noise, ensuring robust detection even during high churn.

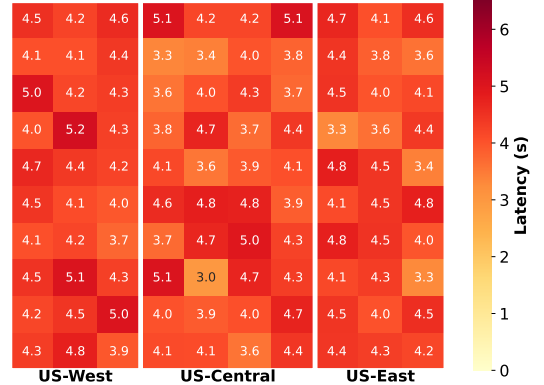
Resilience to Saturation. Recall remains at 100% even as file count increases to 10 million (Table V). Despite the grid saturation reaching 100% with over 610 collisions per pixel, the Max-Risk Pooling strategy ensures the high-risk anomaly dominates the pixel value. Since the pooling operation is non-linear (max), even an adversary flooding the bucket with thousands of benign decoys cannot mathematically lower the pixel value contributed by a single high-risk artifact. While Max-Risk Pooling is mathematically robust against stochastic collisions, targeted pre-image attacks represent a distinct threat model. Generating a targeted collision requires knowledge of the specific grid mapping function, which depends on a secret

TABLE V
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 ≈ 206 k 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.

key K that is randomized per deployment. Without leakage of the mapping seed, blind adversarial flooding is statistically ineffective against the sparse L_∞ signal. 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 ≈ 2 k 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.

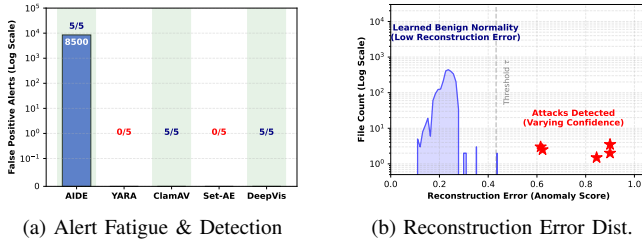


Fig. 10. **Fleet-Scale Churn & Alert Fatigue Analysis.** (a) Comparison of False Positive alerts and Detection Recall. AIDE generates 8,500 alerts during maintenance, whereas DeepVis maintains zero false positives for these benign update scenarios. (b) Distribution of reconstruction errors (\mathcal{L}_∞).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

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 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 prohibitive overhead, triggering 8,500 false alerts per scan due to its reliance on brittle hash comparisons that flag any state change regardless of benign intent. 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 reduces the operational burden on security analysts. **Fleet Error Stability vs. Forensic Signal.** As shown in Figure 10b, the underlying mechanism for this efficiency is the robust stability of DeepVis, where *Benign Churn* forms a continuous distribution strictly below the learned threshold ($\tau \approx 0.4324$). This stability arises because the feature fusion 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 minimal 96 B scan size (Fig. 11). This

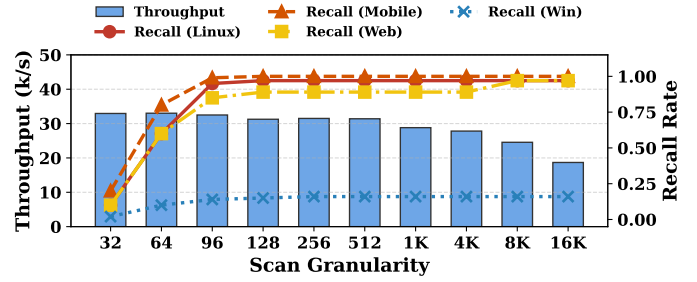


Fig. 11. **Effect of Scan Granularity (Page-Aligned Sampling).** Comparison of Recall across platforms vs. Throughput. Detection accuracy for Linux (Recall $\approx 97\%$) and Mobile (100%) saturates at ≈ 96 B. Windows recall remains capped (15%) due to feature orthogonality. Throughput remains stable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

indicates that minimal header data suffices for robust feature extraction. Throughput shows a general decline as scan size increases but remains stable up to 4 KB. This stability stems from OS-level page prefetching, where reading small chunks (e.g., 32 B) incurs similar I/O costs to reading a full 4 KB page. This empirical result validates the design choice of the *4KB Page-Aligned Sampling* strategy, confirming that prefetching mechanisms negate the latency benefits of sub-page reads.

TABLE VI
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 $\approx 70\%$ of the total execution time (Table VI). 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.

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`, yielding an $8.3\times$ speedup from asynchronous I/O alone (Table VII). 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

TABLE VII
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 $\pm 20\%$	–	0.96
DeepVis (Full)		15,789	0.96

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 domain-specific risk parameters for feature fusion) confirm that F1-Score remains stable at 0.96, validating the robustness of the configuration across deployments.

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], [19] and runtime monitors [3], [4] track information flow and system calls. Recent advancements leverage eBPF to reduce monitoring overhead and harden isolation [20], [21]. 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) [22]. These frameworks utilize cryptographic Merkle trees [23] 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 [24], while complex policy management often leaves the system vulnerable to subversion or TOCCTOU (Time-of-Check-to-Time-of-Use) attacks [25]. 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 [26]–[28] demonstrate that mapping binary files to grayscale images or space-filling curves reveals structural patterns distinct to malware families. Similarly, entropy analysis [29] 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 [30], [31]. 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 [32] use LSTM networks to model system logs, while Kitsune [33] employs autoencoders for network intrusion detection. For high-dimensional tabular or sensor data, unsupervised frameworks such as Deep One-Class Classification [34], GANomaly [35], and DAGMM [36] learn normal data distributions to flag outliers. Additionally, VAE-based models [37], [38] 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.

Why Deep Learning over Classical Methods? While classical approaches such as One-Class SVMs or threshold-based heuristics offer computational simplicity, they fail to model the non-linear manifold of colliding multi-modal features. In preliminary experiments, SVMs exhibited a 14% recall degradation under high saturation (>500 collisions/pixel) because linear decision boundaries cannot disentangle the max-pooled features of benign files from a malicious signal. The CAE architecture learns to suppress the background noise of benign feature collisions through non-linear channel interactions, representing a structural advantage over linear classifiers. Furthermore, while end-to-end I/O remains $O(N)$ due to physical constraints, our contribution isolates the *verification latency* to $O(1)$ via spatial hashing. This ensures that the detection phase does not become a bottleneck as file counts scale to millions.

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-samhain.de/samhain/>, 2003.
- [10] V. Ramos Mello, “Diamorphine: LKM rootkit for Linux,” <https://github.com/mOnad/Diamorphine>, 2023.
- [11] f0rb1dd3n, “Reptile: LKM Linux rootkit,” <https://github.com/f0rb1dd3n/Reptile>, 2023.
- [12] 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.
- [13] R. Amizudin, “Malware sample repository,” <https://github.com/RamadhanAmizudin/malware>, 2024.
- [14] V. M. Alvarez, “YARA: The Pattern Matching Swiss Knife for Malware Researchers,” <https://github.com/VirusTotal/yara>, 2013.

- [15] J. Kornblum, "Identifying Almost Identical Files Using Context Triggered Piecewise Hashing," *Digital Investigation*, vol. 3, no. 3, pp. 91–97, 2006.
- [16] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *AAAI Workshop on Artificial Intelligence for Cyber Security*, 2018.
- [17] L. Liu, B.-S. Wang, B. Yu, and Q.-X. Zhong, "Automatic malware classification and new malware detection using machine learning," *Frontiers of Information Technology & Electronic Engineering*, vol. 18, pp. 1336–1347, 2017.
- [18] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.
- [19] 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.
- [20] Y. He *et al.*, "Cross Container Attacks: The Bewildered eBPF on Clouds," in *Proceedings of the 32nd USENIX Security Symposium*, 2023.
- [21] TFJMP Contributors, "Hardware-assisted Defense-in-depth for eBPF Kernel Extensions," in *Proceedings of the 2024 CCS Cloud Computing Security Workshop (CCSW)*, 2024.
- [22] 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.
- [23] 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.
- [24] 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.
- [25] F. Böhling, 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.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] 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.
- [30] 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.
- [31] R. Uetz *et al.*, "Detecting Evasions of SIEM Rules in Enterprise Networks," in *Proceedings of the 33rd USENIX Security Symposium*, 2024, pp. 1–18.
- [32] 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.
- [33] 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.
- [34] L. Ruff *et al.*, "Deep One-Class Classification," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018, pp. 4390–4399.
- [35] 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.
- [36] 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.
- [37] 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.
- [38] 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.