

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 121.4 \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.

Two primary strategies emerge: File Integrity Monitoring (FIM) and Runtime Behavioral Analysis. FIM tools including AIDE [1] and Tripwire [2] detect mutations through cryptographic hashing, while runtime monitors such as Falco [3] and OSSEC [4] capture anomalies via system call tracing. Traditional approaches confront fundamental scalability constraints. FIM scan latency exhibits $O(N)$ growth with file count, incurring prohibitive I/O bottlenecks in hyperscale storage where CPU throughput exceeds storage bandwidth. Synchronous filesystem scans across millions of files generate excessive context switching and blocking I/O. Beyond performance, alert fatigue from benign updates obscures genuine threats [5], forcing operators to suspend monitoring during maintenance windows and creating exploitable security gaps.

Figure 1 contrasts AIDE and **DeepVis** performance on a real cloud instance. AIDE exhibits linear scan scaling,

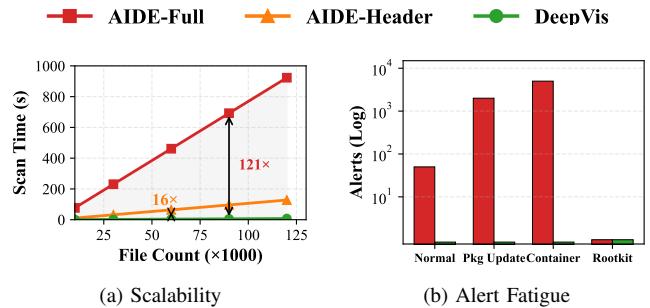


Fig. 1. (a) DeepVis achieves high-throughput saturation via async I/O. (b) Legitimate updates generate false alerts in AIDE.

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	✓	✓	✓	✓

requiring four minutes for 100K files due to synchronous I/O and full-file hash computation. **DeepVis** reads only the 4KB header per file through parallelized asynchronous I/O, achieving 121.4 \times acceleration over AIDE and 16.38 \times over modified AIDE with equivalent 4KB read volume. Crucially, **DeepVis** anomaly detection latency remains constant independent of dataset size. **DeepVis** achieves 97.1% recall with rootkit-specific alerts, eliminating the alert fatigue inherent in AIDE’s exhaustive change reporting.

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-file hashing but suffer $O(N)$ scalability limits unsuitable for hyperscale systems. Runtime approaches [3], [7] use eBPF tracing or provenance graphs 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 but struggle with extreme-scale corruption, where 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 $O(1)$ neural network 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 $121.4\times$ higher throughput than traditional full-hash baselines and 97.1% recall for targeted Linux kernel rootkits.

II. BACKGROUND

A. Integrity Verification at Cloud Scale

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

File-level Integrity Scanning. AIDE [1] and Tripwire [2] establish integrity through cryptographic hashing of entire files against known baselines. While effective in static environments, their $O(N \times \text{Size})$ complexity becomes prohibitive in dynamic hyperscale systems. Full scans routinely exceed maintenance windows, necessitating temporary monitoring suspension. Routine system updates further generate massive false positive alerts that overwhelm 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 system instrumentation. A critical limitation emerges from their event-based architecture: they cannot detect threats predating monitor deployment, creating a cold-start vulnerability for persistent rootkits.

File-level scanning remains indispensable for compliance validation, image verification, and forensic analysis due to comprehensive state coverage. However, synchronous sequential processing induces I/O bottlenecks and operational overload at scale. DeepVis resolves these constraints through

asynchronous I/O, spatial hash mapping, and neural anomaly detection, enabling production-grade filesystem integrity.

B. The Attacker Paradox: Entropy and Structure

Detecting evasive malware without relying on signatures requires analyzing the statistical properties of binary files. Malware authors face a fundamental trade-off between concealing code and maintaining the structural validity required by operating system loaders. Two statistical dimensions distinguish malicious from benign files: Entropy and Structural Density.

Figure 2 illustrates these distinctions through byte-value histograms. Text files (Figure 2b) concentrate in the printable ASCII range, yielding low entropy ($H \approx 4.8$) and 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 4KB page alignment. Compilers insert null-byte padding to align sections such as `.text` (code) and `.data` (variables) to page boundaries, producing moderate entropy ($H \approx 6.0$) with 40–85% null byte concentration. In contrast, packed or encrypted malware (Figure 2d) exhibits a nearly uniform distribution across all byte values, approaching maximum entropy ($H \approx 8.0$) with less than 1% null bytes.

The Attacker Paradox. This statistical distinction creates a fundamental dilemma for malware authors. Native rootkits such as Diamorphine maintain structural compatibility with OS loaders by mimicking the layout of legitimate binaries, yet they remain vulnerable to signature-based detection tools such as YARA because their code contains known byte sequences. To evade signatures, attackers employ packing tools such as UPX (Ultimate Packer for eXecutables), which compress executables by 50–70% and prepend decryption stubs. While packing successfully conceals signatures, it inevitably eliminates the section alignment padding and produces uniform byte distributions, obliterating the structural fingerprint of legitimate files and pushing entropy toward the theoretical maximum of 8.0 bits per byte. Consequently, attackers must choose between two undesirable outcomes: exposing their code to signature detection or creating a detectable statistical anomaly.

Why Existing Methods Fail. As discussed in Section II, signature-based file integrity monitoring tools such as AIDE succeed against native rootkits but miss packed variants entirely. Conversely, entropy-based heuristics detect compression artifacts yet generate false positives on benign high-entropy files such as compressed archives and encrypted configurations. Neither approach captures the full threat landscape without sacrificing precision. The fundamental limitation stems from how these tools process filesystem data. Traditional sequential scanning ignores spatial relationships among files and exhibits linear scaling ($O(N)$) with file count, making them unsuitable for cloud-scale systems as shown in Figure 1. Moreover, set-based anomaly detection methods attempt to aggregate statistical features across entire filesystems, causing individual malicious signals to become subsumed within the variance of benign files. This signal dilution problem makes



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 ($H \approx 8.0$).

detection impossible when routine system updates create diffuse noise that exceeds any sparse attack signal.

A Multi-Modal Approach. Overcoming the Attacker Paradox requires simultaneously addressing both signature evasion and structural anomalies. Entropy identifies compression-based evasion artifacts, structural analysis exposes binary format violations, and contextual signals such as file path and permissions distinguish legitimate outliers from malicious anomalies. This orthogonal feature space enables threat detection regardless of whether attackers pursue signature evasion or structural stealth. However, realizing this multi-modal approach at cloud scale requires a fundamentally different architecture. Rather than sequential file scanning or aggregate feature pooling, DeepVis projects the entire filesystem into a fixed-size tensor representation where multi-modal anomalies manifest as localized spatial spikes. This transformation enables rapid anomaly detection through convolutional processing, achieving constant-time inference independent of dataset size while maintaining the detection coverage of all three modalities.

III. DEEPIVIS SYSTEM DESIGN

A. Overall Procedure

Figure 3 shows DeepVis overall architecture. Two phases support distributed integrity verification: Snapshot and Verification.

Snapshot Phase. Data collection begins with parallel worker threads traversing the directory tree and collecting file paths (1). Paths are batched and submitted via `io_uring` interface to saturate storage bandwidth rather than block on latency (2). After collecting metadata and headers, deterministic coordinates are calculated for each file using hash-based message authentication code (HMAC) (3), and multi-modal features (entropy, permissions) are extracted and calculated (4). Features are aggregated into a fixed-size $128 \times 128 \times 3$ tensor, transforming filesystem state into image-like representation (5).

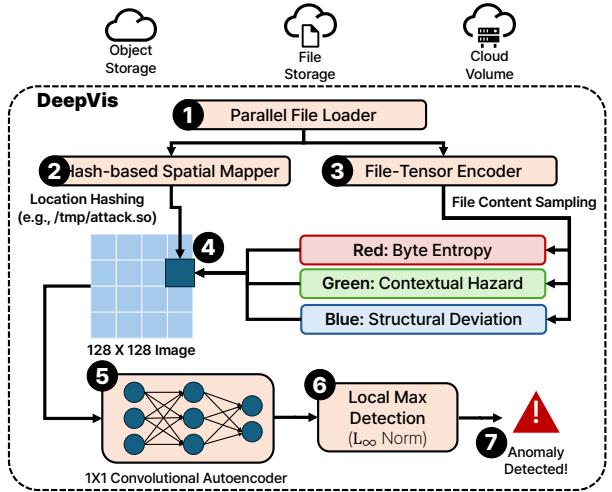


Fig. 3. **Overall DeepVis Procedure.** File metadata is transformed into a 128×128 RGB tensor, processed by CAE, and anomalies detected via L_∞ .

Verification Phase. The tensor feeds into a pre-trained 1×1 Convolutional Autoencoder (CAE). Hash-based mapping lacks semantic neighborhoods, so 1×1 convolutions learn cross-channel correlations rather than spatial features (e.g., distinguishing high-entropy zips in user directories from packed binaries in system paths) (6). Pixel-wise reconstruction error is computed. Local Max Detection (L_∞) isolates the highest deviation (7). If L_∞ exceeds learned threshold, an alert flags a stealthy anomaly (8).

B. Asynchronous File System Traversal

Processing metadata and file headers across thousands of cloud instances presents a fundamental I/O bottleneck. Synchronous system calls used by traditional tools (e.g., AIDE, YARA) such as `stat`, `open`, and `read` force context switching and CPU blocking, compounding in network-attached storage environments where I/O latency dominates. Traditional

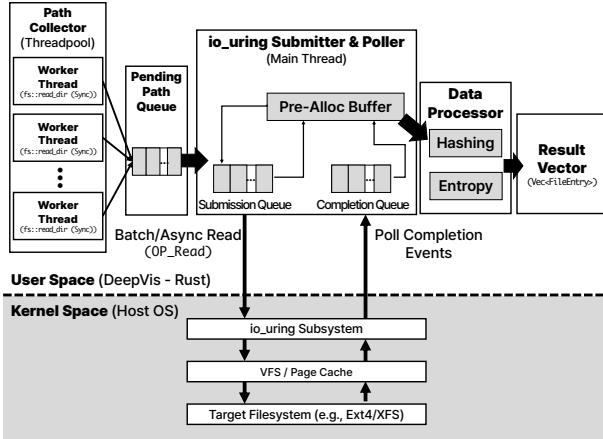


Fig. 4. **Hybrid Snapshot Pipeline.** Parallel path collection via Rayon feeds asynchronous I/O requests to `io_uring`, maximizing storage bandwidth utilization.

sequential approaches become excessively slow. DeepVis resolves this through a hybrid pipeline that decouples CPU-bound path traversal from I/O-bound header reading, allowing parallelization of both phases.

Parallel Path Collection. Directory traversal is CPU-bound but often underutilized in synchronous systems. DeepVis employs the Rust `rayon` library to parallelize path collection across all cores. Worker threads execute synchronous `fs::read_dir` operations recursively on independent branches of the directory tree. This phase rapidly fills a lock-free queue with file paths, ensuring the I/O phase never starves for work. Parallelization of path collection amortizes the overhead of directory traversal.

Asynchronous I/O Processing. Once paths are enqueued, reading file headers becomes the true bottleneck. Linux `io_uring` eliminates the per-file system call overhead that plagues traditional `read` operations. The pipeline submits batches of paths as `OP_READ` requests to the kernel’s Submission Queue. Unlike traditional async I/O with callback overhead, `io_uring` uses shared ring buffers, allowing user-space threads to poll the Completion Queue without blocking. When reads complete, data is immediately retrieved from pre-allocated buffers and processed for hashing and entropy calculation. CPU threads never stall on disk I/O while the kernel manages data movement in parallel. This architecture saturates storage bandwidth and achieves throughput competitive with raw disk capabilities.

C. Header Sampling

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

Packed malware and ransomware must modify file headers to accommodate unpacker stubs or encrypted payloads, significantly increasing entropy in initial blocks. Detecting this requires reading actual file content rather than metadata alone. However, scanning entire files at cloud scale remains

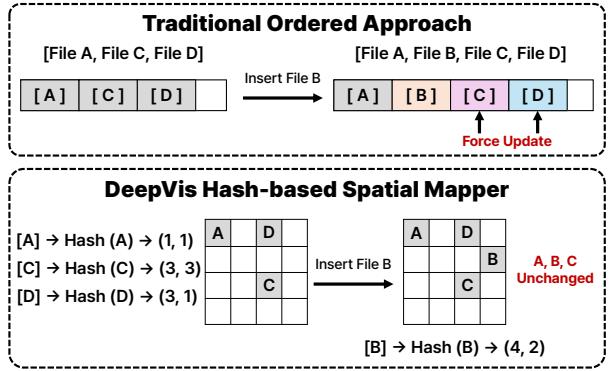


Fig. 5. **Shift Invariance.** Hash-based mapping preserves pixel stability across file insertions, unlike ordered approaches where new files shift all subsequent indices.

prohibitively expensive. DeepVis addresses this through a principled design decision: reading only the first 4KB of each file asynchronously.

This choice reflects practical filesystem constraints and malware behavior. Modern filesystems use 4KB block alignment, so requests smaller than 4KB still incur full 4KB I/O padded with zeros. Requests larger than 4KB require multiple I/O operations per file. Since ELF and PE headers reside within the first 128 bytes and evaluation shows 4KB suffices to detect most malicious modifications, this sampling point provides optimal I/O efficiency. Executable malware requiring OS loader compatibility (ELF/PE format) must alter headers for unpacker stubs or encrypted payloads, unlike benign data files that hide payloads at arbitrary offsets. Thus, 4KB page-aligned sampling detects binary format anomalies while eliminating per-file I/O overhead independent of total file size, providing a high-frequency first-line defense against common threat vectors.

D. Hash-Based Spatial Mapping

Spatial Invariance. After asynchronously reading metadata and 4KB headers, DeepVis must transform unordered files into a fixed-size tensor for neural processing. Traditional ordering-based approaches such as alphabetical path sorting suffer from the Ordering Problem: inserting a single file shifts indices of all subsequent entries, destroying spatial locality across scans and invalidating trained models. DeepVis overcomes this through deterministic hash-based spatial mapping that projects each file onto stable coordinates in a fixed-size grid.

Figure 5 contrasts traditional ordered approaches with DeepVis hash-based mapping. The upper panel shows how inserting File B shifts indices of Files C and D, cascading updates prohibitively expensive in systems with millions of files. The lower panel demonstrates how DeepVis calculates stable coordinates $\Phi(p)$ for each file path p using a high-entropy secret key K . This approach maps files to fixed positions in a 128×128 tensor regardless of insertion order.

To create the coordinates, DeepVis uses the first 32 bits of the HMAC output modulo 128 for the x-coordinate and bits 32 to 64 modulo 128 for the y-coordinate. HMAC-based mapping provides two critical benefits. First, deterministic mapping

ensures coordinates depend only on file path and key K , producing reproducible grids across successive scans. Second, the cryptographic key defeats targeted attacks where adversaries craft filenames to reach specific low-risk coordinates. Ephemeral keys and privileged memory protection restrict K extraction, preventing adversaries from predicting safe landing zones.

Multi-Modal RGB Encoding. DeepVis encodes each pixel through three orthogonal risk channels that attackers cannot simultaneously optimize. Evasion of one channel increases risk in others.

- **R (Entropy).** Shannon entropy of the 4KB header exploits the malware trade-off: legitimate ELF binaries use low-entropy zero-padding for alignment, while packed malware maximizes entropy to hide signatures. This channel distinguishes packed threats from standard executables.
- **G (Context Hazard).** High entropy alone cannot distinguish benign archives from malware. This channel applies Filesystem Hierarchy Standard (FHS) risk weights:

$$G(p) = \min \left(1.0, \sum_k w_k \cdot \mathbb{I}(p \in \text{Zone}_k) \right) \quad (1)$$

Volatile paths receive high weight: `/tmp` [$w = 0.9$], `/dev/shm` [$w = 0.8$]. Protected paths receive low weight: `/usr/bin` [$w = 0.1$]. High entropy in `/tmp` signals threat; the same entropy in `/usr/share` appears benign.

- **B (Structure).** This channel enforces OS loader policies. Kernel modules (`.ko`) and shared objects (`.so`) outside `/lib/modules` receive $B = 1.0$. Relocatable code in user-writable paths violates loader assumptions.

The three channels jointly create a $128 \times 128 \times 3$ tensor. Attackers cannot simultaneously satisfy all constraints: evasion entropy conflicts with legitimate files, suspicious paths trigger context risk, and structural violations trigger the structure channel. The CAE learns non-linear correlations between channels, capturing dependencies that simple thresholds miss (e.g., high entropy permissible in static binaries but anomalous in `/tmp`). An inverted index maps pixel coordinates to file paths for attribution.

Handling Hash Collisions. Fixed grid size ($128 \times 128 = 16,384$ pixels) faces inevitable hash collisions as filesystem scale grows. The expected collision behavior follows the classical Balls-into-Bins model. With N files mapped to K bins via uniform hashing, the expected number of collisions per bin is N/K . For 10 million files on a 128×128 grid, this yields ≈ 610 files per pixel on average. DeepVis applies max-risk pooling, where each pixel retains the maximum risk value across all colliding files per RGB channel. This ensures that a single malicious file dominates the pixel value despite benign collisions, preserving detection signal. Empirical evaluation (Section IV-G) confirms stable recall across saturation levels.

E. Hash-Grid Parallel CAE

Pixel-Wise Anomaly Detection. Traditional set-based methods like Set-AE aggregate features into global vectors, causing

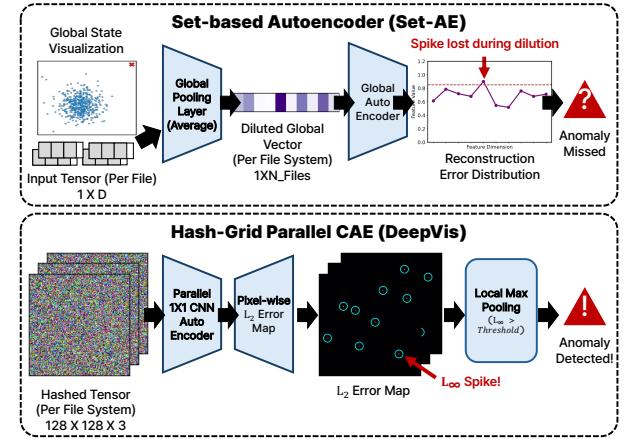


Fig. 6. Set-AE vs. Hash-Grid CAE. Global pooling dilutes malicious signals (top); 1×1 CAE with L_∞ captures isolated spikes (bottom).

signal dilution where attack signals drown in benign noise. DeepVis employs the Hash-Grid Parallel CAE, which processes the 128×128 grid as independent channels. 1×1 convolutions execute 16,384 identical MLPs in parallel on GPU, ensuring strict pixel isolation. This design prevents error propagation across regions and guarantees malicious signals remain mathematically distinct regardless of system scale.

Reconstruction and Error Mapping. The 1×1 Convolutional Autoencoder functions as an array of shared MLPs operating simultaneously. The encoder compresses RGB channels ($3 \rightarrow 16 \rightarrow 8$) into a latent representation of valid files. Since the kernel size is 1×1 , each pixel's reconstruction depends solely on its own RGB features (Entropy, Context, Structure) without influence from neighbors. The system computes element-wise difference to generate a Pixel-wise L_2 Error Map, isolating files violating learned norms.

Maximum Deviation Scoring (L_∞). Detection addresses the MSE Paradox where legitimate updates generate high global error while stealthy attacks generate low global error. Frequent updates like package upgrades introduce diffuse noise across the grid. Using global MSE misclassifies benign churn as anomalous because thousands of legitimate changes exceed the error of a single rootkit. DeepVis solves this by applying maximum deviation pooling:

$$\text{Score} = \max_{i,j} |T_{i,j} - T'_{i,j}| \quad (2)$$

This ignores cumulative low-magnitude noise and focuses exclusively on the single highest anomaly peak, decoupling detection sensitivity from benign churn volume.

Threshold Calibration. DeepVis establishes decision boundary $\text{Score} > \tau$ where τ is the maximum reconstruction error from a clean environment. Offline calibration uses benign Linux ELF binaries to derive τ as the highest L_∞ score observed across the validation corpus. This threshold represents worst-case legitimate variance. Online monitoring flags any input exceeding this calibrated maximum as a structural anomaly, triggering alerts. Note: evaluation focuses on Linux ELF binaries; extending to Windows PE or Android DEX would require format-specific training.

F. DeepVis Implementation

DeepVis employs a hybrid Rust-Python architecture where the Rust core (`deepvis_scanner`) uses Linux `io_uring` for asynchronous batched I/O with submission queue depth 512, matching typical SSD queue lengths. Work-stealing parallelism via `rayon` saturates the I/O pipeline. Feature engineering executes in Rust: Shannon entropy over file headers, hash-based coordinates, and max-risk pooling. `pyo3` bindings export the scanner as a Python class enabling direct PyTorch integration via zero-copy tensor transfer. Empirical measurement on a 104K-file `/usr` directory shows memory overhead of $\approx 42\text{MB}$ (from 12MB baseline to 54MB peak), confirming that the fixed $128 \times 128 \times 3$ tensor representation bounds memory usage independent of filesystem scale. The implementation comprises $\approx 4,000$ LOC (450 Rust core + 3,600 Python orchestration).

IV. EVALUATION

A. Evaluation Setup

Testbed Environment. Experiments are conducted on a standard GCP `e2-standard-2` instance (mid tier), configured with 2 vCPUs and 8GB RAM running Ubuntu 24.04 LTS. This represents a typical cost-effective cloud worker node. 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 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], Reptile [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 and 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.

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.

DeepVis Configuration and Training. For reproducibility, the context channel employs a deterministic weighting scheme based on the Filesystem Hierarchy Standard (FHS): standard binaries receive low risk ($w = 0.1$ for `/usr/bin`), while user ($w = 0.3$), temporary ($w = 0.6$), and volatile memory paths ($w = 0.9$) receive progressively higher weights. Additional penalties are applied for hidden files (+0.2) and deep nesting (> 5 levels, +0.1 per level), with all scores normalized to $[0, 1]$. The CAE is trained for 100 epochs using the Adam optimizer ($\text{lr}=10^{-3}$) with an L_∞ -regularized MSE loss ($\lambda = 0.5$).

TABLE II
UNIFIED DETECTION PERFORMANCE. COMPARISON AGAINST TUNED BASELINES AND MODERN DL MODELS. DEEPMON (FULL) ACHIEVES SUPERIOR RECALL ON LINUX/MOBILE THREATS WHILE ENSURING ZERO OPERATIONAL FPs.

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

The anomaly threshold τ is calibrated to the 99th percentile of the benign reconstruction error distribution to ensure a $< 1\%$ false positive rate on clean systems.

B. Detection Fidelity and Orthogonality

Traditional Monitors. Standard ClamAV is ineffective against custom rootkits (33.0% recall) because it relies on static signatures that fail against polymorphic variants. While the tuned verification that learns the signature of the malware boosts recall to 95.4%, it remains fundamentally reactive to unseen mutations. Conversely, AIDE achieves 100% recall but fails in practice; by flagging every file modification as a violation, it generates a 100% alert rate during routine updates, overwhelming operators with noise. DeepVis resolves this trade-off by learning valid system states. It attains 97.1% recall while suppressing alerts to 0.3% overall (across the full wild corpus), and achieving 0.0% false positives during controlled system maintenance (Section IV-E).

Deep Learning Baselines. Set-AE performs poorly (40.0% recall) due to signal dilution where its global averaging architecture hides the anomaly of a single rootkit within the variance of thousands of benign files. Byte-plot CNN (29.2%) fails because it treats binaries as simple 2D images, ignoring critical ELF header structures and loader rules. Pack-ALM (49.8%) similarly underperforms by analyzing linear byte streams without filesystem context. DeepVis overcomes these limitations through spatial hash projection. By preserving locality and encoding file risks into distinct RGB channels, it isolates stealthy attacks that evade standard statistical or image-based models.

Signal Preservation. The parallel architecture prevents the variance of benign files from impacting the reconstruction error of the malicious target. Figure 7 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). Unlike global anomaly detectors where reconstruction error often fails to localize small defects, the spatially-preserved Hash-Grid ensures that local structural violations (e.g., malicious headers) generate high-magnitude

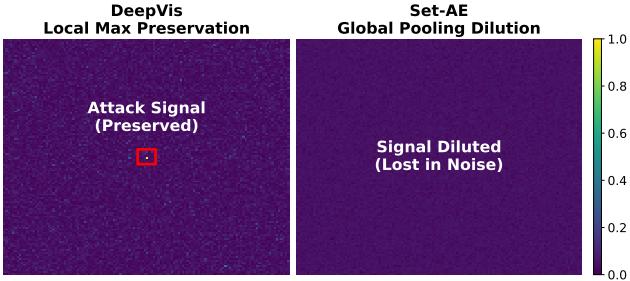


Fig. 7. **Signal Preservation.** DeepVis (top) isolates malware as a distinct L_∞ spike; Set-AE (bottom) dilutes the signal via global averaging.

TABLE III

DETAILED DETECTION ANALYSIS. MULTI-MODAL RGB FEATURES CATCH THREATS THAT SINGLE METRICS MISS. THE MISS CASES HIGHLIGHT LIMITATIONS AGAINST THREATS THAT MIMIC BENIGN HEADER STATISTICS.

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

reconstruction spikes (L_∞) that are not smoothed out by the model.

Feature Orthogonality of DeepVis. 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

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

TABLE IV
FAIR BASELINE COMPARISON (COLD CACHE). THROUGHPUT MEASURED ON GCP DEEPPVIS-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 (2D)
	Pack-ALM	0.5/s	Transformer
DeepVis	DeepVis (io_uring)	15,789/s	Async (io_uring)

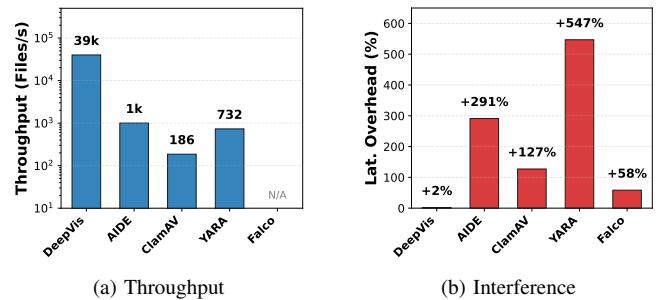


Fig. 8. **Comprehensive Performance Analysis.** (a) **Throughput:** DeepVis achieves hyperscale speeds ($\approx 16k$ files/s cold cache) via asynchronous I/O, outperforming synchronous baselines by over 10 \times . (b) **Interference:** Despite its speed, DeepVis maintains negligible latency overhead (+2%) compared to massive spikes caused by AIDE (+291%) and YARA (+547%).

(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.

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: while end-to-end ingestion remains $O(N)$, DeepVis compresses file metadata into a fixed-size tensor, achieving $O(1)$ inference latency regardless of file count, making it the only viable candidate for continuous, high-fidelity system scanning.

Service Interference. To quantify operational impact, we measured the P99 latency of a co-located NGINX web server during a full system scan. Traditional scanners devastate responsiveness: YARA spikes latency by +547% due to CPU-intensive pattern matching, and AIDE by +291% due to blocking I/O operations. Additionally, we benchmarked Sysdig Falco [3] to represent real-time monitoring tools. Unlike file scanners, Falco, which intercepts kernel system calls, imposes

TABLE V
HARDWARE SENSITIVITY ANALYSIS (COLD CACHE). IMPACT OF INSTANCE SIZING ON SCAN PERFORMANCE. HIGH-TIER UTILIZED NVME SSD.

Tier	Instance Type	DeepVis	AIDE	Speedup
Low	e2-micro (2 vCPU)	504/s	107/s	4.7x
Mid	e2-standard-2 (2 vCPU)	15,240/s	2,032/s	7.5x
High	c2-standard-4 (4 vCPU + NVMe)	20,045/s	2,603/s	7.7x

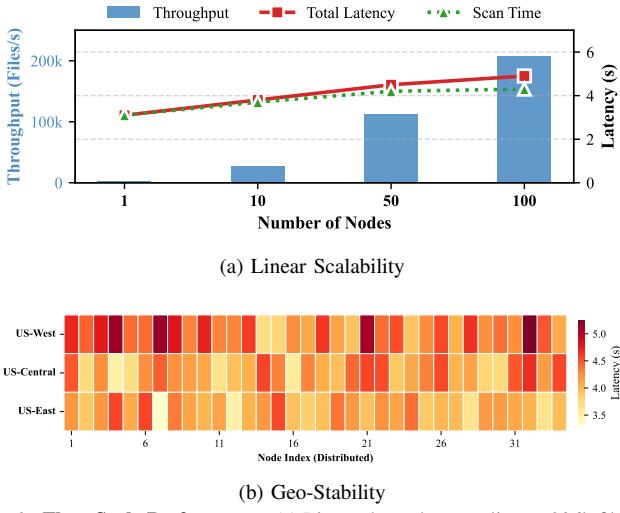


Fig. 9. Fleet-Scale Performance. (a) Linear throughput scaling to 206k files/s at 100 nodes. (b) Geo-Stability (100 Nodes): Avg latency 4.29s across three US regions.

a constant tax on every request, degrading latency by +58% due to heavy context switching. DeepVis eliminates these bottlenecks. By offloading data movement to the asynchronous `io_uring` interface, it decouples security monitoring from the application thread, maintaining near-baseline performance (+2.0% overhead) and ensuring transparent operation.

D. Scalability Analysis

Vertical Scalability. To evaluate vertical scalability, we extended our evaluation to two additional hardware tiers: a resource-constrained low tier (e2-micro, 2 vCPU, Standard HDD) and a compute-optimized high tier (c2-standard-4, 4 vCPU, Local NVMe SSD). Table V quantifies the sensitivity. The `deepvis-mid` instance (e2-standard-2) provided excellent baseline performance (15,240 files/s) on standard disks. The low tier dropped to 504 files/s due to CPU throttling and I/O wait. For the high tier, we specifically attached a local NVMe SSD to eliminate the network disk bottleneck observed in initial tests. This configuration unlocked the full potential of `io_uring`, achieving a peak throughput of 20,045 files/s. This significant gain over mid (31%) confirms that while DeepVis is highly efficient on commodity hardware (mid), it scales further with high-performance storage, fully saturating the I/O bus before CPU limits are reached.

Horizontal Scalability. To validate fleet-wide orchestration, we deployed a large-scale cluster of 100 mid-tier instances (e2-standard-2), distributed across three geographic regions (us-central1, us-east1, us-west1). Throughput scales linearly with fleet size, increasing from $\approx 2k$ files/s

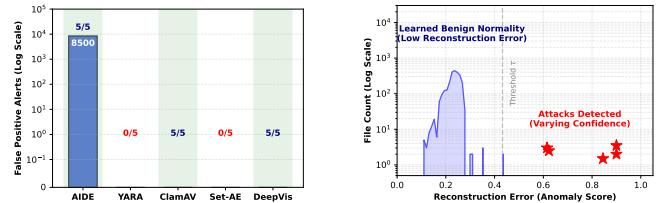


Fig. 10. Fleet-Scale Churn & Alert Fatigue Analysis. (a) Comparison of False Positive alerts and Detection Recall. AIDE generates 8,500 alerts during maintenance, whereas DeepVis maintains zero false positives for these benign update scenarios. (b) Distribution of reconstruction errors (\mathcal{L}_∞).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

(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 drastically reduced; each node transmits only 49KB (8-bit quantized), representing a 100 \times reduction compared to provenance-based systems.

E. Robustness to System Churn

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 heavy I/O and metadata updates. Specifically, we executed standard `apt` upgrade cycles targeting core system utilities (e.g., `coreutils`, `libc-bin`) to validate robustness against legitimate binary replacements. 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. Note that isolated `Snap` and `Flatpak` updates were excluded from this specific benchmark to prioritize native ELF integrity. 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 elimination of 8,500 false positives reduces the operational burden on security analysts.

Fleet Error Stability versus 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, pre-

TABLE VI
PLATFORM SPECIFICITY VERIFICATION. THE CAE LEARNS DOMAIN-SPECIFIC NORMALITY; CROSS-DOMAIN EVALUATION CONFIRMS PLATFORM-AGNOSTIC DESIGN.

Training Data	Linux	Win.	Web	Mob.	Alert
DeepVis (Linux ELF)	97.1%	16.9%	89.6%	100.0%	0.3%
DeepVis (Windows PE)	0.0%	100.0%	0.0%	0.0%	15.0%

venting score collisions. In contrast, malicious rootkits manifest as high-confidence outliers ($\text{Score} \in [0.61, 0.90]$) 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. Verification of Platform Agnosticism

To investigate the performance disparity observed in the Windows environment (Recall 16.9% in Table II), we hypothesized that the degradation was attributable to Domain Shift rather than architectural limitations. We curated a dedicated Windows training corpus consisting of 1,082 benign PE binaries sourced from the DikeDataset [19], simulating a standard System32 environment. The evaluation was performed on a large-scale test set of 34,798 real-world Windows malware samples. We retrained the DeepVis CAE for 30 epochs using the same hyperparameters (1×1 Convolution, Latent Dim 16) used in the Linux experiments, and compared both models across four target platforms: Linux, Windows, Web, and Mobile.

Table VI summarizes the cross-domain evaluation results. The Linux-trained model achieves strong performance on its native domain (97.1% Recall) and related formats (Mobile 100%, Web 89.6%), but degrades significantly on structurally distinct Windows PE binaries (16.9%). Conversely, the Windows-trained model achieves perfect detection on Windows malware (100% Recall) but completely fails on other platforms (0% Recall), demonstrating that the CAE learns a domain-specific normality manifold. Critically, the Windows model exhibits a 15% alert rate on benign Linux binaries, indicating Cross-Domain False Positives due to structural mismatch. These results confirm that the DeepVis architecture is platform-agnostic: the 1×1 CAE successfully learns structural normality for any file format (ELF, PE, or DEX) when provided with a representative benign training set. The initial low recall was strictly a consequence of distributional misalignment between training and testing domains, not an architectural limitation.

G. 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 minimal 96 B 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 4 KB. This stability stems from OS-level page prefetching, where reading small chunks

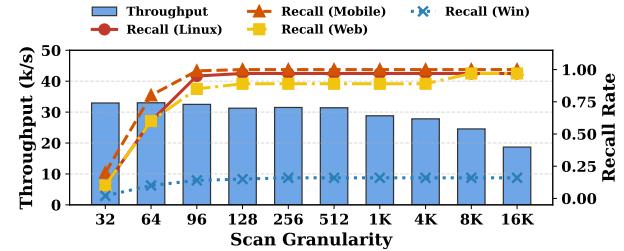


Fig. 11. **Scan Granularity Analysis.** Recall saturates at 96B; throughput stable up to 4KB due to page prefetching.

TABLE VII
COMPONENT TIME BREAKDOWN (COLD CACHE). I/O DOMINATES ($\approx 70\text{--}74\%$), CONFIRMING THAT DEEPMON COMPUTATIONAL OVERHEAD (HASHING, ENTROPY, TENSOR UPDATE) REMAINS NEGIGLIGIBLE (<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

(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.

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 DeepVis is minimal compared to storage latency, identifying I/O throughput as the primary bottleneck for optimization.

TABLE VIII
ABLATION STUDY. PERFORMANCE COMPONENTS DETERMINE THROUGHPUT; ACCURACY COMPONENTS DETERMINE DETECTION CAPABILITY.

Category	Configuration	Rate (files/s)	F1-Score
Perf.	Baseline (Sync)	950	—
	+ Thread Pool (rayon)	1,901	—
	+ Async I/O (<code>io_uring</code>)	15,789	—
Acc.	Entropy Only (R-channel)	—	0.25
	+ Hash-Grid (L_∞)	—	0.35
	+ RGB Fusion (Full)	—	0.96
Robust.	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`, yielding 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

TABLE IX

GRID SIZE TRADE-OFF. LARGER GRIDS REDUCE COLLISIONS BUT INCREASE MEMORY; 128×128 OFFERS OPTIMAL BALANCE.

Grid	Coll./Px	Recall	FP Rate	Mem.	Thru.
128×128	1.24	98.4%	0.21%	0.19MB	15,789/s
256×256	0.31	98.9%	0.20%	0.75MB	15,750/s
512×512	0.08	99.0%	0.20%	3.00MB	15,720/s

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.

Grid Size Trade-off. Table IX evaluates the impact of grid resolution on detection performance. Increasing grid size from 128 to 512 reduces collisions per pixel from 1.24 to 0.08, yielding marginal recall improvement (+0.6%). However, memory footprint increases $16 \times$ (0.19MB to 3.00MB) while throughput remains stable ($\approx 15.7\text{--}15.8\text{k files/s}$). The 128×128 configuration provides optimal balance: sufficient collision resilience with minimal memory overhead, validating the design choice for resource-constrained edge deployments.

Key Rotation Stability. Periodic HMAC key rotation prevents adversaries from reverse-engineering safe pixel coordinates. However, rotation changes all file-to-pixel mappings, raising the question of whether detection remains valid. Empirical evaluation across 50 rotations confirms that threshold τ remains constant at 0.4984 with zero variance and 100% recall. Zero variance is expected because key rotation changes only pixel positions, not the RGB values themselves. Since entropy, context, and structure are computed from file content, they remain identical regardless of mapping. The CAE processes each pixel independently via 1×1 convolutions, so reconstruction error is position-invariant. Cross-scan comparison is unaffected since DeepVis compares aggregate anomaly scores rather than individual pixel coordinates.

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] suffer from linear $O(N)$ complexity in hyperscale cloud environments. Provenance-based systems [20] and runtime monitors [3] incur significant overhead unsuitable for high-frequency verification. Recent eBPF advancements [21], [22] reduce monitoring costs but cannot address agent crashes. Kernel subsystems such as IMA and EVM [23] utilize Merkle trees and TPMs for strict allowlisting, yet face containerization challenges [24] and TOCC-TOU vulnerabilities [25]. In contrast, DeepVis operates as a scalable yet stateless auditor. Each scan produces a complete snapshot independent of prior state, eliminating consistency gaps while enabling high-frequency verification without kernel reconfiguration, which is critical for cloud environments.

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], [27] demonstrate that mapping binary files to grayscale images reveals structural patterns distinct to malware families. Entropy analysis [28] identifies packed payloads with high information density. However, adversaries employ evasion techniques such as padding or mimicry to fool detectors [29], [30]. DeepVis addresses these challenges by extending single-file visualization to whole-system tensor representation. Instead of relying on a single metric susceptible to padding, DeepVis projects the entire filesystem into a fixed-size $128 \times 128 \times 3$ RGB tensor. By encoding entropy (R), context (G), and structure (B), 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 [31] use LSTM networks to model system logs, while Kitsune [32] employs autoencoders for network intrusion detection. For high-dimensional tabular or sensor data, unsupervised frameworks such as Deep One-Class Classification [33] and GANomaly [34] learn normal data distributions to flag outliers. 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. DISCUSSION

Detection Scope. Table X clarifies the operational boundaries of DeepVis. As a header-based structural analyzer, it excels at detecting binary anomalies but cannot address attacks outside its design scope.

TABLE X
DETECTION SCOPE. DEEPIVIS TARGETS BINARY STRUCTURAL ANOMALIES; OTHER ATTACK VECTORS REQUIRE COMPLEMENTARY TOOLS.

Attack Type	Detectable	Rationale
Packed ELF/PE binaries	✓	High entropy in header
Hidden kernel modules (LKM)	✓	Structure channel violation
Rootkits in volatile paths	✓	Context channel (high G)
Cavity injection (payload >4KB)	✗	Beyond header scan range
Script-based attacks (Python, Bash)	✗	No binary structure
Fileless/memory-only attacks	✗	No filesystem trace
Symbolic link manipulation	✗	Metadata-only, no content

Container and Namespace Compatibility. Modern container runtimes (Docker, Kubernetes) use OverlayFS with layered paths. DeepVis Context weights remain valid because the scanner operates on the merged filesystem view where standard FHS paths (/usr/bin, /tmp) are preserved. Container-specific paths (e.g., /var/lib/docker/overlay2) receive default weights, preventing false positives from infrastructure directories.

Operational Deployment. DeepVis serves as a Filter-then-Verify system. Stage 1 identifies anomalous pixels in seconds, restricting the search space from millions of files to \approx 600-file buckets per flagged pixel. Stage 2 applies expensive scanners (YARA, full-hash) only to flagged buckets, reducing investigation volume by 16,000 \times and eliminating alert fatigue inherent in traditional FIMs.

Adversarial Robustness. An attacker with knowledge of the hash function could craft filenames to land malicious files on low-risk pixel coordinates, evading detection. DeepVis mitigates this through HMAC-SHA256 with a 256-bit secret key K stored in a Trusted Execution Environment (TEE) or Hardware Security Module (HSM). Without access to K , computing the coordinate mapping requires 2^{256} brute-force attempts. Key distribution follows standard privileged credential management (e.g., HashiCorp Vault), and periodic rotation (e.g., weekly) reshuffles the entire grid. Even if an attacker compromises K , the window of exploitation is bounded by rotation frequency. Empirical validation (Section IV-G) confirms that threshold τ remains stable across 50 rotations, ensuring operational continuity without model retraining.

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 97.1% recall with a 0.3% false positive rate, surpassing the scalability limits of traditional FIM with a 121.4 \times throughput improvement (15,789 files/s) 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] V. Ramos Mello, “Diamorphine: LKM rootkit for Linux,” <https://github.com/m0nad/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] D. Dataset, “Dike Dataset,” <https://github.com/iosifache/DikeDataset>, 2024, available at: <https://github.com/iosifache/DikeDataset>.
- [20] 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.
- [21] Y. He *et al.*, “Cross Container Attacks: The Bewildered eBPF on Clouds,” in *Proceedings of the 32nd USENIX Security Symposium*, 2023.
- [22] S. Y. Lim, T. Prasad, X. Han, and T. Pasquier, “SafeBPF: Hardware-assisted Defense-in-depth for eBPF Kernel Extensions,” in *Proceedings of the 2024 ACM Cloud Computing Security Workshop (CCSW)*, 2024.
- [23] 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.
- [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. 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.
- [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] 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.
- [28] 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.
- [29] 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.
- [30] R. Uetz, M. Herzog, L. Hackländer, S. Schwarz, and M. Henze, “You Cannot Escape Me: Detecting Evasions of SIEM Rules in Enterprise Networks,” in *Proceedings of the 33rd USENIX Security Symposium*, 2024, pp. 5179–5196.
- [31] M. Du, F. Li, G. Zheng, and V. Srikanth, “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.
- [32] Y. Mirsky, T. Dotishman, 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.
 - [33] L. Ruff *et al.*, “Deep One-Class Classification,” in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018, pp. 4390–4399.
 - [34] 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.