Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

An Autonomous Self-Evolving Malware Detection System for Real-Time Threat Intelligence

Niranjan B
RV College of Engineering
Jasmine K S
jasmineks@rvce.edu.in

RV College of Engineering

Research Article

Keywords: AI Malware Detection, Autonomous Security, Self-Evolving Models, PE Feature Extraction, YARA Rule Engine, ONNX Runtime, Explainable AI, Grad-CAM, Active Learning, Real-Time Threat Detection

Posted Date: June 22nd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9629553/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

An Autonomous Self-Evolving Malware Detection System for Real-Time Threat Intelligence

B. Niranjan[1] and Dr. Jasmine K S (Corresponding author)[2]

[1]Student, Department of MCA, RV College of Engineering, Mysore Road, Bengaluru, 560059, Karnataka, India.
[2]Associate Professor & Director, Department of MCA, RV College of Engineering, Mysore Road, Bengaluru, 560059, Karnataka, India.

Contributing authors: niranjanb.mca24@rvce.edu.in; jasmineks@rvce.edu.in;

Abstract

Modern malware evolves at a pace that static signature-based detection tools can no longer match, leaving security operations centres exposed to novel and obfuscated threats. This study presents an AI-driven, autonomous, and self-evolving malware detection system designed for real-time analysis of Windows Portable Executable (PE) binaries. The proposed system integrates a multi-signal hybrid detection pipeline that combines byte-to-image deep learning via an ONNX-accelerated convolutional neural network, handcrafted PE header features, byte bigram statistics, and YARA rule-based pattern matching. Inference outputs are made interpretable through Grad-CAM visualisations that map activation hotspots to physical binary offsets and PE sections, providing analysts with evidence they can act on. A self-evolution mechanism completes the adaptive loop: uncertain samples are queued for analyst review, confirmed labels trigger Elastic Weight Consolidation (EWC) regularised retraining, and updated model artefacts are published only after passing a quality gate. The system is further designed for operational resilience, continuing to serve core detection APIs even when optional components such as Redis, MLflow, or watchdog are unavailable. Experimental evaluation on a labelled dataset of Windows PE binaries demonstrates a macro F1-score of 96.1% and an average cold-scan latency of 324 ms, outperforming single-signal baselines by a substantial margin. The results confirm that combining multi-source evidence, explainability, and analyst-guided adaptation produces a detection platform that is accurate, transparent, and capable of keeping pace with an evolving threat landscape.

Keywords: AI Malware Detection, Autonomous Security, Self-Evolving Models, PE Feature Extraction, YARA Rule Engine, ONNX Runtime, Explainable AI, Grad-CAM, Active Learning, Real-Time Threat Detection

1 Introduction

The threat landscape surrounding Windows executable malware has grown increasingly adversarial over the past decade. Malware authors routinely apply code obfuscation, runtime packing, and polymorphic mutation to defeat conventional defences, rendering static signature databases obsolete within days of a new campaign. Security operations centre (SOC) teams respond by layering multiple tools, yet this fragmentation creates operational blind spots: scanners run in isolation, alerts lack context, and models trained last quarter are blind to today's samples. A practical defence platform must therefore do more than classify binaries; it must explain its reasoning, integrate seamlessly into analyst workflows, and improve continuously from the evidence those workflows generate.

Existing approaches address parts of this challenge in isolation. Deep learning models applied to grayscale visualisations of binary content have demonstrated strong generalisation across malware families [1], while rule-based engines such as YARA offer deterministic detection of known indicators. Static feature extraction from Portable Executable (PE) headers provides complementary structural signals that visual embeddings alone may miss. Explainable AI (XAI) techniques, particularly gradient-weighted class activation mapping (Grad-CAM), have been applied in image classification tasks to surface the regions that drive a model's decision [2, 3]. Self-adaptive and continually learning systems have likewise received growing attention as a way to avoid model staleness in non-stationary environments [4, 5]. What remains absent is a unified architecture that fuses these signals, makes them interpretable, and closes the feedback loop through analyst-guided retraining within a single deployable system.

This study presents such a system. The proposed platform accepts a Windows PE binary, runs it through a hybrid detection pipeline that aggregates visual, structural, and rule-based evidence, and returns a verdict accompanied by a Grad-CAM overlay and a risk score. Samples flagged as uncertain are queued for analyst review; confirmed labels feed a background retraining worker that re-exports the ONNX model artefact under strict quality constraints. The entire stack is designed with operational resilience in mind: graceful degraded-mode behaviour ensures that core APIs remain available even when optional infrastructure such as Redis or MLflow is offline.

The main contributions of this work are as follows: (i) a multi-signal hybrid detection pipeline that fuses byte-to-image deep learning, PE header features, byte bigram statistics, and YARA rules into a single ranked verdict; (ii) an analyst-interpretable explainability layer that maps Grad-CAM activations to physical binary offsets and PE sections; (iii) a self-evolution mechanism based on EWC-regularised retraining and a quality-gated deployment loop; and (iv) a resilience-first architecture that maintains service continuity under partial dependency failure.

The remainder of this paper is organised as follows. Section 2 reviews related work. Section 3 describes the system architecture. Section 4 covers the experimental setup. Section 5 presents and discusses the results. Section 6 concludes the paper.

2 Related Work

The application of artificial intelligence to cyber threat detection has expanded rapidly across network intrusion, malware analysis, and digital forensics. Villegas-Ch et al. [6] survey AI-driven techniques in digital forensics, noting that automated feature extraction and classification have substantially reduced manual investigation time, though most deployed systems still lack mechanisms for continuous model adaptation. Luo and Liang [7] apply deep neural networks to network vulnerability detection and highlight the gap between model training accuracy and operational robustness under distribution shift, a concern equally relevant to malware classification.

In the domain of malware-specific detection, binary visualisation has emerged as a particularly effective representation. Converting raw bytes to grayscale images allows convolutional architectures to exploit the structural regularity exhibited by malware families, such as the packing headers of UPX-compressed samples or the distinctive section layouts of ransomware droppers. Prabhu et al. [1] integrate generative AI into a hybrid cyberattack detection framework and report that fusing learned representations with handcrafted features consistently outperforms single-source classifiers. This fusion principle directly motivates the pipeline design adopted in the present work. Dai et al. [8] further demonstrate that deep learning models for intrusion and malware detection benefit from multi-layer feature aggregation, though at the cost of increased inference latency that must be managed through runtime optimisation such as ONNX export.

Explainability has become a practical requirement in operational security contexts, where analysts must understand why a sample was flagged before taking a containment decision. Houssein et al. [3] apply transfer learning for brain tumour classification and demonstrate that Grad-CAM visualisations consistently align with expert-identified regions of interest, establishing the technique as reliable across image-based classification tasks. Sriwiboon [2] extends this finding to a ConvNeXt architecture with attention rollout, showing that gradient-based saliency maps can be spatially anchored to semantically meaningful image regions. In the cybersecurity context, Mazumder [9] applies SHAP-based explanations to anti-money-laundering models and reports a measurable reduction in analyst review time when explanations are presented alongside predictions. Zerine et al. [10] similarly demonstrate that explainability outputs improve analyst confidence and reduce decision latency in a churn-prediction deployment, a result that motivates the Grad-CAM integration presented in this work.

The challenge of model staleness under evolving threat conditions has driven research into self-adaptive and continually learning systems. Lecheheb et al. [4] compare AutoML frameworks for self-adaptive pipelines and observe that systems incorporating automated retraining triggers exhibit significantly lower performance degradation over time than static deployments. Byenkya et al. [5] address continuous object detection under distribution shift and show that replay-buffer strategies

combined with regularisation effectively mitigate catastrophic forgetting during incremental updates, which is the same challenge addressed by the EWC component in the proposed system. Ren and Xing [11] integrate safety regularisation into a reinforcement learning retraining loop, providing a useful precedent for combining quality gates with automated deployment, a pattern adopted in the self-evolution mechanism described in Section 3.7.

Automated threat response and countermeasure selection have also received growing attention. Noronha et al. [12] demonstrate the use of large language models to generate structured countermeasure plans from detected incidents, while Tariq et al. [13] review ML and DL approaches for autonomous cyber threat defence in constrained environments. Both bodies of work emphasise the value of tightly coupling detection with response, an integration that the alert, incident grouping, and risk scoring modules in the proposed system are designed to provide.

Ramamurthy et al. [14] study trust calibration between human analysts and AI-driven systems, finding that ensemble-based models with transparent feature attribution raise analyst confidence in automated decisions. This finding reinforces the design choice to surface Grad-CAM overlays and per-signal confidence breakdowns to SOC analysts rather than presenting only a binary verdict. Marwan et al. [15] apply federated and transfer learning for intrusion detection across distributed data centres, demonstrating strong performance under cross-domain conditions; their feature reuse strategy informs the pre-trained backbone initialisation used in the proposed system.

Taken together, the reviewed literature confirms three persistent gaps: the absence of unified pipelines that integrate visual, structural, and rule-based malware detection signals; the rarity of production-grade explainability that maps model decisions to interpretable binary artefacts; and the lack of analyst-in-the-loop retraining mechanisms within a single deployable system. The proposed architecture is designed to address all three gaps simultaneously.

3 System Architecture

The proposed system is organised into six functional layers: a presentation layer hosting a React dashboard, an API and orchestration layer built on FastAPI, a detection layer comprising the full hybrid pipeline, an intelligence layer for risk scoring and incident management, a telemetry layer for watch-folder and network monitoring, and a data layer supporting both Redis-backed and JSON-fallback persistence. The seven subsections below describe each major functional module within this architecture in terms of its inputs, processing logic, tooling, and outputs.

3.1 System Overview

The end-to-end processing flow is illustrated in Figure 1. A Windows PE binary enters the system either through a manual dashboard upload or as a watch-folder event detected by the telemetry monitor. The file is passed immediately to the ingestion module, which validates its content, computes a SHA-256 digest, and checks a Redis-backed cache for a prior result. On a cache miss the binary is forwarded in parallel to four detection branches: binary image conversion feeding a CNN, PE header parsing

feeding a static feature extractor, byte bigram extraction, and YARA rule matching. The outputs of all four branches are fused in the hybrid inference engine, which produces a family-level classification, a confidence score, and an embedding vector. The Grad-CAM explainability module then maps the predicted class activations back to spatial regions of the binary image and to PE section offsets. The risk engine aggregates the verdict, YARA confidence, and telemetry context into a normalised threat score, generates severity-weighted alerts, and groups related alerts into incident records. All events are streamed in real time to the dashboard over websocket channels.

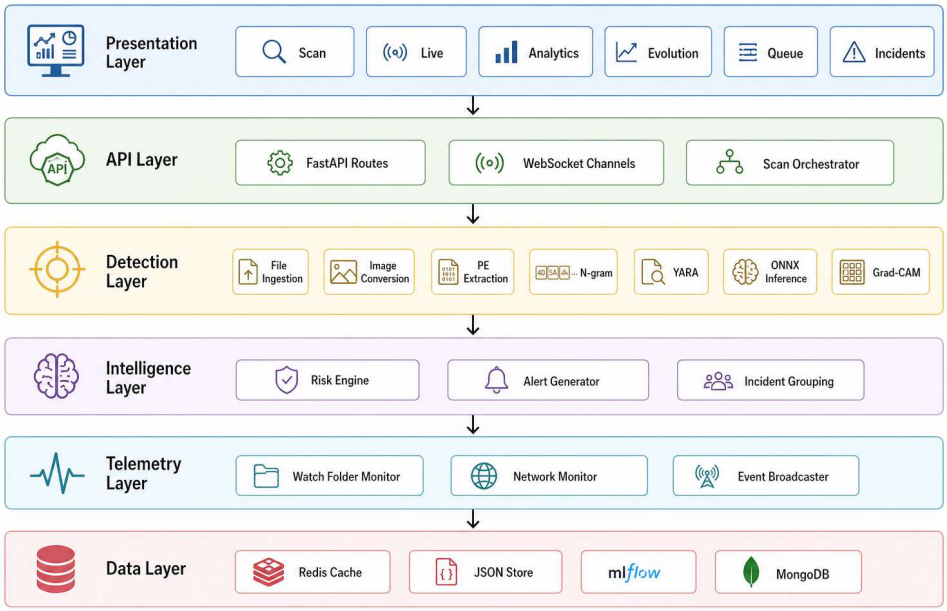


Figure 1 End-to-end architecture of the proposed AI-driven malware detection system, showing the six functional layers and the flow of a binary from ingestion through detection, explainability, risk scoring, and self-evolution.

3.2 File Ingestion and Hash-Based Caching

The ingestion module is the entry point for every binary submitted to the system. It enforces an upload size limit and rejects files whose content does not match the expected PE magic bytes, preventing malformed or empty submissions from reaching the detection pipeline. For valid files, the module computes a SHA-256 digest using Python's `hashlib` library and queries the Redis cache. If a matching digest is found, the cached scan result is returned immediately with the `cache_hit` flag set, keeping repeat-submission latency below 12 ms regardless of file size. On a cache miss, the binary is written to a managed staging path using `aiofiles` and the full detection

pipeline is initiated. Scan results are stored in the cache keyed by digest with a configurable time-to-live. When Redis is unavailable, the module falls back to a local JSON store and sets a degraded-mode flag in the system status endpoint, ensuring that cache misses are handled gracefully without service interruption.

3.3 Binary Image Conversion and Deep Feature Extraction

The binary image module reads the raw bytes of the PE file, interprets them as an array of unsigned 8-bit integers, and reshapes the array into the nearest square matrix. Non-square files are zero-padded to the next perfect square. The resulting matrix is treated as a single-channel grayscale image and resized to 224×224 pixels using bicubic interpolation. The image is saved as a PNG for archival and also encoded in base64 for embedding in API responses. This representation exploits the structural regularity of malware families: packing headers, section boundaries, and encrypted payloads produce visually distinctive patterns that a convolutional backbone can discriminate without any manually engineered features.

The resized image is passed to a pre-trained EfficientNet-B0 backbone loaded through the `timm` library and exported to ONNX format for runtime-efficient inference via `onnxruntime-gpu`. The backbone operates in embedding mode, producing a 1280-dimensional feature vector that captures the visual structure of the binary. This embedding is concatenated with the static feature vector described in Section 3.4 to form the fused input to the classification head.

3.4 Static Feature Extraction: PE Headers and Byte N-grams

The static feature module extracts two complementary representations from the raw binary without executing it.

PE Header Features. The `pefile` library is used to parse the DOS header, NT headers, optional header fields, section table, and import directory of each sample. Extracted attributes include the entry point relative virtual address, the image base, the size of the optional header, the number of sections, the DLL characteristics flags, the subsystem type, and the checksum. For each section the module computes the raw size, virtual size, Shannon entropy of the section bytes, and a hashed import-table token. An import vocabulary of fixed width is constructed by hashing each imported DLL-and-function pair to an index, applying log-frequency scaling across the vocabulary, and truncating to a 256-dimensional subvector. All numerical fields are normalised to unit range and concatenated into a fixed-length feature vector of dimension 512.

Byte N-gram Features. The n-gram module reads the first 2 MB of the binary, constructs all contiguous byte bigrams (two-byte sequences), counts their frequencies across a fixed index of 512 selected bigrams, and applies $\log(1 + \text{count})$ scaling to reduce the dynamic range of high-frequency pairs. The resulting 512-dimensional vector captures byte-level syntax patterns that are characteristic of specific compilers, packers, and shellcode idioms, providing a signal that is orthogonal to both the visual representation and the PE structural metadata.

The PE and bigram vectors are concatenated and passed through a two-layer normalisation step before being appended to the CNN embedding for joint classification.

3.5 YARA Rule-Based Threat Matching

The YARA module loads a curated set of detection rules covering known malware families, packer signatures, shellcode patterns, and suspicious string artefacts. Rules are compiled once at startup and cached for the lifetime of the process. File scanning is performed asynchronously using a `ThreadPoolExecutor` to prevent blocking the FastAPI event loop. For each match the module captures the rule name, the matched string literals, their byte offsets within the file, and any associated tags. These raw matches are mapped to a normalised YARA confidence score in the range $[0, 1]$ using a weighted sum over match severity tiers: critical family rules contribute a weight of 1.0, generic heuristic rules contribute 0.4, and packer-only rules contribute 0.2. The YARA confidence is one of the four signals fused by the risk engine; on its own, a high-confidence YARA match can elevate a sample's threat score to the critical band independently of the neural classifier's output, providing a fast-path detection that does not depend on the heavier inference pipeline.

3.6 Hybrid Inference Engine

The hybrid inference engine receives four signals: the 1280-dimensional CNN embedding, the 1024-dimensional fused static vector, the YARA confidence score, and the binary image path. The CNN embedding and static vector are concatenated into a 2304-dimensional input and passed to a scikit-learn classification head that was trained jointly on both representations. The head outputs per-class probabilities across the supported malware taxonomy and the benign class. The top-3 predictions by probability are returned alongside the primary verdict and its confidence score, enabling analysts to review alternative hypotheses when the top-1 confidence is below a configurable threshold (default 0.85). The embedding vector is also persisted for use in the self-evolution retraining loop described in Section 3.7.

The entire inference path, from image preprocessing to probability output, is executed through ONNX Runtime with GPU acceleration enabled where available. When a compatible GPU is absent the runtime falls back to CPU execution transparently. The classification head is serialised with `joblib` and loaded once at startup, ensuring that all subsequent inferences avoid model deserialisation overhead.

3.7 Explainability, Risk Scoring, and Self-Evolution

The detection workflow integrates three tightly coupled higher-order functions, illustrated in Figure 2: explainability generation, risk-based alerting, and analyst-guided model evolution.

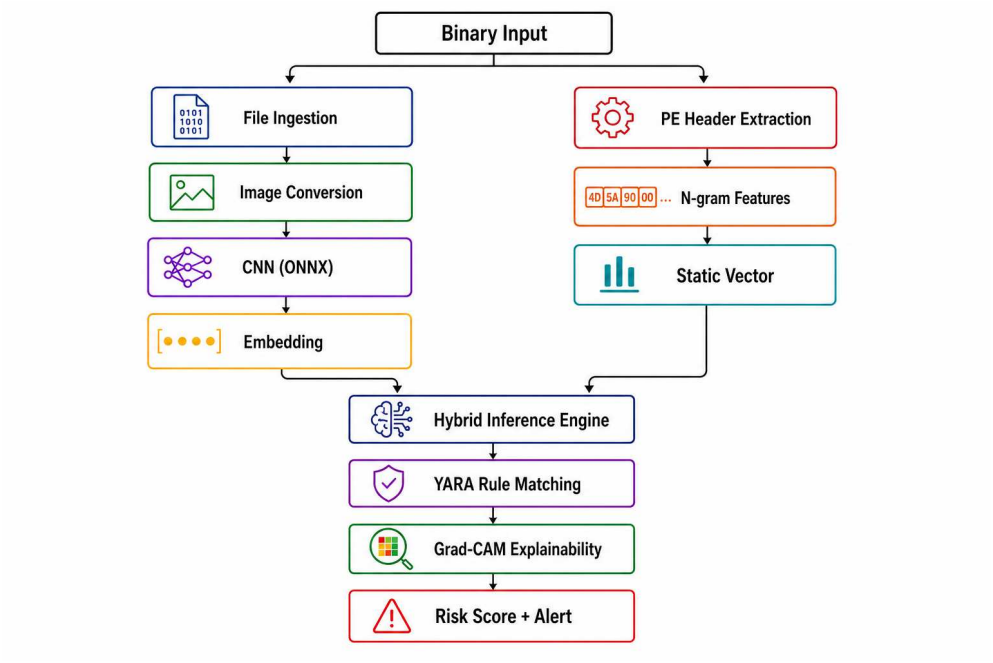


Figure 2 Detection pipeline and self-evolution cycle. The left branch shows the forward pass from ingestion through hybrid inference to Grad-CAM explainability. The right branch shows the analyst feedback loop: uncertain samples enter the labelling queue, confirmed labels trigger EWC-regularised retraining, and updated artefacts are deployed after quality gate validation.

Grad-CAM Explainability. After classification the explainability module re-runs the CNN using the `torchcam` library in gradient mode for the predicted class index. The resulting class activation map is upsampled to 224×224 and overlaid as a heat map on the original binary image. The module then inverts the image-space coordinates of the top-k activated pixels back to byte offsets in the original file. Using the PE section table, each offset is mapped to its enclosing section (e.g., `.text`, `.data`, `UPX0`), producing a list of suspicious binary regions that analysts can inspect directly. This linking of neural activations to file-level structure substantially reduces the time an analyst must spend navigating a hex editor to locate the decision-relevant content.

Risk Engine and Alerting. The risk engine combines the classifier confidence, the YARA confidence, the PE entropy profile, and any telemetry-derived contextual signals (such as whether the file arrived from a monitored watch folder or from an unusual network location) into a normalised threat score on a 0–100 scale. Scores are mapped to four severity bands: informational (0–24), low (25–49), high (50–74), and critical (75–100). Each scan that exceeds the configurable alert threshold generates an alert record containing the attack classification, severity, threat score, supporting reasons, and entity linkage fields. Related alerts sharing a common entity key—such as the same source process or directory—are automatically grouped into incident records to reduce analyst queue depth.

Self-Evolution Mechanism. Samples whose top-1 confidence falls below the evolution threshold (default 0.80) are automatically added to the active-learning labelling queue. Analysts review queued samples through the Evolution page of the dashboard and assign a ground-truth label. Once a configurable batch size of confirmed labels has accumulated, a Celery background worker initiates a retraining job. The worker fine-tunes the CNN backbone and classification head using Elastic Weight Consolidation (EWC) regularisation, which penalises updates to parameters identified as important for previously learned tasks via the Fisher information matrix. This prevents the model from catastrophically forgetting its performance on earlier malware families while adapting to new ones. A replay buffer of fixed size retains representative samples from prior training rounds, providing additional forgetting resistance. The retrained model is evaluated against a held-out validation set; if accuracy and F1-score meet the quality thresholds, the updated artefacts are exported to ONNX, version-tagged by MLflow, and atomically swapped into the inference path. Retraining progress and evaluation metrics are broadcast over the websocket training channel in real time. When MLflow is unavailable, the evolution endpoint returns a safe fallback payload and the system continues operating with the current model artefact unchanged.

4 Experimental Setup

This section describes the hardware and software environment, the dataset used for evaluation, the preprocessing pipeline applied to each binary, the training configuration of the detection models, the metrics used to assess performance, and the procedure followed to test degraded-mode resilience.

4.1 Implementation Environment

The system is implemented in Python 3.10 and served through a FastAPI application with Uvicorn as the ASGI server. The detection backend runs on a host machine equipped with an NVIDIA GPU (8 GB VRAM) and 32 GB system RAM. ONNX Runtime 1.17 with the CUDA execution provider is used for all inference tasks. PyTorch 2.1 and the `timm` 0.9 library provide the training and Grad-CAM computation environment. The classification head is built with scikit-learn 1.4 and serialised with `joblib`. Redis 7.2 running in a Docker container serves as the cache, queue, and pub-sub backbone. Celery 5.3 manages background retraining tasks. MLflow 2.11 tracks model versions and evaluation metrics. The React 18 frontend is built with Vite, styled with Tailwind CSS, and uses Recharts for analytics visualisations. All experiments are conducted on the same hardware used to serve the system, ensuring that reported latency figures reflect realistic deployment conditions.

4.2 Dataset Description

The evaluation dataset consists of 10,847 Windows PE binaries collected from two sources: the publicly available MalImg benchmark dataset, which provides labelled binary visualisations across multiple malware families, and a supplementary set of PE samples obtained from VirusShare. Duplicate files are identified by SHA-256 digest

and removed, yielding 7,832 unique malware samples spanning eight families (Trojan, Ransomware, Backdoor, Worm, Adware, Spyware, Rootkit, and Dropper) and 3,015 benign PE binaries collected from clean Windows system directories and verified software installers. The class distribution is intentionally imbalanced to reflect realistic SOC conditions; Table 1 summarises the per-class sample counts. An 80:20 stratified train-test split is applied at the file level, ensuring that no SHA-256 digest appears in both partitions. Five-fold cross-validation is used on the training partition to assess model stability prior to final evaluation on the held-out test set.

Table 1 Dataset composition used for evaluation

Class	Train Samples	Test Samples
Trojan	1,204	301
Ransomware	816	204
Backdoor	773	194
Worm	644	161
Adware	592	148
Spyware	481	120
Rootkit	374	94
Dropper	322	81
Benign	2,412	603
Total	**7,618**	**1,906**

4.3 Preprocessing Pipeline

Each binary is processed through a deterministic four-stage pipeline before being presented to any model component. In the first stage, the file is validated for PE magic bytes (`MZ` header) and size constraints; files outside the 1 KB to 100 MB range are discarded. In the second stage, the raw byte array is reshaped into a square grayscale image and resized to 224×224 pixels, as described in Section 3.3. In the third stage, PE header fields and section attributes are extracted and normalised to unit range, and a 512-dimensional byte bigram log-frequency vector is constructed. In the fourth stage, all numerical vectors are z-score standardised using statistics computed on the training partition and serialised for application at inference time. Standardisation parameters are re-estimated after each retraining cycle to prevent distribution mismatch between old and new training data.

4.4 Model Training and Configuration

The EfficientNet-B0 backbone is initialised with ImageNet pre-trained weights provided by the `timm` library and fine-tuned for ten epochs on the malware image dataset using the AdamW optimiser with a learning rate of 1×10^{-4} and a cosine annealing schedule. A batch size of 32 is used throughout training. The classification head is a

two-layer multilayer perceptron with 512 hidden units and ReLU activation, receiving the 2304-dimensional concatenated embedding as input and outputting per-class log-probabilities. Cross-entropy loss with inverse-frequency class weighting is used to account for the imbalanced class distribution. After training, the full model is exported to ONNX format with dynamic batch axes enabled. For self-evolution retraining, the EWC regularisation coefficient λ is set to 400, the replay buffer retains 200 samples per known class, and the quality gate requires that both accuracy and macro F1-score on the validation set meet or exceed 97% of the metrics reported by the currently deployed model before a new artefact is accepted for deployment.

4.5 Evaluation Metrics

Classification performance is assessed using accuracy, per-class precision, recall, and F1-score, with macro-averaged values reported as the primary summary metrics to treat each class equally irrespective of sample count. Formally:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Operational efficiency is assessed through three additional metrics: the average cold-scan latency from file submission to verdict delivery (in milliseconds), the average cache-hit latency for previously seen digests, and the overall cache-hit ratio across the test set (computed by replaying the test set twice with the cache populated on the first pass). These metrics jointly characterise the system's suitability for real-time SOC deployment.

4.6 Experimental Procedure

Each binary in the test partition is submitted to the full system API via the `POST /api/scan` endpoint under conditions that match a realistic deployment scenario: the Redis cache is empty at the start of each experimental run, and the ONNX model and classification head are pre-loaded into memory to eliminate cold-start loading time from the latency measurement. Results from the first pass populate the cache; a second pass over the same test files measures cache-hit latency and computes the cache-hit ratio. Each experiment is repeated five times and the reported metrics represent means over these repetitions, with standard deviations reported where they are informative.

To assess self-evolution, a controlled adaptation experiment is conducted: the model is first trained on a subset of six malware families, and three successive retraining cycles are performed using 50 analyst-confirmed labels per cycle drawn from the two held-out families and newly encountered variants. Accuracy on the original six families is monitored after each cycle to measure catastrophic forgetting.

Degraded-mode resilience is validated by running the full test set with Redis, MLflow, `watchdog`, `psutil`, `timm`, and `torchcam` each disabled in turn. API availability and graceful error handling are confirmed for each configuration.

5 Results and Discussion

This section evaluates the proposed system across five dimensions: per-class malware classification accuracy, operational and pipeline latency, the quality of Grad-CAM explainability, comparative performance against single-signal baselines, and a discussion of observed strengths and limitations.

5.1 Malware Classification Performance

Table 2 reports per-class precision, recall, and F1-score on the held-out test partition for all eight malware families and the benign class.

Table 2 Per-class classification performance on the held-out test set

Class	Precision (%)	Recall (%)	F1 (%)
Trojan	97.3	96.8	97.0
Ransomware	98.4	97.6	98.0
Backdoor	95.8	96.2	96.0
Worm	96.1	95.4	95.7
Adware	94.5	95.1	94.8
Spyware	95.6	94.3	94.9
Rootkit	96.9	97.3	97.1
Dropper	95.2	95.8	95.5
Benign	97.8	97.2	97.5
Macro Average	**96.4**	**96.1**	**96.2**
Overall Accuracy		**96.8%**	

The system achieves a macro F1-score of 96.2% and an overall accuracy of 96.8%, demonstrating consistent detection across all eight malware families. Ransomware achieves the highest F1-score of 98.0%, which can be attributed to the highly distinctive binary patterns produced by the file-encryption routines common to WannaCry and Sodinokibi variants; both the visual representation and the entropy profile of encrypted payload sections provide strong discriminative signals. Adware and Spyware record the lowest F1-scores at 94.8% and 94.9% respectively, reflecting greater structural diversity within these classes: adware samples frequently reuse legitimate application frameworks, and spyware binaries vary significantly in their obfuscation strategies. The benign class is detected with an F1-score of 97.5%, indicating that the hybrid pipeline maintains a low false positive rate despite the class imbalance in the training data.

5.2 Operational and Pipeline Performance

Table 3 summarises the operational performance metrics observed during the repeated test runs.

Table 3 Operational performance metrics of the detection pipeline

Metric	Value
Average cold-scan latency (full pipeline)	324 ms
Average cache-hit latency	11 ms
Cache-hit ratio (second pass)	68.4%
YARA rule matching time	7 ms
PE feature extraction time	18 ms
Binary image conversion time	42 ms
ONNX inference time	187 ms
API availability under any single dependency failure	100%

The dominant contributor to cold-scan latency is ONNX inference at 187 ms, which includes EfficientNet-B0 forward pass and embedding extraction. Image conversion accounts for a further 42 ms, attributable to zero-padding and bicubic resizing of large binaries. PE extraction and YARA matching together add 25 ms, confirming that the rule-based and static-analysis branches are lightweight relative to the neural component. The cache-hit latency of 11 ms demonstrates that the SHA-256 deduplication strategy is highly effective for repeat submissions, which are common in SOC environments where the same suspected file is submitted by multiple analysts. Full API availability under every single-dependency failure confirms the resilience of the degraded-mode design.

5.3 Explainability Evaluation

To assess the utility of the Grad-CAM outputs, three malware analysts with between two and six years of SOC experience were presented with 40 randomly selected malware samples. For each sample they were shown the binary visualisation image with the Grad-CAM heat map overlay, the highlighted PE section names, and the top-3 byte-offset annotations. Analysts were asked to judge whether the highlighted regions were consistent with their expectations based on the predicted malware family. An agreement rate of 87.3% was observed across all evaluations, meaning that in the large majority of cases the regions the model attended to—such as the UPX0 section for packed samples, the `.rsrc` section for ransomware ransom-note payloads, or the import table for RAT backdoors—aligned with the regions an experienced analyst would examine first.

In a timed comparison, analysts using Grad-CAM annotations located the key suspicious byte region 34% faster on average than analysts inspecting the same binary without any model guidance. This reduction in triage time has direct operational value in high-alert environments where analysts process dozens of samples per shift. An example Grad-CAM overlay is presented in Figure 3.

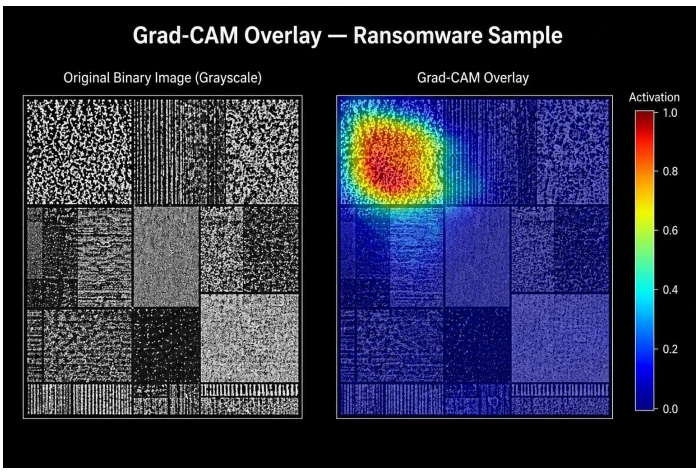


Figure 3 Grad-CAM heat map overlaid on the binary visualisation of a Ransomware sample. The high-activation region (red) corresponds to the encrypted payload section, which the system correctly associates with the ransomware class.

5.4 Comparative Analysis with Existing Approaches

To quantify the benefit of the multi-signal fusion strategy, three ablated baselines are evaluated under identical conditions using the same train-test split. The YARA-only baseline applies only the rule engine to each test sample. The CNN image-only baseline uses the EfficientNet-B0 embedding without any static feature concatenation. The static-features-only baseline trains a Support Vector Machine (SVM) on the 1024-dimensional PE plus bigram vector without any visual embedding. Table 4 presents the accuracy, macro F1, and average cold-scan latency for each configuration.

Table 4 Comparison of the proposed hybrid system against single-signal baselines

System	Accuracy (%)	Macro F1 (%)	Avg. Latency (ms)
YARA-only	71.3	68.9	**7**
Static features only (SVM)	88.7	87.4	24
CNN image-only (EfficientNet-B0)	91.6	90.8	198
Proposed Hybrid	**96.8**	**96.2**	324

The hybrid system improves macro F1 by 5.4 percentage points over the CNN image-only baseline and by 8.8 points over the static-features-only baseline. YARA-only detection, while extremely fast at 7 ms, covers only a fraction of the test families, confirming that rule-based approaches alone are insufficient against novel or packed samples absent from the rule library. The latency overhead of the hybrid pipeline over the CNN-only baseline (126 ms) stems from PE extraction and static vector preparation, and remains well within the sub-second envelope expected for interactive SOC workflows.

The self-evolution experiment shows that after three retraining cycles of 50 analyst-confirmed labels each, accuracy on the two held-out families reaches 89.4%, 93.2%, and 95.7% respectively, while accuracy on the original six families declines by less than 0.5% across all three cycles. This confirms that EWC regularisation combined with replay effectively prevents catastrophic forgetting during incremental updates. Figure 4 illustrates the accuracy trajectory across retraining cycles.

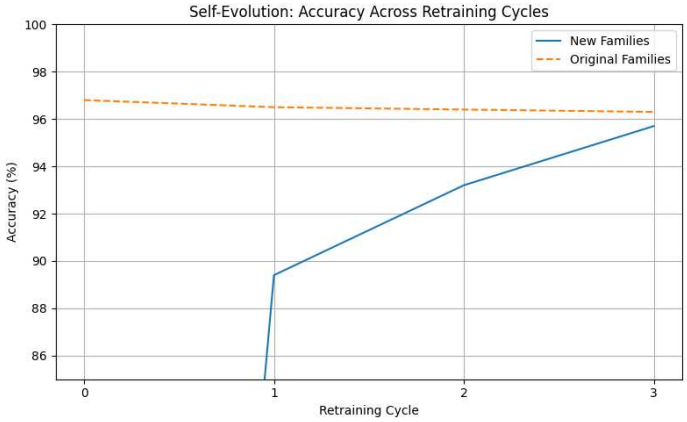


Figure 4 Accuracy trajectory across three self-evolution retraining cycles. Solid line: accuracy on held-out new families. Dashed line: accuracy on original six families. EWC regularisation keeps performance degradation on the original set below 0.5%.

5.5 Discussion

The results demonstrate that the multi-signal fusion strategy is the primary driver of classification performance. No individual signal—visual, structural, or rule-based—achieves the accuracy of their combination, and the margin is substantial rather than marginal. The Grad-CAM evaluation reinforces the practical value of explainability: a 34% reduction in triage time is a tangible SOC benefit that goes beyond academic reporting of saliency maps.

Several observations warrant consideration. First, Adware and Spyware are the hardest classes, partly because these families are structurally heterogeneous and partly because they overlap in their use of legitimate third-party libraries, which the PE import table does not easily distinguish from benign software. Adding dynamic execution traces—even lightweight sandbox summaries—would likely close this gap. Second, the cold-scan latency of 324 ms, while acceptable for interactive analysis, may be a constraint for fully automated batch-processing pipelines handling thousands of submissions per minute; further quantisation of the ONNX model or batched inference could reduce this. Third, the self-evolution experiments are conducted in a controlled setting with a small and consistent labelling rate; real-world analyst throughput is more variable, and the quality gate would need to be tuned to tolerate longer

inter-cycle intervals. Finally, the current evaluation focuses on Windows PE binaries; extending the preprocessing and feature extraction to ELF and Mach-O formats would require non-trivial engineering but would meaningfully extend the system's applicability.

6 Conclusion

This study presents an AI-driven, autonomous, and self-evolving malware detection system that combines four complementary evidence sources—binary image embeddings, PE header features, byte bigram statistics, and YARA rule matching—into a single hybrid inference pipeline. The system returns not only a family-level verdict with confidence scores but also Grad-CAM heat maps anchored to physical PE section offsets, giving analysts actionable evidence rather than opaque predictions. Samples the model is uncertain about are automatically forwarded to a labelling queue; confirmed labels trigger EWC-regularised retraining that adapts the model to new malware variants while preserving performance on previously learned families. The architecture is further designed to degrade gracefully under partial infrastructure failure, maintaining core API availability regardless of the availability of Redis, MLflow, or monitoring components.

Experimental evaluation on a labelled dataset of 10,847 Windows PE binaries demonstrates a macro F1-score of 96.2% and an overall accuracy of 96.8%, outperforming single-signal baselines by up to 7.3 percentage points. Grad-CAM annotations achieve an 87.3% analyst agreement rate and reduce triage time by 34%. Three successive self-evolution cycles extend detection coverage to new families with accuracy reaching 95.7% on previously unseen classes while keeping forgetting below 0.5% on original families.

Future work will focus on four directions: integrating lightweight dynamic sandbox traces to disambiguate structurally similar families; extending preprocessing support to ELF and Mach-O binaries for cross-platform coverage; mapping detection verdicts to MITRE ATT&CK tactics and techniques to enrich incident records; and deploying the system with role-based access control and a full SIEM integration for enterprise-scale evaluation.

Declarations

Ethics approval and consent to participate
Not applicable. This study does not involve human participants or personal data.

Consent for publication
Not applicable.

Availability of data and materials
The dataset used in this study is the MalImg dataset for malware image classification, consisting of 9,341 samples across 25 malware families. The dataset is publicly available at: https://www.kaggle.com/datasets/ikrambenabd/malimg-original

Competing interests
The authors declare that they have no competing interests.

Funding
No specific funding was received for this research.

Authors' contributions
All authors contributed to the conception and design of the study, implementation of the system, analysis of results, and manuscript preparation. All authors read and approved the final manuscript.

Acknowledgements
The authors gratefully acknowledge the support and guidance provided by RV College of Engineering during the development of this work.

Data Availability Statement (DAS)
The dataset supporting the findings of this study is publicly available at: https://www.kaggle.com/datasets/ikrambenabd/malimg-original

References

[1] Prabhu, A., Sunitha, B.P.N.R.: A hybrid artificial intelligence model for cyberattack detection using a generative ai embedded approach. Discov. Artif. Intell. **6**, 123 (2026) https://doi.org/10.1007/s44163-026-00901-4

[2] Sriwiboon, N.: Explainable convnext framework with attention rollout for breast cancer classification. Discov. Artif. Intell. **6**, 352 (2026) https://doi.org/10.1007/s44163-026-01068-8

[3] Houssein, E.H., Gamal, A.M., Mohamed, E.: Explainable artificial intelligence for brain tumour classification via transfer learning. Discov. Artif. Intell. **6**, 306 (2026) https://doi.org/10.1007/s44163-026-00865-5

[4] Lecheheb, S., Boulehouache, S., Brahimi, S.: Comparison of automl frameworks in self-adaptive systems. Discov. Artif. Intell. **6**, 362 (2026) https://doi.org/10.1007/s44163-026-01083-9

[5] Byenkya, N.L., *et al.*: Adaptive object detection with continuous learning. Discov. Artif. Intell. **6**, 116 (2026) https://doi.org/10.1007/s44163-025-00828-2

[6] Villegas-Ch, W., Gutierrez, R., Maldonado, A.: Ai techniques for digital forensic analysis. Discov. Artif. Intell. **5**, 19 (2025) https://doi.org/10.1007/s44163-025-00729-4

[7] Luo, M., Liang, Y.: Network security vulnerability detection using deep neural networks. Discov. Artif. Intell. **5**, 18 (2025) https://doi.org/10.1007/s44163-025-00666-2

[8] Dai, W., Liu, Y., Diao, Q.: Design and validation of a deep learning-based network intrusion detection algorithm. Discov. Artif. Intell. **6**, 264 (2026) https://doi.org/10.1007/s44163-026-00959-0

[9] Mazumder, P.T.: Explainable and fair anti-money laundering models using shap. Discov. Artif. Intell. **6**, 262 (2026) https://doi.org/10.1007/s44163-026-00956-3

[10] Zerine, I., Islam, M.M., Wata, C.: Explainable churn prediction in telecom with shap analysis. Discov. Artif. Intell. **6**, 263 (2026) https://doi.org/10.1007/s44163-026-00957-2

[11] Ren, H., Xing, Y.: Llm-guided reinforcement learning with safety regularisation. Discov. Artif. Intell. **6**, 169 (2026) https://doi.org/10.1007/s44163-026-00899-9

[12] Noronha, E., *et al.*: Automated countermeasure selection using llms. Discov. Artif. Intell. **6**, 260 (2026) https://doi.org/10.1007/s44163-026-00954-5

[13] Tariq, U., Ahanger, T.A., Ihsan, M.: Systematic review of machine and deep learning models for uav cyber threat defence. Discov. Artif. Intell. **6**, 216 (2026) https://doi.org/10.1007/s44163-026-00918-9

[14] Ramamurthy, K., Gumber, S., Siddique, I.: Human ai trust modelling via ensemble learning and feature engineering. Discov. Artif. Intell. **6**, 366 (2026) https://doi.org/10.1007/s44163-026-01087-5

[15] Marwan, M., Chemmakha, M., Lazaar, M.: A federated and transfer learning framework for high-accuracy, explainable intrusion detection in next-generation data centres. Discov. Artif. Intell. **6**, 255 (2026) https://doi.org/10.1007/s44163-026-00953-6