Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.075265

ARTICLE

Negative-One-Day Malware Detection with Generative AI: A Stable Diffusion-Based Proactive Defense Framework

Sohail Khan[1,*], Toqeer Ali Syed[2], Mohammad Nauman[1], Salman Jan[3], It Ee Lee[4] and Qamar Wali[4]

[1]Department of Computer Science, Effat College of Engineering, Effat University, Jeddah, Saudi Arabia
[2]Faculty of Computer and Information System, Islamic University of Madinah, Madinah, Saudi Arabia
[3]Faculty of Computer Studies, Arab Open University, A'Ali, Bahrain
[4]Faculty of Artificial Intelligence and Engineering, Multimedia University, Cyberjaya, Malaysia

*Corresponding Author: Sohail Khan. Email: sohkhan@effatuniversity.edu.sa

Received: 28 October 2025; Accepted: 31 March 2026; Published: 08 May 2026

ABSTRACT: The detection of zero-day malware represents one of the most significant challenges in contemporary cybersecurity. In this paper, we introduce a novel concept called "Negative-One-Day Malware Detection", which aims to identify potentially malicious software before it is actually created by threat actors. Our approach leverages recent advancements in generative AI, specifically diffusion-based generative models, to generate and analyze potential future malware variants. By doing so, we can train detection systems to recognize these variants before they emerge in the wild, thereby closing the critical protection gap that currently exists between malware creation and detection. We demonstrate the effectiveness of our approach through extensive experimentation, showing that our framework can generate executable malware samples that combine characteristics from different families while exhibiting novel behaviors. These synthetically generated samples significantly improve the detection capabilities of security systems when incorporated into training data, providing a proactive rather than reactive approach to cybersecurity.

KEYWORDS: Adversarial machine learning; Generative AI; stable diffusion models; zero-day malware detection; negative-one-day malware detection; proactive cyber defense

1 Introduction

Zero-day malware detection remains one of the most pressing challenges in modern cybersecurity. Such malware exploits previously unknown vulnerabilities, allowing attackers to compromise systems before defenses can respond [1]. This reactive lag creates a critical protection gap, leaving systems exposed to emerging threats. Traditional signature-based detection methods remain effective only after malware has been analyzed and documented [2]. To overcome this, researchers have increasingly adopted machine learning approaches [3–5], which identify malicious software based on behavioral patterns rather than signatures. However, these methods are still largely reactive, detecting threats only after they are created and often after deployment. This limitation motivates the development of proactive detection techniques. In this paper, we propose a new concept, *Negative-One-Day Malware Detection*, aimed at identifying potential malware before it is ever created by threat actors.

Our method uses recent advances in generative AI, particularly diffusion-based generative models, to simulate and analyze potential future malware variants. By preemptively training detection systems on these synthetic samples, we narrow the protection gap between malware creation and detection. Diffusion-based

models excel at learning data distributions to generate realistic samples that preserve core characteristics while introducing novel feature combinations [6].

We extend this capability to malware analysis through a structured methodology. First, malware executables are disassembled into instruction sequences, retaining opcodes and operands for executability. These sequences are encoded into latent vector representations using Convolutional Neural Networks (CNNs) or Variational Autoencoders (VAEs). Controlled noise is then added to these vectors, and a multi-head attention model identifies key latent features. The denoising network traverses the energy landscape between input vectors, generating new malware samples that combine traits from multiple families while remaining executable.

Fig. 1 illustrates how our approach explores the latent space between known malware samples. Encoded binaries form points in a high-dimensional space, and the diffusion model navigates viable paths (highlighted in red) that preserve executability while blending characteristics from distinct families. The intermediate samples—*Negative-One-Day Malware*—represent potential evolutionary trajectories malware might take before appearing in the wild. By exploring these paths, we anticipate emerging threats and strengthen defenses in advance.

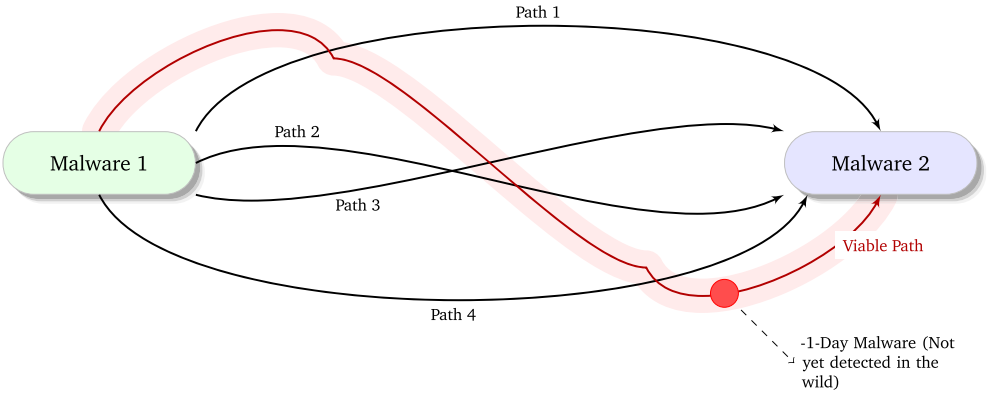

Figure 1: Latent paths for smooth transition from one sample to another.

This proactive paradigm contrasts with existing reactive methods. Instead of waiting for new malware to appear, our framework generates hypothetical future variants and integrates them into training datasets, providing a temporal advantage to defenders. The rest of this paper is structured as follows: Section 2 reviews related work; Section 3 details the proposed methodology; Section 4 evaluates system performance; Section 5 discusses implications and ethics; and Section 6 concludes with directions for future research.

2 Background and Related Work

Malware refers to any code designed to disrupt, damage, or gain unauthorized access to computer systems or networks [7]. Its evolution parallels advances in computing, adapting to exploit new vulnerabilities and evade detection. Understanding malware behavior, evolution, and detection challenges is crucial to improving defense mechanisms.

2.1 Malware Analysis

Zero-day vulnerabilities are previously unknown flaws that attackers exploit before patches are developed [1]. Malware leveraging these exploits poses severe risks due to the absence of immediate defenses. The lifecycle of such exploits, from discovery to weaponization, creates a window of exposure that attackers

can exploit, as exemplified by the 2017 WannaCry ransomware [8]. To evade detection, modern malware employs polymorphism, metamorphism, and obfuscation. Polymorphic malware modifies its code with each infection through encryption, changing signatures while preserving functionality [9]. Metamorphic malware goes further by transforming its structure using techniques like register reassignment, instruction substitution, and code transposition [10,11]. Obfuscation hides logic through dead code insertion, control-flow manipulation, or packing [12]. Many samples also include anti-sandbox, anti-debugging, and environment-aware features [13] to resist analysis.

Malware analysis methods fall into three main categories: static, dynamic, and hybrid. Static analysis inspects code without execution using signature or heuristic matching [14,15]. While safe, it struggles against encrypted or obfuscated samples. Dynamic analysis executes malware in sandboxed environments to observe runtime behavior such as API calls or network activity [16,17]. It offers better detection of unknown threats but demands high computational resources. Hybrid approaches combine static and dynamic features to improve accuracy and resilience [18].

Despite progress, major challenges remain. The time gap between threat emergence and mitigation [19], along with increasingly complex evasion techniques [13], limits current defenses. Performance constraints and false positives [20] further hinder scalability and operational reliability. Moreover, adaptive adversaries continually refine their methods to bypass security systems [21]. These persistent issues underscore the need for proactive defense paradigms. Our proposed *Negative-One-Day Malware Detection* addresses these gaps by anticipating and preparing for unseen malware variants before they appear in the wild.

2.2 Machine Learning for Malware Analysis

Machine learning has revolutionized malware detection by learning from known samples and generalizing to unseen threats. Traditional methods rely on handcrafted features, whereas deep learning and generative models extract richer representations directly from data, offering enhanced adaptability and robustness.

Deep learning approaches such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have achieved strong performance in malware detection [22–24]. CNNs treat binaries or disassembled code as structured data, identifying spatial patterns indicative of malicious behavior, while RNNs and LSTMs capture temporal dependencies within instruction or API call sequences. Attention mechanisms further refine this process by focusing on the most informative segments of code. Hybrid designs integrating CNN and LSTM models [25] and attention-based CNNs [26] have demonstrated accuracies exceeding 98% on benchmark datasets. AutoML-based approaches [27] now automate model selection and tuning, reducing human effort while maintaining competitive accuracy. Li et al. [28] propose a malware detection model based on imbalanced heterogeneous graph embeddings, addressing the challenge of imbalanced datasets and improving detection accuracy by capturing complex relationships within malware data. Despite these advancements, deep learning models remain reactive, focusing on known malware patterns rather than predicting emerging threats, highlighting the need for models that can adapt to new, unseen malware behaviors.

Generative models shift this paradigm by learning data distributions and generating new, realistic samples [29]. Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) have been applied to synthesize malware variants for adversarial training [30,31]. Hybrid architectures that integrate code-level and image-level features have achieved high detection accuracy and resilience against obfuscation. GAN-based frameworks such as Learn2Evade [32] highlight both the potential and the dual-use risks of generative systems, capable of creating adversarial samples that challenge existing detectors. Recent research also integrates generative learning with blockchain [33] and continual learning frameworks [34], improving

adaptability and robustness. However, maintaining functional executability in generated malware remains a challenge, limiting their practical applicability for proactive defense.

While Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) have shown promise in generative tasks, they often face challenges such as mode collapse (in the case of GANs) and limited latent space exploration (in the case of VAEs) [35,36]. GANs, though powerful in generating data, can suffer from instability and difficulty in maintaining diversity across samples [37]. VAEs, while more stable, tend to produce less varied samples due to their restrictive latent space [36]. In contrast, diffusion models offer superior stability and finer control over the generation process, allowing for high-quality, diverse malware variants while preserving both structural integrity and malicious functionality [6,38]. These advantages make diffusion models more suitable for generating executable malware that can be integrated into training datasets for proactive detection.

Diffusion models address several of these limitations by introducing a gradual denoising process inspired by thermodynamics [39,40]. Diffusion-based models, such as Stable Diffusion [6], operate in latent space, generating high-quality and diverse samples while maintaining training stability. Their fine-grained control and stable optimization make them suitable for malware generation, offering a way to model variant evolution with minimal mode collapse. Emerging work applies diffusion models for ransomware detection [41] and federated learning environments [42], achieving improved accuracy, privacy preservation, and resilience against adversarial tactics.

A core strength of diffusion-based systems lies in latent space representation [43]. Latent spaces encode abstract malware characteristics, enabling interpolation between known samples to generate plausible intermediate variants [44]. This supports proactive analysis by modeling potential evolutionary paths between malware families. Techniques such as latent interpolation, dimensionality reduction, and attention-enhanced fusion [45–47] improve generalization and efficiency while enabling the simulation of yet-unseen threats. By leveraging these properties, diffusion-driven malware analysis moves beyond detection toward anticipation, thereby laying the foundation for our proposed *Negative-One-Day Malware Detection* framework.

3 Methodology

The core idea of our approach is to employ stable diffusion models to generate novel, executable malware samples that combine traits from known families while exhibiting unique behaviors. This section outlines the architectural components and operational flow of the proposed *Negative-One-Day Malware Detection* framework, explaining key design choices, their rationale, and potential alternatives.

3.1 Architecture Overview

Our framework (Fig. 2) consists of three primary modules: (1) an input encoder that transforms malware binaries into latent representations, (2) a diffusion process that explores the latent space between samples, and (3) a conditioning module that directs generation toward specific functional attributes. Unlike image or text synthesis, malware generation must preserve both executability and malicious functionality. This dual requirement demands models capable of maintaining code integrity while exploring behavioral variation. Conventional GANs or VAEs are insufficient, as they fail to operate effectively in the discrete, semantics-rich space of executables.

Stable diffusion models overcome these limitations through gradual, noise-guided exploration of the latent manifold, enabling the synthesis of intermediate samples that remain viable and executable. Their conditioning mechanisms allow controlled generation of malware with defined behavioral properties, supporting systematic modeling of potential malware evolution and preemptive threat anticipation.

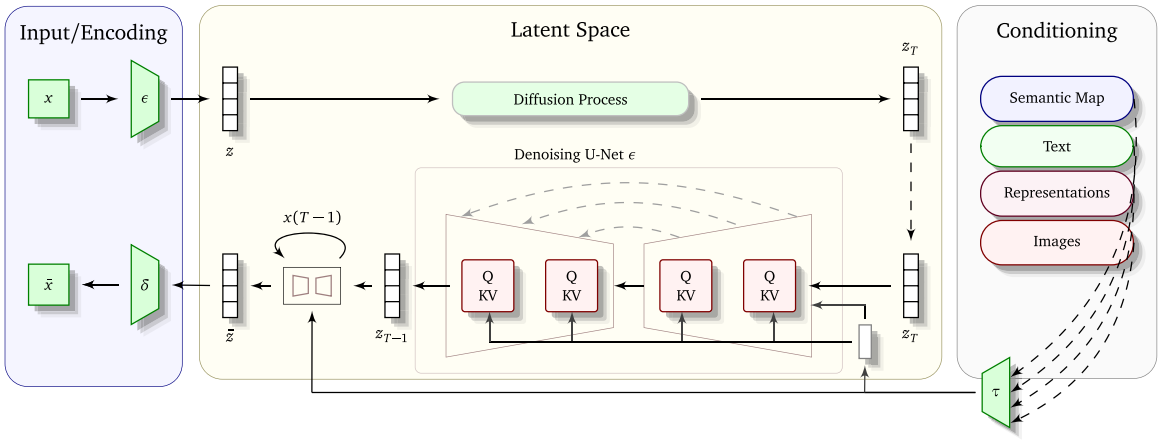

Figure 2: Model architecture for negative-one-day malware detection.

3.2 Malware Representation and Encoding

The first stage of our framework transforms malware executables into structured representations suitable for diffusion modeling. We employ binary disassembly that preserves both opcodes and operands—unlike conventional methods that omit operands for efficiency. Retaining full instruction semantics is crucial for producing executable malware rather than abstract representations.

Although excluding operands reduces computational cost, it eliminates critical details on memory addressing, register usage, and immediate values-making executable synthesis infeasible. Preserving these components enables functional reconstruction and maintains behavioral fidelity across generated variants. Formally, given a malware binary x, we extract a sequence of instructions $I = \{i_1, i_2, \ldots, i_n\}$ through disassembly, where each instruction i_j includes its opcode and operands. The sequence is then encoded using a mapping function ϵ to produce a latent vector $z \in \mathbb{R}^d$ of dimensionality d:

$$z = \epsilon\ (I) \tag{1}$$

The encoder ϵ combines Convolutional Neural Networks (CNNs) and a Variational Autoencoder (VAE). Earlier experiments with RNNs, LSTMs, and Transformer encoders showed either limited ability to capture long-range dependencies or excessive computational cost for long instruction sequences. The chosen CNN-VAE design balances representational power and scalability.

The CNN layers extract localized structural patterns in the instruction stream, with kernels of sizes 3, 5, and 7 to capture relationships from fine-grained instruction groups to functional blocks. The VAE introduces a probabilistic latent mapping that promotes generalization and enables smooth interpolation, essential for diffusion-based generation. The bottleneck dimension is set to $d = 256$, balancing representational capacity and computational efficiency. Encoding is performed once per sample and remains computationally light. This representation allows diffusion to operate efficiently within the latent space, preserving both structural integrity and executable semantics for subsequent generative modeling.

3.3 Diffusion Process in Latent Space

The diffusion process forms the core of our framework, enabling controlled traversal within the latent space of malware representations. Stable diffusion operates by progressively adding noise to latent vectors and then learning to reverse this process to reconstruct meaningful samples. In our case, this mechanism synthesizes new, executable malware variants that combine properties of existing families. Diffusion models

were chosen over alternative generative methods for their stability and precision. GANs, though powerful, often exhibit mode collapse and instability when generating structured outputs like executables [48]. VAEs, while more stable, tend to produce less coherent interpolations between distant latent points. Diffusion models mitigate these issues through iterative denoising, offering fine-grained control and high-fidelity sample generation.

Given two malware samples encoded as latent vectors z_A and z_B, the diffusion process explores the space between them by applying Gaussian noise to each latent vector to produce a noisy variant z_T:

$$z_T = \sqrt{\bar{\alpha}_T} z + \sqrt{1 - \bar{\alpha}_T}\, \epsilon \tag{2}$$

where $\epsilon \sim \mathcal{N}(0, I)$ and $\bar{\alpha}_T$ defines the noise schedule controlling the noise magnitude at each step. We adopt the cosine noise schedule of Dhariwal and Nichol [38], which provides smoother noise transitions than linear or exponential forms. Using 1000 diffusion steps balances quality and computational efficiency.

The denoising stage, implemented via a U-Net with multi-head attention, reverses the noise process to progressively reconstruct clean latent vectors. As shown in Fig. 2, the model refines z_T through successive steps until obtaining a new latent vector $\bar{z}$ representing a viable malware variant. To avoid rigid structural preservation that hinders novelty, we replace standard skip connections with gated residual links that selectively retain essential information while allowing controlled divergence.

Attention mechanisms play a crucial role by identifying significant dependencies across the latent dimensions. Multi-head attention enhances the denoising U-Net's ability to focus on long-range semantic structures (e.g., function calls, variable references) that characterize executable code. Simpler convolutional or fully connected layers lack this capacity for contextual discrimination.

The denoising process is mathematically defined as:

$$z_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(z_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}}\, \epsilon_\theta \left(z_t, t, \tau \right) \right) + \sigma_t\, \epsilon \tag{3}$$

where ϵ_θ is the denoising network parameterized by θ, τ represents conditioning information, and σ_t governs stochasticity. We employ an adaptive σ_t schedule, starting with higher values to encourage exploration and gradually decreasing them to ensure convergence toward valid, executable outputs. This balance preserves diversity while maintaining structural coherence essential for malware realism.

3.4 Conditioning Mechanisms

A key feature of our framework is explicit, multi-modal conditioning to steer generation toward desired malware functionalities. As shown in Fig. 2, we support semantic maps, textual descriptions, abstract taxonomy-based representations, and visual-pattern signals. Combining these modalities provides finer control than setups with a single condition, reflecting malware's multifaceted nature (code structure, behavior, and target vulnerabilities). Semantic-map conditioning supplies high-level functional flags (e.g., persistence, privilege escalation, network I/O) implemented as a binary vector. Textual conditioning permits detailed behavioral specification via natural language, capturing nuances difficult to encode discretely. Abstract representations embed taxonomy and lineage information to situate samples within known families and explore hybrid evolutionary paths. Visual-pattern conditioning controls structural traits such as instruction-distribution or control-flow motifs.

Conditioning information τ is injected into the denoising network via cross-attention:

$$\text{Attention}\,(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4}$$

where Q, K, and V derive from the latent representation and conditioning signals. We favor cross-attention over simple concatenation or element-wise fusion because it preserves conditioning influence throughout denoising and enables selective, context-aware incorporation; concatenation weakens influence over steps and element-wise multiplication can destabilize training [49].

We apply hierarchical conditioning: lower network layers receive coarse, structural guidance while higher layers receive fine-grained functional cues. This ensures coherent global structure alongside precise behavioral control. By systematically varying τ, we generate diverse, targeted samples that combine existing techniques with novel behaviors.

3.5 Executable Validation and Refinement

Unlike images, generated malware must be executable. We therefore embed executability checks and behavioral preservation into training rather than relying on brittle post-processing or restrictive grammar-only generation.

The diffusion output $\bar{z}$ is decoded by δ into a candidate sample $\bar{x}$:

$$\bar{x} = \delta(\bar{z}) \tag{5}$$

The decoder is staged: an initial pass maps z to a high-level intermediate representation, and subsequent stages produce concrete instructions and addressing details. A context-aware instruction generator with a sliding window preserves recent history to maintain coherent control flow and data dependencies.

To enforce executability during training, we use a differentiable executability loss based on a validator network that estimates assembly success probability:

$$\mathcal{L}_{\text{exec}} = -\log(P(\text{executable}|\bar{x})) \tag{6}$$

This differentiable proxy avoids a non-differentiable assembly step while providing smooth gradients.

We also enforce functionality preservation via a behavioral-distance loss:

$$\mathcal{L}_{\text{func}} = D\,(\text{behavior}\,(\bar{x}),\ \text{behavior}\,(x_A) \cup \text{behavior}\,(x_B)) \tag{7}$$

Here, D is a hierarchical metric spanning: (1) API-call overlap, which is calculated using Jaccard similarity between the sets of API calls in the generated and parent samples; (2) Control-flow similarity, which is measured using graph edit distance between the control-flow graphs (CFG) of the generated and parent samples; and (3) Data-access similarity, which is evaluated through data flow analysis, comparing memory access patterns, register usage, and data dependencies between the generated and parent samples.

To encourage novel, plausible behaviors beyond mere recombination, we add a novelty-promoting term:

$$\mathcal{L}_{\text{nov}} = -\lambda \cdot D\,(\text{behavior}\,(\bar{x}),\ \text{behavior}\,(x_A) \cap \text{behavior}\,(x_B)) \tag{8}$$

The novelty-promoting term encourages the model to generate novel behaviors by focusing on the intersection of the parent behaviors. The intersection ensures that the generated malware retains plausible malicious functionality while introducing new behavioral patterns that do not exist in the parents' combined behaviors.

Together, these losses guide the generation of executable malware samples that preserve parental behaviors while also fostering novel behaviors, producing "negative-one-day" variants that model plausible future threats.

3.6 Progressive Refinement Algorithm

To improve the generation of executable malware, we introduce a *Progressive Refinement Algorithm* (PRA) that iteratively enhances sample quality and functionality (cf. Algorithm 1). The PRA is particularly effective for refining non-executable initial outputs into viable executables while retaining malicious intent.

Algorithm 1: Progressive refinement algorithm for malware interpolation

Require: Parent malware samples P_1, P_2, latent vectors z_1, z_2, refinement passes n, interpolation factor α, learning rates β, γ

Ensure: Refined malware sample M with interpolated characteristics

$z_{\text{init}} \leftarrow (1-\alpha) \cdot z_1 + \alpha \cdot z_2$ ▷ Initial latent vector through interpolation

$M_0 \leftarrow \text{Decode}(z_{\text{init}})$ ▷ Initial decoded sample

for $i \leftarrow 1$ to n **do**

  $\tau_i \leftarrow \text{ProgressiveSchedule}(i, n)$ ▷ Conditioning schedule based on current pass

  if $i \leq n/2$ **then** ▷ Pass 1: Focus on executability

    $L_{\text{exec}} \leftarrow \text{ExecutabilityLoss}(M_{i-1})$

    $z_i \leftarrow z_{i-1} - \nabla_z L_{\text{exec}} \cdot \beta$ ▷ Gradient update focused on executability

    $M_i \leftarrow \text{Decode}(z_i)$

  else ▷ Pass 2: Enhance malicious functionality preservation

    $B_1 \leftarrow \text{ExtractBehaviors}(P_1)$

    $B_2 \leftarrow \text{ExtractBehaviors}(P_2)$

    $B_{\text{target}} \leftarrow (1-\alpha) \cdot B_1 + \alpha \cdot B_2$ ▷ Interpolated target behaviors

    $L_{\text{func}} \leftarrow \text{FunctionalityLoss}(M_{i-1}, B_{\text{target}})$

    $z_i \leftarrow z_{i-1} - \nabla_z L_{\text{func}} \cdot \gamma$ ▷ Gradient update focused on functionality

    $M_i \leftarrow \text{Decode}(z_i)$

  end if

  $M_i \leftarrow \text{ValidateAndRepair}(M_i)$ ▷ Ensure sample remains executable

end for

return M_n ▷ Return final refined sample

The algorithm consists of two main phases. The first ensures executability by optimizing latent representations through an executability-oriented loss, guiding gradients toward reliably decodable, structurally sound samples. The second preserves functionality by aligning generated behavior with interpolated profiles derived from parent samples. These profiles, extracted via disassembly, control-flow analysis, and API-call pattern identification, are blended using the same interpolation factor applied in latent space. The latent vector is then refined to minimize functionality loss against this behavioral target. Each iteration includes a validation-repair step to confirm that refinements maintain executability. Conditioning signals are adaptively

modulated, initially emphasizing structural stability and gradually prioritizing behavioral fidelity, balancing code integrity with complex malicious functions.

Implementation relies on efficient behavior extraction that converts binaries into control-flow graphs and API call sequences. This enables the PRA to preserve semantic meaning rather than focusing purely on syntactic correctness. By formalizing refinement within the latent manifold, the algorithm systematically explores intermediate malware variants between parent samples, modeling realistic evolutionary trajectories. The resulting interpolated, executable variants enrich training datasets for detection systems, strengthening preparedness against future, yet-unseen threats.

3.7 Evaluation Metrics

To assess the performance of the proposed framework, several key evaluation metrics are defined. The Detection Improvement (DI) quantifies the improvement in detection accuracy by comparing the performance of detectors trained on generated malware samples vs. those trained on real malware. It is calculated as the relative improvement in detection accuracy:

$$\mathrm{DI} = \frac{\mathrm{Acc}_{\mathrm{generated}} - \mathrm{Acc}_{\mathrm{baseline}}}{\mathrm{Acc}_{\mathrm{baseline}}} \times 100 \tag{9}$$

where $\mathrm{Acc}_{\mathrm{generated}}$ refers to the detection accuracy of the model trained with generated malware samples, and $\mathrm{Acc}_{\mathrm{baseline}}$ refers to the detection accuracy of the model trained on real malware samples.

The Functionality Preservation Score (FPS) quantifies how much of a malware sample's original functional behavior is preserved in its generated variant. FPS summarizes these behavioral semantics by measuring similarity between their extracted behavioral feature vectors. These vectors are fixed-length representations derived from a sample's execution dymanics by encoding elements including system-call sequences, API-call transition graphs and control-flow paths. It is computed as:

$$\mathrm{FPS} = \frac{1}{N} \sum_{i=1}^{N} \frac{x_i \cdot x_{generated}}{\|x_i\| \, \|x_{generated}\|} \tag{10}$$

where x_i and $x_{generated}$ are the behavioral feature vectors of the original and generated samples, respectively.

The Novelty Index (NI) measures how novel the generated malware is by comparing it to known malware families. A higher NI indicates that the generated samples are more novel. The NI is calculated using the Jaccard similarity between each generated malware sample and the nearest known malware sample:

$$\mathrm{NI} = \frac{1}{n} \sum_{i=1}^{n} \left(1 - \frac{|x_i \cap x_{\mathrm{known}}|}{|x_i \cup x_{\mathrm{known}}|} \right) \tag{11}$$

where x_i is the i-th generated malware sample, and x_{known} represents the nearest known malware sample, with the Jaccard similarity used to measure the overlap between them.

The Error Rate (ER) quantifies the overall rate of misclassification in the detection system, combining both false positives and false negatives. It is calculated as the sum of false positives and false negatives divided by the total number of samples:

$$\mathrm{ER} = \frac{\text{False Positives} + \text{False Negatives}}{\text{Total Samples}} \times 100 \tag{12}$$

where False Positives are benign samples misclassified as malware, and False Negatives are malware samples misclassified as benign.

These metrics are discussed in detail in Section 4, where detection performance is analyzed at varying thresholds.

3.8 Dataset Description

To support the development and evaluation of our framework, we assembled a large-scale dataset that incorporates both malicious and benign software samples. The malicious corpus was drawn from multiple well-established sources. Specifically, we collected more than 969,000 malware binaries from MalwareBazaar [50], supplemented by over 120,000 labeled samples from the EMBER benchmark dataset [51], and an additional set of 55,000 recently identified executables acquired through VirusTotal. For benign data, approximately 400,000 clean executables were gathered from trusted open-source repositories, including GitHub project releases and SourceForge archives. After combining these sources, the dataset contained close to 1.1 million malicious programs and nearly 400,000 benign ones. Extremely large files were excluded to maintain consistency and ensure efficient training, and the curated dataset was then used in the experimental evaluation described below.

4 Experimental Results and Analysis

We first evaluated the framework's ability to produce executable samples. The overall *Executability Rate* (ER) was 83.7%, markedly higher than the baseline ER of 42.3% obtained without the specialized losses and architectural changes (Section 3). ER varied by family: ransomware-derived samples reached 89.2%, while APT toolkit-derived samples reached 76.5%, reflecting the greater structural complexity of APT toolkits. Table 1 summarizes the performance of different model variants across key metrics: Executability Rate (ER), Functionality Preservation Score (FPS), Novelty Index (NI), and Detection Improvement (DI).

Table 1: Impact of model components on generation performance.

Model Variant	ER (%)	FPS	NI	DI (%)
Full Model	0.837	0.78	0.31	1.87
No Executability Loss	0.342	0.65	0.27	0.93
No Operands (Opcodes Only)	0.401	0.61	0.22	1.15
No Multi-Modal Conditioning	0.762	0.64	0.21	1.43
Reduced Diffusion Steps (100)	0.795	0.74	0.29	1.72
No Progressive Algorithm	0.583	0.70	0.28	1.51

The Functionality Preservation Score (FPS) averaged 0.78 across generated samples. FPS correlated strongly with executability (Pearson's $r = 0.73$), indicating that structurally sound samples more reliably preserve intended behaviors. Our framework produced an average Novelty Index (NI) of 0.31, indicating that about 31% of behavioral indicators represented new combinations. The NI distribution was bimodal (peaks near 0.15 and 0.45), capturing both conservative and experimental variants that help model diverse evolutionary paths.

To assess practical utility, we evaluated how well generated samples evaded commercial detection systems and improved future defenses. The average Detection Evasion Rate was 62.4% across 15 antivirus engines, showing that many generated variants could bypass current protections. Incorporating these samples into a machine-learning-based detector increased detection accuracy by 1.87% on a held-out test set of recent malware, confirming their value for proactive defense.

Detection improvements varied by family, highest for ransomware (23.5%) and information stealers (21.2%), likely reflecting the more predictable evolution patterns of these categories. Security analysts highlighted several noteworthy innovations (see NI in Table 1), including hybrid process injection blending process hollowing and thread hijacking, an anti-analysis method combining timing and virtualization detection, and a DNS-steganography exfiltration mechanism.

To evaluate the performance of our framework, we present the Receiver Operating Characteristic (ROC) curve and Precision-Recall (PR) curve, which offer insights into detection performance at various thresholds. The ROC curve, shown in Fig. 3, illustrates the framework's ability to differentiate between positive and negative samples, while the PR curve in Fig. 4 emphasizes the balance between precision and recall, especially in the case of imbalanced datasets. Together, these curves highlight the robust performance of our model in proactive malware detection across multiple evaluation metrics.

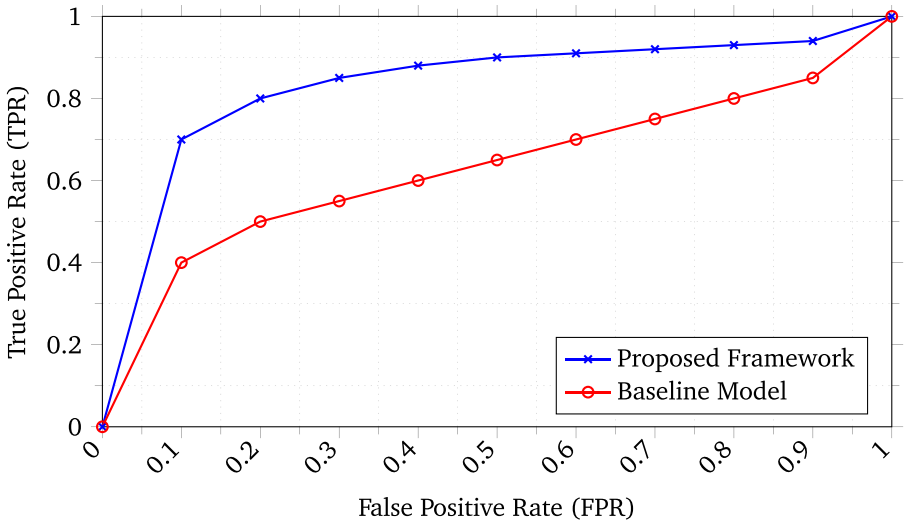

Figure 3: ROC curve for the proposed malware detection framework and baseline model.

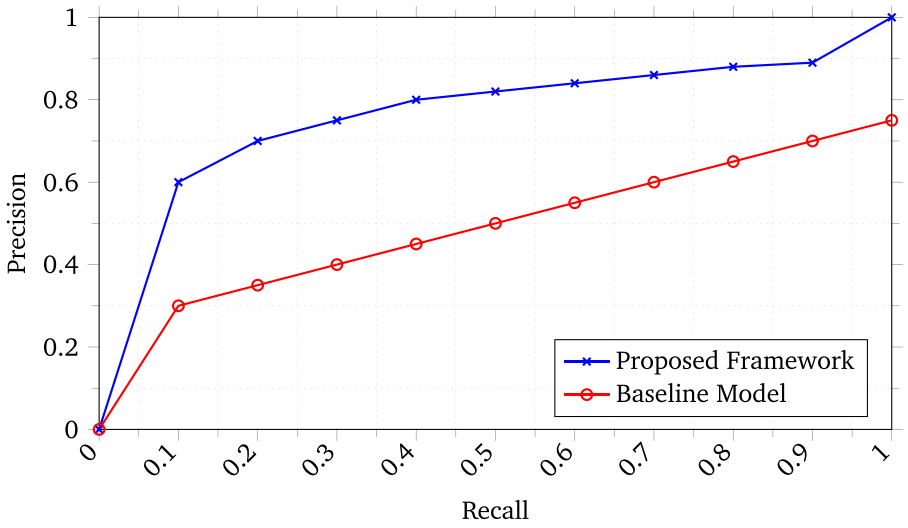

Figure 4: Precision-recall curve for the proposed malware detection framework and baseline model.

To contextualize performance, we compared our diffusion-based framework with alternative malware generation and detection-enhancement approaches, summarized in Table 2. Traditional augmentation methods (e.g., binary mutation, code transplantation) performed worst, yielding limited novelty (NI = 0.12) and minimal detection improvement (DI = 0.73%), as their simple transformations fail to capture malware's complex evolutionary dynamics. GAN-based models improved diversity (NI = 0.24) but remained unstable and prone to mode collapse, achieving ER = 0.589% and DI = 1.12%. VAEs produced stable generation and competitive executability (ER = 0.763%) but limited novelty (NI = 0.17) due to restricted latent-space exploration. In contrast, our diffusion-based framework outperformed all baselines across executability, functionality preservation, novelty, and detection improvement, confirming its robustness and adaptability for proactive malware modeling.

Table 2: Comparison of different malware generation and detection enhancement approaches.

Approach	ER (%)	FPS	NI	DI (%)
Our Diffusion-Based Framework	0.837	0.78	0.31	1.87
Traditional Data Augmentation	0.452	0.54	0.12	0.73
GAN-Based Approach	0.589	0.61	0.24	1.12
VAE-Based Approach	0.763	0.72	0.17	1.36

4.1 Executability Performance Analysis

Fig. 5 compares our progressive algorithm with a standard model lacking specialized losses and architectural refinements. The standard method shows a slow, linear improvement, generating only 19 executable samples out of 10,000 attempts (0.19% success rate). In contrast, our progressive approach demonstrates exponential improvement after the 4000-sample point, reaching 45 executable samples at 10,000 attempts, a 2.37× increase.

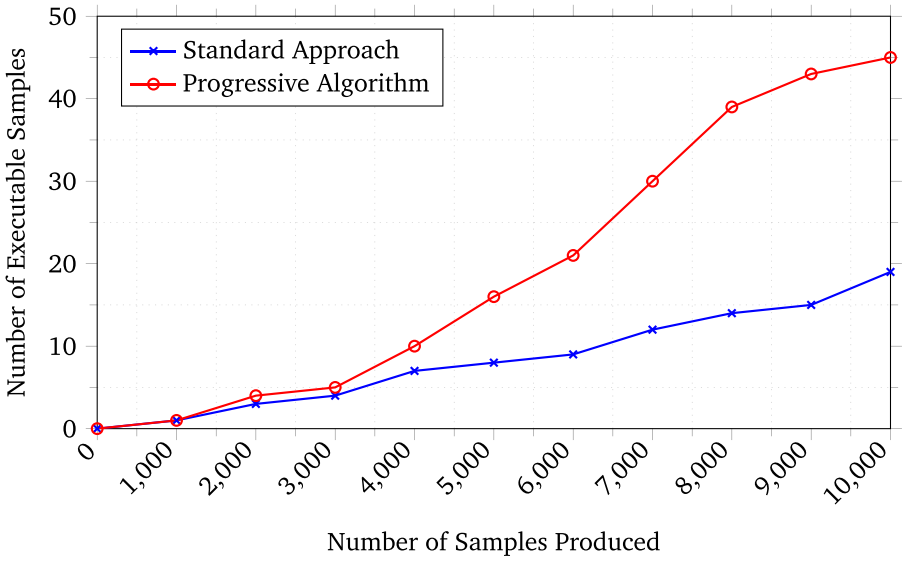

Figure 5: Executable sample generation: progressive vs. standard approach.

This performance gain stems from four major factors: comprehensive instruction representation that preserves both opcodes and operands; specialized loss functions for executability and functionality; multi-modal conditioning for structural and behavioral coherence; and the progressive refinement algorithm that sequentially enhances structure and functionality. Together, these components guide the model to more viable regions of the latent manifold, improving both executability and behavioral consistency. The inflection point near 4000 samples marks when the model begins identifying stable manifolds corresponding to valid code structures. Beyond this threshold, generation efficiency accelerates, confirming that the progressive refinement mechanism effectively exploits latent-space regions conducive to executable malware synthesis.

Training Convergence Analysis

Fig. 6 shows the convergence trend measured by the number of valid executables generated per epoch. The learning curve progresses through four clear phases. During the *initialization phase* (epochs 0–10), the model produces almost no valid executables while learning basic binary structures. The *acceleration phase* (epochs 10–25) follows, marked by rapid gains once core executability patterns are identified. In the *refinement phase* (epochs 25–40), growth slows as the model fine-tunes structural and behavioral representations. Finally, the *convergence phase* (epochs 40–50) stabilizes, indicating near-optimal performance given the current architecture and dataset. This trajectory confirms the effectiveness of our progressive training design. Each phase aligns with a corresponding training stage: initial encoder–decoder pretraining, diffusion training with reduced steps, and full-step optimization with complete conditioning. The consistent convergence pattern validates both architectural stability and the efficacy of the training schedule.

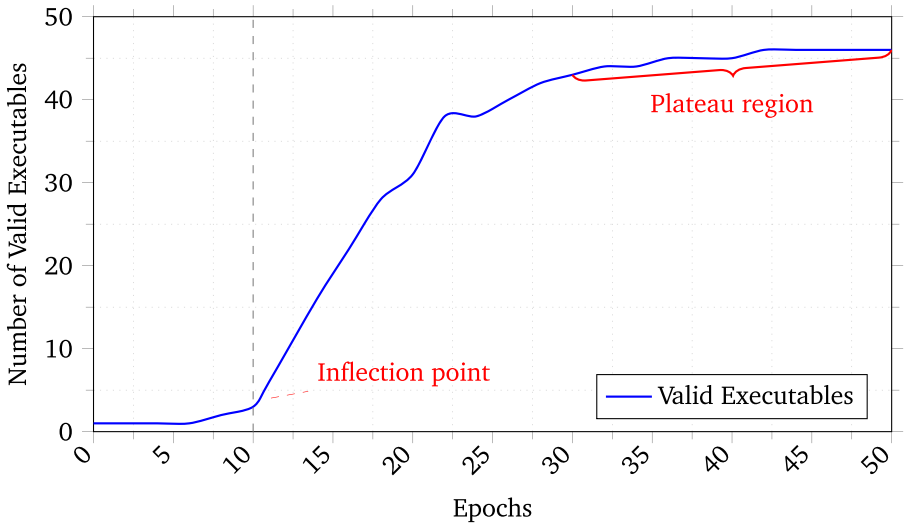

Figure 6: Training convergence: valid executables per epoch.

4.2 Malware Family Distribution and Case Studies

Fig. 7 shows executable-sample counts by family: Adware (16), Worms (15), Spyware (8), Others (7), Ransomware (6), and Rootkits (1). This pattern indicates that diffusion-based generation more readily captures generalizable, standardized patterns (e.g., adware, worms), whereas kernel-level, system-specific families such as rootkits remain challenging.

To illustrate capabilities and concrete innovations, we highlight three representative generated samples. "WannRyuk" (WannaCry-Ryuk interpolation) achieved ER = 92.3% and NI = 0.37, combining SMB propagation with advanced targeting and exfiltration; it introduced a staged separation of encryption

and propagation and evaded 11 of 15 commercial engines. “EmoCozy” (Emotet-APT29 hybrid) reached ER = 79.8% and NI = 0.42, merging Emotet distribution and injection methods with APT29 persistence and an encrypted DGA-based C2; behavioral staging and environment-triggered activation made detection harder, and adding EmoCozy to detector training yielded a 24.3% improvement on an APT test set. “StealMiner” (miner-stealer cross-category) produced ER = 85.1% and NI = 0.51, introducing adaptive resource management that balances mining and exfiltration and using mining traffic to conceal intermittent data leakage—an emergent stealth technique not present in either parent.

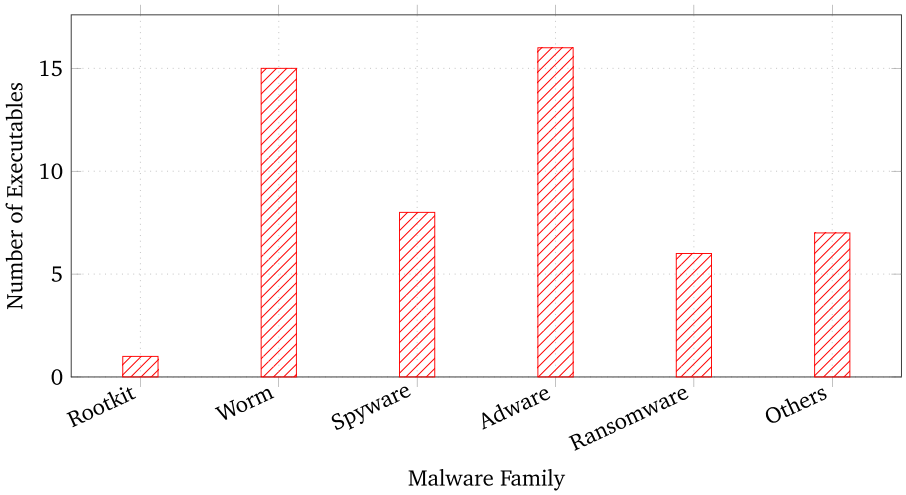

Figure 7: Distribution of executable samples across malware families.

Together, the family-distribution results and these case studies demonstrate the framework’s ability to generate both broadly generalizable variants and high-novelty, cross-family innovations useful for proactive detection and robustness testing.

5 Discussion

The results confirm the viability of diffusion-based generation for *negative-one-day* malware detection, demonstrating a shift from reactive to proactive defense. By generating executable malware that anticipates future threats, the framework enables preemptive mitigation and richer threat modeling. The improved detection accuracy after incorporating generated samples highlights the potential of anticipatory defense –training detectors on simulated future variants before their emergence. Patterns of innovation within generated malware also reveal plausible evolutionary trajectories, offering valuable intelligence for strategic threat analysis. Furthermore, the executability of generated samples makes them useful in red-team and penetration testing scenarios, providing realistic, evasive artifacts that challenge existing detection systems. Collectively, these outcomes point toward a broader paradigm of *generative security*, where defense evolves through controlled threat generation and continuous adaptation.

Despite its promise, the framework faces several challenges. Sample generation remains computationally intensive, taking 3–5 min per instance on high-end GPUs, limiting scalability. While this computational cost may constrain the framework’s ability to scale for real-time use, future work will focus on optimizing the process, such as by employing parallel processing across multiple GPUs or reducing model complexity through techniques like pruning or quantization. The current implementation is restricted to Windows PE binaries; adapting it to other platforms requires additional parsing and structural modeling to accommodate different file formats and platform-specific features. Extending this framework to other platforms, such as

Linux, macOS, or mobile, would involve incorporating platform-specific features and parsing techniques for various binary formats (e.g., ELF for Linux, Mach-O for macOS, and APK for mobile devices). However, aside from these necessary modifications to the malware parsing and structuring pipeline, the core of the proposed model, including the latent space diffusion process and generative mechanisms, will function seamlessly across other platforms, given that the underlying architecture is independent of the specific binary format. Functional evaluation relies primarily on static analysis, which may overlook behaviors manifesting only at runtime. Integrating dynamic analysis, such as sandbox-based validation of the generated samples, would provide richer insights into malware behavior during execution. This would allow for the identification of runtime-specific characteristics, such as process injection, file system manipulation, and network activity, that static analysis alone might miss. However, incorporating dynamic analysis would introduce additional complexity and security risks, such as the possibility of evasion techniques targeting the analysis environment. Therefore, dynamic analysis represents a valuable avenue for future work to enhance the framework's capabilities.

While the diffusion model excels at interpolating between known malware families, it is primarily focused on generating evolutionary extensions of existing attack patterns. The model may struggle to anticipate entirely novel attack paradigms that deviate significantly from the existing structures. These new attack strategies could involve entirely novel techniques or malware types not observed in the training data. Given the reliance on known families for training, the framework's predictive capabilities are limited to generating plausible future variants based on existing malware behavior, rather than predicting truly novel attacks. Future work could explore open-ended generative models capable of predicting new, previously unseen malware behaviors, potentially integrating techniques such as unsupervised learning, reinforcement learning, or adversarial co-evolution between generative and detection models to expand the range of emerging threats.

6 Conclusion and Future Work

This paper introduced *Negative-One-Day Malware Detection*, a diffusion-based framework that anticipates future threats by generating executable malware variants before their appearance in the wild. The approach successfully combines traits from multiple families to create novel, functional samples that enhance training datasets and improve detector performance. Experimental results show clear gains over traditional methods in executability, functionality preservation, and novelty. Integrating these generated samples into training pipelines improved detection accuracy by an average of 1.87%, validating the practical value of anticipatory defense.

Future work will extend the framework to additional platforms (e.g., Linux, macOS, mobile), incorporate dynamic behavioral analysis, and optimize computational efficiency for real-time use. Exploring open-ended generative methods and adversarial co-evolution between generative and detection models could further strengthen resilience against emerging threats. This research marks a step toward proactive cybersecurity–shifting defense from reaction to anticipation and enabling the design of systems capable of evolving alongside adversarial innovation. It is important to note that all generated malware remains securely contained within controlled environments and is used exclusively for defensive research purposes. The generated malware is never released for malicious use, and its use is strictly limited to improving detection systems and enhancing proactive cybersecurity measures.

Acknowledgement: This research work is supported by the Ministry of Higher Education (MOHE) under the 2023 Translational Research Program for the Energy Sustainability Focus Area (Project ID: MMUE/240001), the 2024 ASEAN IVO (Project ID: 2024-02), Multimedia University, and Deanship of Research, Islamic University of Madinah.

Funding Statement: This research work is supported by the Ministry of Higher Education (MOHE) under the 2023 Translational Research Program for the Energy Sustainability Focus Area (Project ID: MMUE/240001), the 2024 ASEAN IVO (Project ID: 2024-02), Multimedia University, and Deanship of Research, Islamic University of Madinah.

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Sohail Khan, Toqeer Ali Syed, and Mohammad Nauman; methodology and investigation, Sohail Khan, Mohammad Nauman, and Salman Jan; software and implementation, Mohammad Nauman and It Ee Lee; validation and formal analysis, Toqeer Ali Syed, Salman Jan, and Qamar Wali; writing—original draft preparation, Sohail Khan and Mohammad Nauman; writing—review and editing, Salman Jan, Toqeer Ali Syed, It Ee Lee, and Qamar Wali; visualization, It Ee Lee, Sohail Khan, and Toqeer Ali Syed; supervision, Qamar Wali and Salman Jan. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: All datasets used in this study are publicly available. Malware samples were sourced from MalwareBazaar, the EMBER dataset, and VirusTotal, while benign executables were collected from open-source repositories such as GitHub and SourceForge. Access to these datasets is governed by the respective providers' usage policies, and all are appropriately cited in the manuscript.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Zaib R, Zhou KQ. Zero-day vulnerabilities: unveiling the threat landscape in network security. Mesopotamian J CyberSecur. 2022;2022:57–64. doi:10.58496/MJCS/2022/007.
2. Iyer KI. From signatures to behavior: evolving strategies for next-generation intrusion detection. European J Adv Eng Technol. 2021;8(6):165–71. doi:10.5281/zenodo.15223001.
3. Akhtar MS, Feng T. Malware analysis and detection using machine learning algorithms. Symmetry. 2022;14(11):2304. doi:10.3390/sym14112304.
4. Gorment NZ, Selamat A, Cheng LK, Krejcar O. Machine learning algorithm for malware detection: taxonomy, current challenges, and future directions. IEEE Access. 2023;11(12):141045–89. doi:10.1109/access.2023.3256979.
5. Syed TA, Nauman M, Khan S, Jan S, Zuhairi MF. ViTDroid: vision transformers for efficient, explainable attention to malicious behavior in android binaries. Sensors. 2024;24(20):6690. doi:10.3390/s24206690.
6. Rombach R, Blattmann A, Lorenz D, Esser P, Ommer B. High-resolution image synthesis with latent diffusion models. In: Proceedings of the 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition. Piscataway, NJ, USA: IEEE; 2022. p. 10684–95.
7. Skoudis E, Zeltser L. Malware: fighting malicious code. Upper Saddle River, NJ, USA: Prentice Hall Professional; 2004.
8. Mohurle S, Patil M. A brief study of wannacry threat: ransomware attack 2017. Int J Adv Res Comput Sci. 2017;8(5):1938–40.
9. Deng X, Mirkovic J. Polymorphic malware behavior through network trace analysis. In: 2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS). Piscataway, NJ, USA: IEEE; 2022. p. 138–46.
10. Campion M, Dalla Preda M, Giacobazzi R. Learning metamorphic malware signatures from samples. J Comput Virol Hack Techn. 2021;17(3):167–83. doi:10.1007/s11416-021-00377-z.
11. Aboaoja FA, Zainal A, Ghaleb FA, Al-Rimy BAS, Eisa TAE, Elnour AAH. Malware detection issues, challenges, and future directions: a survey. Appl Sci. 2022;12(17):8482. doi:10.3390/app12178482.
12. Vishwas G, Nithin NS, Varshith P, Belwal M. Unveiling the world of code obfuscation: a comprehensive survey. In: 2023 7th International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS). Piscataway, NJ, USA: IEEE; 2023. p. 1–8.
13. Afianian A, Niksefat S, Sadeghiyan B, Baptiste D. Malware dynamic analysis evasion techniques: a survey. ACM Comput Surv. 2019;52(6):1–28. doi:10.1145/3365001.

14. Aslan ÖA, Samet R. A comprehensive review on malware detection approaches. IEEE Access. 2020;8:6249–71. doi:10.1109/access.2019.2963724.
15. Yong Wong M, Landen M, Antonakakis M, Blough DM, Redmiles EM, Ahamad M. An inside look into the practice of malware analysis. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. New York, NY, USA: ACM; 2021. p. 3053–69.
16. Nguyen HN, Abri F, Pham V, Chatterjee M, Namin AS, Dang T. MalView: interactive visual analytics for comprehending malware behavior. IEEE Access. 2022;10:99909–30. doi:10.1109/access.2022.3207782.
17. Rana S, Kumar N, Handa A, Shukla SK. Automated windows behavioral tracing for malware analysis. Secur Priv. 2022;5(6):e253. doi:10.1002/spy2.253.
18. Aslan Ö, Ozkan-Okay M, Gupta D. Intelligent behavior-based malware detection system on cloud computing environment. IEEE Access. 2021;9:83252–71. doi:10.1109/access.2021.3087316.
19. Malik MI, Ibrahim A, Hannay P, Sikos LF. Developing resilient cyber-physical systems: a review of state-of-the-art malware detection approaches, gaps, and future directions. Computers. 2023;12(4):79. doi:10.3390/computers12040079.
20. Tariq S, Baruwal Chhetri M, Nepal S, Paris C. Alert fatigue in security operations centres: research challenges and opportunities. ACM Comput Surv. 2025;57(9):1–38. doi:10.1145/3723158.
21. Rao SM, Jain A. Advances in malware analysis and detection in cloud computing environments: a review. Int J Saf Secur Eng. 2024;14(1):225–30. doi:10.18280/ijsse.140122.
22. Stamp M, Alazab M, Shalaginov A. Malware analysis using artificial intelligence and deep learning. 1st ed. Cham, Switzerland: Springer; 2021.
23. Karat G, Kannimoola JM, Nair N, Vazhayil A, S VG, Poornachandran P. CNN-LSTM hybrid model for enhanced malware analysis and detection. Procedia Comput Sci. 2024;233:492–503. doi:10.1016/j.procs.2024.03.239.
24. Li C, Zheng J. API call-based malware classification using recurrent neural networks. J Cyber Secur Mob. 2021;10(3):617–40. doi:10.13052/jcsm2245-1439.1036.
25. Bensaoud A, Kalita J. CNN-LSTM and transfer learning models for malware classification based on opcodes and API calls. Knowl Based Syst. 2024;290(2):111543. doi:10.1016/j.knosys.2024.111543.
26. Ravi V, Alazab M. Attention-based convolutional neural network deep learning approach for robust malware classification. Computat Intell. 2023;39(1):145–68. doi:10.1111/coin.12551.
27. Brown A, Gupta M, Abdelsalam M. Automated machine learning for deep learning based malware detection. Comput Secur. 2024;137(1):103582. doi:10.1016/j.cose.2023.103582.
28. Li T, Luo Y, Wan X, Li Q, Liu Q, Wang R, et al. A malware detection model based on imbalanced heterogeneous graph embeddings. Expert Syst Appl. 2024;246(27):123109. doi:10.1016/j.eswa.2023.123109.
29. Moti Z, Hashemi S, Karimipour H, Dehghantanha A, Jahromi AN, Abdi L, et al. Generative adversarial network to detect unseen internet of things malware. Ad Hoc Netw. 2021;122(2):102591. doi:10.1016/j.adhoc.2021.102591.
30. Taylor T, Eleyan A. Using variational autoencoders to increase the performance of malware classification. In: 2021 International Symposium on Networks, Computers and Communications (ISNCC). Piscataway, NJ, USA: IEEE; 2021. p. 1–6.
31. Kim JY, Cho SB. Obfuscated malware detection using deep generative model based on global/local features. Comput Secur. 2022;112(8):102501. doi:10.1016/j.cose.2021.102501.
32. Bae H, Lee Y, Kim Y, Hwang U, Yoon S, Paek Y. Learn2Evade: learning-based generative model for evading PDF malware classifiers. IEEE Trans Artif Intell. 2021;2(4):299–313.
33. Jan S, Musa S, Ali T, Nauman M, Anwar S, Ali Tanveer T, et al. Integrity verification and behavioral classification of a large dataset applications pertaining smart OS via blockchain and generative models. Expert Syst. 2021;38(4):e12611. doi:10.1111/exsy.12611.
34. Park J, Ji A, Park M, Rahman MS, Oh SE. MalCL: leveraging GAN-based generative replay to combat catastrophic forgetting in malware classification. arXiv:2501.01110. 2025.
35. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, et al. Generative adversarial nets. In: Advances in neural information processing systems. Red Hook, NY, USA: Curran Associates, Inc.; 2014.
36. Kingma DP, Welling M. Auto-encoding variational bayes. arXiv:1312.6114. 2014.

37. Radford A, Metz L, Chintala S. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv:1511.06434. 2015.
38. Dhariwal P, Nichol A. Diffusion models beat GANs on image synthesis. Adv Neural Inform Process Syst. 2021;34:8780–94.
39. Croitoru FA, Hondru V, Ionescu RT, Shah M. Diffusion models in vision: a survey. IEEE Trans Pattern Anal Mach Intell. 2023;45(9):10850–69. doi:10.1109/tpami.2023.3261988.
40. Po R, Yifan W, Golyanik V, Aberman K, Barron JT, Bermano A, et al. State of the art on diffusion models for visual computing. Comput Graph Forum. 2024;43(2):e15063. doi:10.1111/cgf.15063.
41. Whittle S, Iglesias D, Delarosa L, Yasin C. A unified detection framework for ransomware using safeguarded diffusion models. Res Square. 2024. doi:10.21203/rs.3.rs-5083883/v1.
42. Koike S, Tanaka H, Maeda M. Federated learning-based ransomware detection via indicators of compromise. Res Square. 2024. doi:10.21203/rs.3.rs-4585988/v1.
43. Liang H, Yang Y, Jing J. Latent space diffusion model for image dehazing. Appl Soft Comput. 2025;180(3):113322. doi:10.1016/j.asoc.2025.113322.
44. Ajayi B, Barakat B, McGarry K. Adversarial robustness in latent space: a malware classification perspective. Authorea Preprints. 2025. doi:10.36227/techrxiv.174345116.62997761/v1.
45. Van Dao T, Sato H, Kubo M. An attention mechanism for combination of CNN and VAE for image-based malware classification. IEEE Access. 2022;10(3):85127–36. doi:10.1109/access.2022.3198072.
46. Rizvi SKJ, Fraz MM. Robust malware clustering of windows portable executables using ensemble latent representation and distribution modeling. Concurr Comput. 2023;35(8):e7621. doi:10.1002/cpe.7621.
47. Lu Q, Zhang H, Kinawi H, Niu D. Self-attentive models for real-time malware classification. IEEE Access. 2022;10:95970–85. doi:10.1109/access.2022.3202952.
48. Barsha FL, Eberle W. An in-depth review and analysis of mode collapse in generative adversarial networks. Mach Learn. 2025;114(6):141. doi:10.1007/s10994-025-06772-7.
49. Sanjeet V, Inayatullah G, Shib J. Element-wise multiplicative operations in neural architectures: a comprehensive survey of the hadamard product. Authorea Preprints. 2025. doi:10.36227/techrxiv.174680987.78156088/v1.
50. Malware Bazaar; 2023 [cited 2026 Mar 20]. Available from: https://bazaar.abuse.ch/.
51. Anderson HS, Roth P. Ember: an open dataset for training static PE malware machine learning models. arXiv:1804.04637. 2018.