Expert Systems With Applications 298 (2026) 129812

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

GraphShield: Advanced dynamic graph-based malware detection using graph neural networks

Eslam Amer [a,*], Shaker El-Sappagh [b,c], Tamer Abuhamad [c], Bander Ali Saleh Al-Rimy [a], Alaa Mohasseb [a]

[a] PAIDS Research Center, School of Computing, University of Portsmouth, UK
[b] Faculty of Computer Science and Engineering, Galala University, Egypt
[c] College of Computing and Informatics, Sungkyunkwan University, South Korea

ARTICLE INFO

Keywords:
Malware detection
Graph neural networks (GNNs)
API Call analysis
Behavioral graph representation
Dynamic malware analysis
GNN Explainer
Zero-day threat detection
Temporal graph modeling
Deep learning for cybersecurity

ABSTRACT

The rising complexity of modern malware-such as polymorphic, fileless, and sandbox-aware variants-has severely diminished the reliability of conventional detection techniques. Models based on sequential data frequently miss intricate behavioral patterns and long-range dependencies, resulting in poor accuracy and limited adaptability to new threats. This paper introduces GraphShield, a graph-centric behavioral detection framework that identifies malware with high precision by analyzing dynamic API call sequences. GraphShield converts raw API calls into temporal graphs, applies semantic vectorization, and leverages attention mechanisms to extract both localized activity and extended behavioral correlations, directly addressing the weaknesses of earlier systems. We design and assess multiple Graph Neural Network (GNN) variants, including Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), Graph Isomorphism Networks (GINs), and Transformer-based architectures combining convolutional, recurrent, and autoencoding layers. These models capture structural and temporal traits of execution traces using both classification-only and combined classification-reconstruction strategies. To enhance transparency, we incorporate GNN interpretation tools that isolate key API call subgraphs and critical decision pathways, making detection outcomes explainable for analysts. GraphShield is trained on 300,000 balanced instances and tested on a separate 200,000-sample holdout set, achieving over 58 % improvement in accuracy over advanced sequence-driven deep learning models while maintaining a false positive rate under 1 %. Key features include BERT-based API call grouping for reducing dimensionality and a Markov-inspired graph stabilization method for managing graphs of variable length. Our top models attain a 99.5 % F1-score on the test set. GraphShield aligns recent graph learning techniques with operational cybersecurity needs, delivering accurate detection and clear, interpretable results.

1. Introduction

The cybersecurity landscape is evolving rapidly as malware grows in both volume and complexity. Security researchers now detect over 450,000 new malware samples every day Institute (2023). These threats increasingly employ advanced evasion techniques like polymorphism and API call obfuscation to bypass traditional defenses. As a result, signature-based detection systems are becoming less effective, currently missing up to 60 % of zero-day attacks CrowdStrike Intelligence Team (2023). The limitations of static analysis have become increasingly apparent. Modern threats bypass conventional safeguards by leveraging stealth techniques, adaptive behaviors, and legitimate system tools, rendering static code inspection insufficient. This approach struggles with obfuscated code, encrypted payloads, and environment-aware malware that detect analysis tools Afianian et al. (2019).

To address these detection gaps, defenders have shifted toward dynamic behavioral analysis. Unlike static methods, dynamic analysis monitors malware during execution, revealing real-time interactions with the system. AI plays a critical role in this shift Tajudeen and Nureni (2025). Machine learning models-especially deep learning and anomaly detection systems Amer et al. (2024), Amer and Elboghdadly (2024)-analyze behavioral features such as system call sequences, memory access anomalies, and I/O activity to spot suspicious behavior. These AI-driven systems outperform signature-based tools when identifying novel

* Corresponding author.
E-mail address: eslam.amer@port.ac.uk (E. Amer).

https://doi.org/10.1016/j.eswa.2025.129812
Received 22 June 2025; Received in revised form 28 August 2025; Accepted 20 September 2025
Available online 26 September 2025

threats. Rather than match against known patterns, they learn what malicious behavior looks like and generalize to unknown sample Amer et al. (2025).

API call monitoring has emerged as a core component of this behavioral approach. It captures critical actions like file access, memory changes, and network use Chen et al. (2024b). This is vital for detecting tactics such as code injection and persistence mechanisms that evade static scanners. Researchers have focused heavily on extracting features from API call sequences to train ML classification models capable of flagging malicious behavior. These models detect new malware variants by comparing live behavior against stored behavioral patterns.

Dynamic malware detection faces multiple interconnected challenges that significantly impact its effectiveness. At the core of these challenges lies the problem of environment-aware malware, which employs sophisticated techniques to detect and evade analysis environments. These malicious programs systematically check for telltale signs of sandboxing, including low CPU core counts, missing input devices, and irregularities in system timers Costamagna et al. (2018). This cat-and-mouse game has escalated into a full-fledged arms race, with malware developers and sandbox designers continuously innovating to outmaneuver each other Amer (2023), Kirat et al. (2011). Compounding this challenge are advanced temporal evasion techniques that exploit the inherent limitations of dynamic analysis systems. Modern threats, particularly advanced persistent threats (APTs), frequently incorporate extended dormancy periods before activating their payloads CrowdStrike Intelligence Team (2023). This patient approach effectively bypasses detection since most dynamic analysis platforms operate within constrained observation windows. While emerging AI solutions offer potential for predicting such long-term behaviors, significant hurdles remain in scaling these approaches for practical deployment Andriesse et al. (2016). The temporal dimension of malware evasion thus presents a persistent blind spot in current detection methodologies.

The detection challenge grows even more complex when considering the increasing prevalence of behavioral noise in modern attacks. Contemporary malware has mastered the art of camouflage by leveraging living-off-the-land techniques, particularly through the abuse of legitimate system tools like PowerShell and Windows Management Instrumentation (WMI) MITRE Corporation (2023). This strategic blending of malicious activities with normal system operations creates a perfect storm for defenders, especially in enterprise environments where the baseline of legitimate software behavior is inherently diverse and complex Schranko et al. (2023). The resulting signal-to-noise ratio problem makes accurate detection increasingly difficult. These technical challenges manifest concretely in the form of persistent false positives and false negatives that plague dynamic detection systems Amer et al. (2021). The root cause often lies in the fundamental difficulty of distinguishing malicious behavior from benign processes when they share similar characteristics. This discrimination problem is exacerbated by the specialized expertise required to engineer effective behavioral features Zhao et al. (2019b), creating a knowledge gap that hinders the development of robust, generalizable detection indicators. Traditional machine learning approaches frequently prove inadequate when confronted with the nuanced complexity of real-world system behaviors.

The situation is further complicated by the inherent ambiguity and interdependence of API features, which significantly reduce model precision Amer and Zelinka (2020), Amer et al. (2021). Many detection systems struggle because they lack a comprehensive, systemic understanding of malware behavior patterns Ceschin et al. (2023). This knowledge gap becomes particularly apparent when dealing with long, complex API call sequences that increase model perplexity and undermine classification accuracy. The cumulative effect of these challenges creates significant barriers to developing dynamic detection systems that are both accurate and practical for real-world deployment.

1.1. Limitations of API call sequence analysis

API call sequences, while powerful for behavioral detection, face major limitations when analyzing modern malware. Traditional sequence-based models linearize execution flows, causing semantic context loss Shoaib et al. (2022). These models treat API calls as discrete events without capturing how they relate across time or execution branches. For example, malware may establish a connection early and exfiltrate data much later. Sequence models often fail to recognize the connection between these two semantically linked actions Shaukat et al. (2023). Another issue is sequence length. API call traces often span hundreds of thousands of events, creating computational overhead and reducing model effectiveness Chen et al. (2021). Handling variable-length sequences requires complex preprocessing, often resulting in further loss of meaningful semantics.

A deeper problem lies in the semantic gap-the inability to understand what API calls mean at the system level Lagraa et al. (2024). For instance, NtCreateFile followed by NtWriteFile may operate on the same file object, but sequence models treat them as unrelated. This leads to false positives when legitimate applications share similar call patterns, and false negatives when malware uses alternative sequences to achieve malicious outcomes Velickovic et al. (2017). This problem worsens with multi-process malware, where behavior is distributed across processes. Sequence models struggle to preserve causal and semantic relationships across these boundaries Nasereddin and Al-Qassas (2024). Lastly, the high perplexity of API sequences-due to the large number of possible functions and execution paths-makes modeling extremely difficult. The lack of a defined structure or language-like rules limits the model's ability to interpret behavior accurately.

1.2. Graph-Based behavioral analysis

Graph-based representations address these limitations by preserving rich structural relationships inherent in program execution. Temporal graphs modeling API calls as nodes and their relationships as edges maintain both ordering information and contextual relationships Amer et al. (2022), Cui et al. (2023). This dual perspective enables recognizing that two API calls accessing the same registry key are related regardless of temporal separation. Recent studies show graph representations improving detection accuracy by 15–20 % over sequence methods while reducing false positives Yan et al. (2023). The hierarchical nature of graph neural networks (GNNs) provides particular advantages, learning both local call sequences and global resource access patterns spanning entire executions Khoshraftar and An (2024).

Despite their advantages, graph-based approaches introduce new challenges. Computational complexity rises substantially compared to sequence methods, particularly for long-running processes generating massive execution graphs Galli et al. (2024). Memory requirements for complete execution graphs can become prohibitive, necessitating innovative compression and streaming techniques. The dynamic nature of execution traces demands temporal graph models handling evolving structures efficiently Mohanta and Saldanha (2020). These technical hurdles currently limit real-time deployment potential, especially in resource-constrained environments.

Interpretability presents another significant hurdle for graph-based detection. While sequence systems highlight suspicious call subsequences, explaining why particular subgraph structures indicate malice remains challenging Kargarnovin et al. (2024). This "black box" problem proves particularly acute in enterprise environments requiring actionable intelligence beyond binary classifications. Developing explainable graph reasoning techniques without sacrificing accuracy represents a critical research frontier NIST (2023), especially as regulatory pressures increase for transparent security decision-making.

Adversarial attacks specifically targeting graph-based detection loom as a growing concern. Malware authors could manipulate API call

patterns to create benign-looking graph structures while maintaining malicious functionality Demetrio et al. (2021). Defending against such attacks requires robust graph learning focusing on semantic relationships rather than superficial structures. The integration of graph-based analysis with other modalities like memory forensics and network behavior analysis may provide additional resilience MITRE Corporation (2023), creating defense-in-depth against evasion attempts.

The resource-intensive nature of comprehensive dynamic analysis creates practical deployment challenges. Full-system emulation with complete behavioral monitoring can require 10-100x more resources than static analysis Parry et al. (2021). This overhead becomes prohibitive when analyzing thousands of samples daily, forcing security vendors to make difficult trade-offs between analysis depth and throughput Or-Meir et al. (2019). Hybrid approaches combining lightweight static prefiltering with targeted dynamic analysis may offer a viable path forward, though determining optimal analysis depth remains nontrivial.

Fileless malware attacks that execute entirely in memory present unique detection challenges. These attacks leverage legitimate system interfaces while leaving minimal forensic artifacts, making behavioral analysis the primary detection method Kara (2023). However, monitoring memory operations with sufficient granularity while maintaining system performance proves technically demanding Koromilas et al. (2016). The situation worsens as malware increasingly employs sophisticated process injection and API unhooking techniques to evade monitoring.

Machine learning approaches to dynamic analysis face substantial real-world deployment hurdles. Extreme class imbalance between malicious and benign samples, coupled with rapid attack evolution, leads to frequent model degradation Pendlebury et al. (2019). Adversarial attacks specifically designed to evade machine learning detectors are becoming commonplace Pitropakis et al. (2019), requiring continuous model retraining and robust feature engineering. These challenges underscore the need for adaptive detection systems capable of evolving with the threat landscape.

This work addresses several critical challenges through novel graph-based behavioral analysis techniques. Our contributions include: (1) an advanced dynamic graph construction framework capturing both short-term and long-range behavioral dependencies, (2) a hierarchical GNN architecture combining attention mechanisms with novel edge-weighting approaches, and (3) comprehensive evaluation demonstrating over 58 % detection improvement over state-of-the-art models while maintaining sub-1 % false positive rates Anderson (2023), Virustotal (2023). The subsequent sections detail our methodology and experimental validation of these advancements.

1.3. Contributions

This work presents a novel framework for dynamic malware detection by leveraging graph-based representations of API call behavior. The principal contributions are summarized as follows:

- Proposing a context-aware API graph construction technique that captures structural malware patterns by modeling API call sequences with embeddings, attention weights, and transition frequencies, overcoming limitations of conventional detection methods.
- Developing a dimensionality reduction approach for API calls by clustering them based on semantic similarity, using BERT embeddings and attention scores to form compact yet meaningful groups while preserving behavioral semantics.
- Introducing a sequence stabilization method that converts API calls into transitions between learned cluster states, generating fixed-size interpretable graphs that represent malicious behavior as Markov processes for efficient inference.
- Designing hybrid GNN architectures including GIN-CNN and GCN-LSTM autoencoders combined with Graph Attention Networks (GAT), Graph Convolutional Networks (GCN), and Transformer-based Graph Autoencoders to integrate neighborhood aggregation, temporal modeling, and spatial abstraction for robust detection.
- Demonstrating extensive experimental results showing an over 58 % accuracy improvement over state-of-the-art techniques, maintaining sub-1 % false positives on large-scale datasets, and outperforming baseline deep learning models in generalization and stability.
- Implementing a GNN explainer that identifies the key subgraph structures and node features driving each GNN model prediction. It produces human-understandable explanations.

This paper is organized to guide the reader through our research systematically. It begins with Section 2, which reviews the background and related work in the field. Section 3 introduces our proposed GraphShield model and methodology. In Section 4, we describe the experimental setup, including datasets and evaluation metrics, and present the empirical results. The robustness and generality of GraphShield are examined in Section 5. Section 6 focuses on the application and analysis of the GNN explainer model. We discuss the operational impact and deployment value of our approach in Section 7. Section 8 addresses current limitations and outlines future research directions. Finally, Section 9 summarizes our contributions and key findings.

2. Related work

The transition to graph-based approaches marked a conceptual breakthrough in malware analysis. Li et al. (2022a) demonstrated that structural graph representations could capture invariant behavioral patterns across malware variants, while Zhang et al. (2022) proved their superiority over sequential models in preserving execution context. These theoretical advances were operationalized by Bilot et al. (2024), whose work on fileless malware detection showcased the practical advantages of graph-based methods against sophisticated, real-world threats. The following sections examine these techniques, their graph representations, neural models, and the challenges they face.

2.1. Graph-Based approaches for malware detection

The rapid evolution of malware has rendered traditional signature-based detection methods increasingly ineffective. Modern malware employs sophisticated evasion techniques such as polymorphism, metamorphism, and code obfuscation, necessitating more advanced detection approaches Bilot et al. (2024). To address this challenge, researchers have proposed graph-based methods that represent programs as graphs, where nodes correspond to system entities (e.g., API calls, processes, registry keys) and edges capture their interactions. These techniques model the complex behavioral patterns characteristic of malicious software Amer and Zelinka (2020), offering the advantage of focusing on behavioral characteristics rather than static signatures Amer (2023), Amer et al. (2022).

Unlike traditional models that treat API calls or system events as sequences, graph-based approaches maintain the structural context of interactions. This structural modeling enables the detection of advanced threats such as fileless malware and Advanced Persistent Threats (APTs) that rely on multi-stage execution strategies Bilot et al. (2024). In addition, graph-based systems can integrate both static and dynamic features, resulting in a more holistic behavioral profile Zhang et al. (2022).

2.2. Graph representations in malware analysis

Building on the concept of behavior modeling, various graph representations have been proposed to capture distinct aspects of program execution. API call graphs are widely used, where nodes denote API functions (e.g., `CreateProcess`, `RegSetValue`, `SocketConnect`) and edges reflect call sequences or dependencies Zhang et al. (2022). These graphs capture characteristic malicious patterns, but face several challenges: semantic gaps between temporally distant calls, obfuscation via indirect

or dynamically resolved calls Bilot et al. (2024), and high computational overhead from large, sparse graphs.

To complement behavioral insights from API calls, control flow graphs (CFGs) extracted from binaries reveal all potential execution paths Artuso et al. (2024). However, CFGs are susceptible to control flow obfuscation, undecidable branching, and disassembly issues in packed or encrypted binaries Bilot et al. (2024). Data dependency graphs (DDGs) address data flow analysis by tracing variable interactions and tainted data propagation Bilot et al. (2024), though they introduce heavy memory costs and sensitivity to compiler transformations Zhang et al. (2022).

Expanding further, researchers have developed heterogeneous graphs that integrate diverse system entities Artuso et al. (2024), capturing multi-dimensional malware behaviors. These complex structures require entity alignment and semantic fusion across domains Li et al. (2024). Temporal graphs build upon this by encoding event timestamps to detect time-sensitive patterns Li et al. (2024), but they face logging overhead, stream synchronization, and high false positive rates from benign periodic behaviors Li et al. (2022a), Zhang et al. (2022).

2.3. Graph neural networks for malware detection

To analyze the rich structural information in these graph representations, researchers have increasingly turned to Graph Neural Networks (GNNs), which are tailored for learning from graph-structured data Wu et al. (2020).

Among GNN variants, Graph Convolutional Networks (GCNs) are commonly applied to API call graphs and system behavior networks. Zhang et al. (2022) and Li et al. (2022b) demonstrated that GCNs can effectively learn patterns like process hollowing (`CreateProcessA/W` → `ZwUnmapViewOfSection` → `WriteProcessMemory` → `ResumeThread`) and ransomware behaviors (`FindFirstFileExW` → `CryptGenKey` → `CryptEncrypt` → `DeleteFileW`).

Despite achieving high accuracy (>96 %), Graph Convolutional Networks (GCNs) exhibit several limitations in malware detection applications. First, they suffer from over-smoothing in deeper network architectures Li et al. (2018). Second, their static nature limits adaptability to dynamic malware behaviors Zhang et al. (2022). Third, they remain vulnerable to adversarial attacks through API call injection Zhang and Zitnik (2020). Additionally, their limited receptive field hinders detection of long-range dependencies - particularly in cross-process coordination scenarios Xu et al. (2019). To address these challenges, researchers have proposed hybrid GCN architectures incorporating attention mechanisms Rossi et al. (2020) and adversarial training approaches Sun et al. (2022).

To improve upon GCNs, Graph Attention Networks (GATs) introduce attention mechanisms that assign varying importance to neighboring nodes during aggregation Veličković et al. (2018). This adaptability is well-suited for malware detection, as shown by Chen et al. (2024a), who demonstrated that GATs could emphasize critical API sequences while downplaying irrelevant system calls. Liu et al. (2022) demonstrated the effectiveness of Graph Attention Networks (GATs) for detecting fileless malware, particularly in identifying critical attack patterns like process injection sequences (`VirtualAllocEx` → `WriteProcessMemory` → `CreateRemoteThread`) Chen et al. (2023). While GATs provide valuable interpretability through their attention weight mechanisms Liu et al. (2021), they present two notable challenges: (1) increased computational requirements leading to longer training times, and (2) persistent vulnerability to adversarial API manipulation attacks Feng et al. (2021). Recent research has addressed these limitations through several innovations, including sparse attention techniques to improve efficiency Ranjani and Chinnadurai (2024) and hybrid temporal-attention architectures that better balance detection accuracy with computational performance Duan et al. (2024).

Pushing the theoretical limits of graph discrimination, Graph Isomorphism Networks (GINs) substantially outperform conventional GNNs by virtue of their alignment with the Weisfeiler-Lehman isomorphism test Xu et al. (2019). These networks demonstrate particular efficacy in identifying polymorphic malware, successfully preserving essential malicious patterns through injective aggregation even when confronted with high structural variation Bouritsas et al. (2022). Empirical evidence confirms that GINs can detect obfuscated variants exhibiting over 80 % structural diversity Gao et al. (2022), effectively countering sophisticated evasion techniques including call graph manipulation and control flow flattening Kim and Cho (2022). However, these capabilities come with notable computational demands, as GINs require significant memory overhead and demonstrate heightened sensitivity to hyperparameter selection Bai et al. (2021), Zhao et al. (2019a). Recent advances have addressed these limitations through two primary approaches: optimized architectures like pruned GINs for enhanced efficiency Gurevin et al. (2024), and innovative hybrids combining temporal and attention mechanisms to maintain robustness while improving performance Wang et al. (2025).

Building upon structural graph representations, hierarchical modeling approaches enable multi-scale analysis of malware behavior by simultaneously capturing both local and global patterns. The MalGraph framework Ling et al. (2022) exemplifies this approach through its hierarchical pooling mechanism, which proves particularly effective for detecting sophisticated multi-stage attacks operating across different system layers. While demonstrating strong detection capabilities, these hierarchical models incur substantial memory overhead (40–60 % increase) and face the inherent challenge of potentially obscuring localized anomalies during the abstraction process Filippo et al. (2020).

Complementing these spatial approaches, temporal graph models address the dynamic nature of malware through techniques like Dynamic Graph Embedding (DGE) Manzoor et al. (2016), which specifically tracks the progression of advanced threats, including ransomware and APTs. However, temporal models struggle with maintaining behavioral context over extended periods, leading to reduced detection accuracy for malware employing long latency periods Liu et al. (2023). Furthermore, the high-frequency sampling required for precise temporal analysis imposes significant storage demands that prove particularly challenging in enterprise-scale deployments Bilot et al. (2024). Recent hybrid architectures have emerged to address these limitations by strategically combining temporal and hierarchical methods Duan et al. (2024). These integrated approaches achieve superior detection of polymorphic threats while implementing adaptive sampling techniques to maintain computational efficiency, effectively balancing the trade-offs between detection accuracy and system performance.

2.4. Limitations of existing graph-Based malware detection approaches

While graph-based approaches have demonstrated superior performance compared to traditional detection methods, a closer examination reveals several persistent challenges that limit their practical effectiveness. These limitations manifest across multiple dimensions of system performance and usability.

The computational demands of current approaches present the first major hurdle. The process of transforming sequential API logs into graph representations, while powerful, imposes significant memory and processing requirements, particularly problematic when analyzing the extended execution traces characteristic of modern stealth malware Bilot et al. (2024). This challenge becomes even more pronounced in hierarchical graph models, where the benefits of multi-scale behavior analysis come at the cost of 40–60 % increased memory overhead Ling et al. (2022). Similar scalability issues plague temporal graph approaches when modeling prolonged attack sequences Liu et al. (2023). Compounding these computational challenges is the vulnerability to adversarial manipulation, where attackers can effectively evade detection through strategic injection of benign API calls or sophisticated call sequence obfuscation Zhang and Zitnik (2020)-a weakness rooted in current systems' over-reliance on potentially superficial structural patterns rather than deeper semantic relationships.

Beyond computational and robustness concerns, the temporal dimension of malware behavior presents its own set of complications. Although recent work has begun incorporating time-aware modeling techniques Rossi et al. (2020), these implementations frequently fail to maintain contextual connections across extended execution windows. This limitation becomes particularly apparent when analyzing advanced threats that strategically space their malicious activities, as the system may miss critical relationships between temporally separated but logically connected behaviors Duan et al. (2024). The situation grows more complex when considering heterogeneous graphs that incorporate multiple system entities (processes, registry keys, etc.). While theoretically offering richer analysis capabilities, these models introduce alignment complexities that can paradoxically obscure the very discriminative patterns they aim to highlight Artuso et al. (2024). These technical challenges translate directly to operational difficulties in real-world deployment, where synchronization errors in dynamic logging systems and the inherent noise of benign background activities combine to generate problematic false positive rates Zhang et al. (2022).

Building on this understanding of current limitations, our GraphShield framework introduces a suite of innovations designed to address these challenges holistically. At the foundation of our approach lies a novel BERT-based semantic clustering technique that simultaneously reduces API call dimensionality while preserving the behavioral semantics crucial for accurate detection. This innovation works in concert with our Markovian graph stabilization method, which elegantly solves the variable-length execution trace problem by transforming these sequences into fixed-size transition matrices, maintaining analytical precision while dramatically improving processing efficiency.

Perhaps most critically, GraphShield's temporal modeling capabilities represent a significant advance over previous approaches. Our transition matrix formulation maintains long-range behavioral dependencies by conceptualizing API call sequences as state transitions in a Markov process-a representation that naturally bridges the gap between localized behavior analysis and global execution pattern recognition. This theoretical innovation is complemented by practical considerations in the form of our integrated GNN Explainer module, which directly addresses the "black box" problem that has long plagued deep learning-based detection systems. Through providing security analysts with clear visualizations of the influential subgraphs and node features driving each detection decision, the framework enables both greater operational transparency and more effective threat intelligence development.

3. Proposed model

The objective is to generate behavioral feature graphs based on API call sequences to capture the intrinsic processes of both malware and goodware. These graphs serve as a foundation for identifying new processes as malicious or benign by analyzing API calls. Since malware often follows distinct execution patterns to evade detection or exploit vulnerabilities, tracking API call sequences provides a more dynamic and resilient approach to threat detection. Unlike static analysis, this method can identify polymorphic and zero-day threats by focusing on behavioral anomalies rather than code signatures. In the following, we discuss the main phases of the proposed model in details.

3.1. Phase 1: Preprocessing and feature generation

The goal is to prepare a focused dataset that captures only relevant API interaction data, eliminating noise and irrelevant patterns. This ensures the dataset reflects true API behavior, strengthens the detection of meaningful calling sequences, and forms a reliable foundation for tasks like anomaly detection and modeling. Upon focusing, the dataset becomes more efficient, representing actual usage patterns and improving the precision of future analysis.

3.1.1. Data preprocessing

The preprocessing step was essential for preparing the dataset for analysis. We removed duplicate API call sequences to remove redundancy and prevent biased results. Next, we excluded sequences with fewer than 20 APIs, as they were too short to offer valuable insights Amer (2023), Owoh et al. (2024). This cleaning step ensures the dataset focuses on more relevant patterns Dabas et al. (2023). The result is a refined dataset of distinct malware and goodware API call sequences.

3.1.2. Sequence analysis

In malware analysis, sequences of API functions exhibit recurring contextual patterns tied to malicious behavior, often persisting across malware families and revealing stable relationships between API calls Amer and Zelinka (2020). Traditional techniques-such as N-grams, Hidden Markov Models (HMMs), Recurrent Neural Networks (RNNs), and Convolutional Neural Networks (CNNs)-face two key limitations: an inability to model long-range dependencies and computational inefficiency Demirkıran et al. (2022). To address these issues, we employed BERT to generate token embeddings and attention scores for API call bigrams Dabas et al. (2023), Demirkıran et al. (2022). According to recent studies, BERT's contextual embeddings achieve higher accuracy than static embeddings like Word2Vec, as its bidirectional attention mechanism dynamically captures nuanced API call dependencies, including long-range semantic relationships and position-aware patterns that traditional methods miss Fields et al. (2024). Unlike Word2Vec, which generates fixed embeddings regardless of context, BERT adapts representations based on surrounding API calls, enabling it to distinguish between benign and malicious sequences with similar call sets but differing execution contexts. Moreover, even in resource-limited environments, fine-tuned transformer models consistently outperform traditional methods, including Term Frequency-Inverse Document Frequency (TF-IDF) and Word2Vec baselines Abd-Elaziz et al. (2025), Abdelmotaleb et al. (2025). TF-IDF, while computationally lightweight, fails to model call order or semantic meaning, reducing its effectiveness for behavioral analysis. In contrast, BERT's ability to weigh critical API transitions via self-attention allows it to prioritize security-relevant sequences (e.g., repeated file writes followed by registry modifications) that simpler statistical approaches overlook.

Through computing both embeddings and attention scores, BERT effectively captures the semantic meaning of individual API calls and their interrelationships. Embeddings convert raw API call data into a model-friendly format, providing meaningful representations of each token. Meanwhile, attention scores identify the most relevant tokens in the sequence, allowing the model to focus on critical parts of the data regardless of their position. This dual approach ensures the model understands both the context of each API call and its relationship with others. Using bigrams-pairs of consecutive tokens-strikes a balance between capturing local dependencies and maintaining computational efficiency, enabling the model to identify malicious behavior patterns without the complexity and high computational cost of larger sequences like trigrams.

BERT uses its attention mechanism to capture long-term dependencies in API call sequences by focusing on relevant tokens, regardless of their position. This enables effective modeling of complex, temporal relationships within the data. However, BERT's maximum context length L poses a challenge when processing long sequences Tay et al. (2022). To address this limitation, we introduce an overlap-stretched chunking algorithm, which efficiently handles lengthy sequences while preserving critical contextual information (Algorithm 1).

Sequential relationships between API calls often provide deeper insights into malicious intent than individual calls alone. For instance, our analysis of the `VirtualAlloc` → `WriteProcessMemory` bigram demonstrates this phenomenon clearly. VirtualAlloc [1] and WritePro-

[1] Microsoft Documentation: VirtualAlloc Documentation.

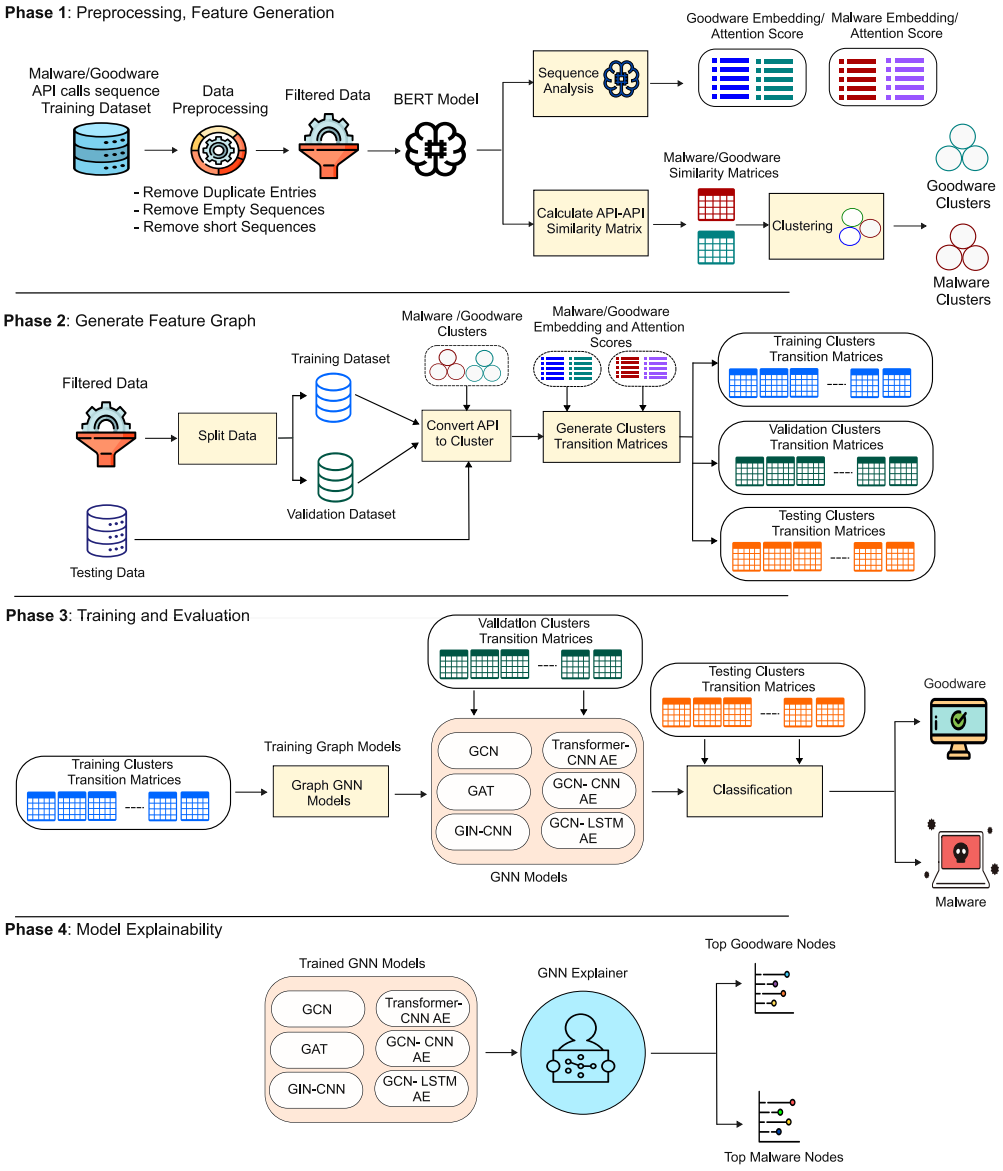


Fig. 1. The Proposed GraphShield Model.

cessMemory[2] are two crucial Windows API functions commonly used in process manipulation and memory management. They are particularly important in process injection techniques Dowd et al. (2006).

While this sequence appears only once in goodware samples—with an embedding score of 0.796 and an attention score of 0.078—it occurs twelve times more frequently in malware, exhibiting stronger average embedding (0.855) and attention (0.082) scores. The single benign occurrence, with its modest embedding and low attention score, suggests that these calls rarely interact significantly in non-malicious software. In contrast, malware samples consistently show higher embedding scores (0.855), indicating a stronger semantic relationship between these memory operations. Although the attention scores appear numerically close (0.082 vs. 0.078), their consistency across multiple malicious samples provides a meaningful distinction absent in goodware.

Given an input sequence $S = [w_1, w_2, \ldots, w_N]$ where $N > L$, we partition it into overlapping chunks $\{C_k\}_{k=0}^{K}$ using a stride s:

$$C_k = [w_{k\cdot s+1}, \ldots, w_{\min(k\cdot s+L,N)}], \quad K = \left\lfloor \frac{N-L}{s} \right\rfloor. \tag{1}$$

The stride s is chosen such that $s < L$, ensuring an overlap of $L - s$ tokens between consecutive chunks. This overlap prevents information loss at chunk boundaries while maintaining computational efficiency. For each chunk C_k, we perform a forward pass through the BERT transformer model to obtain:

- Token embeddings $H_k \in \mathbb{R}^{L\times d}$, where d is the hidden dimension.
- Attention weights $\{A_k(t)\}_{t=1}^{T}$, where T is the number of transformer layers.

Since tokens appear in multiple chunks due to overlap, their representations need to be aggregated. We employ *max-pooling* to extract the most salient contextual features. To compute the similarity score of token w_i, we use cosine similarity between its embedding and the mean embedding of its chunk:

$$\text{sim}(w_i) = \max_{k|w_i\in C_k} \left(\frac{\mathbf{H}_k[j_i] \cdot \bar{\mathbf{h}}_k}{|\mathbf{H}_k[j_i]||\bar{\mathbf{h}}_k|} \right), \tag{2}$$

where the mean embedding of chunk C_k is:

$$\bar{\mathbf{h}}_k = \frac{1}{|C_k|} \sum_{j=1}^{|C_k|} \mathbf{H}_k[j]. \tag{3}$$

[2] Microsoft Documentation: WriteProcessMemory Documentation

Algorithm 1 Embedding and Attention Extraction.

Input: API call sequence $S = [w_1, w_2, \ldots, w_N]$, maximum length $L = 512$, stride $s = 256$
Output: Token embeddings $\{\text{sim}(w_i)\}$, attention scores $\{\text{attn}(w_i)\}$

1. Chunk Partitioning
Partition S into overlapping chunks $\{C_k\}_{k=0}^{K}$ using Equation (1).

2. Chunk Embedding and Attention Extraction
for each chunk C_k **do**
  Encode C_k with BERT to obtain token embeddings H_k and attention weights $\{A_k(t)\}$.
end for

3. Token-level Feature Computation
for each token w_i **do**
  Identify all chunks C_k containing w_i.
  for each chunk C_k containing w_i **do**
    Compute chunk mean embedding $\bar{\mathbf{h}}_k$ using Equation (3).
    Compute similarity score using Equation (2).
    Compute attention score using Equation (4).
  end for
end for

4. Max-pooling Across Chunks
for each token w_i **do**
  Apply max-pooling over all similarity scores and attention scores obtained from chunks containing w_i.
end for

return $\{\text{sim}(w_i)\}$, $\{\text{attn}(w_i)\}$

For attention aggregation, we compute the average attention score across all transformer layers and apply max-pooling:

$$\text{attn}(w_i) = \max_{k|w_i \in C_k} \left(\frac{1}{T} \sum_{t=1}^{T} \mathbf{A}_k^{(t)}[0, j_i] \right), \tag{4}$$

where j_i represents the position of w_i in chunk C_k.

The chunking and pooling strategy preserves the most relevant contextual features for each token across all its occurrences. Overlapping segments reduce boundary effects, while max-pooling distills the most informative representations. This method enhances robustness in processing long sequences while maintaining computational efficiency. Ultimately, we derive per-sequence embeddings and attention weights for every bigram in both malicious and benign API call traces.

3.1.3. API Clustering

The extensive number of APIs in our analysis necessitates dimensionality reduction to extract a manageable yet representative set of sequence patterns. Clustering API call sequences effectively organizes and interprets API interactions, particularly for distinguishing between malware and goodware behaviors Amer and Zelinka (2020). Our core observation reveals that while both software types may invoke overlapping APIs, they frequently exhibit distinct calling relationships and contextual associations. Therefore, grouping similar API calls according to their interaction patterns within sequences reduces computational complexity while enhancing the model's ability to discern malicious behavior. This approach additionally facilitates identification of usage trends, call dependencies, and behavioral nuances-critical factors in differentiating benign from malicious activity.

To implement this approach, we computed pairwise similarity scores between APIs in malware and goodware training samples, generating two $n \times n$ similarity matrices, where n is the number of APIs. Our primary objective focused on determining the optimal number of clusters for indexing individual APIs. For this purpose, we employed the elbow method, a well-established technique for evaluating the trade-off between within-cluster variance and cluster granularity Sammouda and El-Zaart (2021). This technique computes the total within-cluster sum of squares (WSS) through the Eq. (5).

$$WSS_k = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2 \tag{5}$$

Here k represents the number of clusters, C_i denotes individual clusters, x represents cluster endpoints, and μ_i corresponds to cluster centroids. To improve clustering efficiency and avoid suboptimal local minima, we utilized KMeans++ for centroid initialization. This variant of the K-means algorithm enhances convergence speed and cluster quality through strategic selection of initial centroids to maximize separation Arthur and Vassilvitskii (2007). The algorithm's centroid initialization improved the stability and accuracy of the clusters, revealing both malware-specific API chains and common usage sequences across software types. This process ultimately produced two distinct cluster groups: one characterizing malware behavior, the other representing goodware patterns. Based on the elbow method analysis, 20 clusters emerged as the optimal number for indexing APIs in both malware and goodware datasets.

3.2. Phase 2: Generate feature graph

The GraphShield model transforms raw API call sequences into structured graph representations for malware detection. This transformation is performed in three core stages. First, raw API calls are mapped to semantic clusters $\{S_1, \ldots, S_n\}$, which represent functionally coherent groups. This converts sequences of varying lengths into fixed-dimensional cluster trajectories. Each trajectory is then partitioned into training, validation, and testing sets. The testing set remains strictly isolated to preserve evaluation integrity.

In the second stage, cluster sequences are processed using Algorithm 2 to construct transition matrices based on a Markov model. In this framework:

- Each cluster S_i represents a state in a finite state space $S = \{S_1, \ldots, S_n\}$
- State transitions are governed by a composite scoring function $\psi(a, b)$
- The function $\psi(a, b)$ integrates three distinct API-level features for transitions between clusters a and b

The feature components are:

1. **Embedding similarity**:
   $e(i, j) = \mathbf{w}^\top [\mathbf{e}_i \oplus \mathbf{e}_j]$
   where $\mathbf{e}_i, \mathbf{e}_j$ are API call embeddings and $\mathbf{w}$ is a learnable weight vector.
2. **Attention weights**:
   $\alpha_{i \to j}$
   derived from transformer self-attention mechanisms.
3. **Normalized transition frequency**:
   $f(i, j) = \frac{\#(i \to j)}{\sum_k \#(i \to k)}$
   representing the empirical probability of transition between API calls.

For cluster transitions $a \to b$, we define $T_{a \to b} = \{(i, j) \mid i \in a, j \in b\}$ as the set of all API-level transitions between clusters a and b. The composite transition score $\psi(a, b)$ combines three feature components-embedding similarity, attention scores, and transition frequency-

through a convex combination with learnable parameters $\lambda_1, \lambda_2, \lambda_3$, constrained by $(\lambda_1 + \lambda_2 + \lambda_3 = 1)$. The score is defined as:

$$\psi(a,b) = \frac{1}{|T_{a\to b}|} \sum_{(i,j)\in T_{a\to b}} \Big(\underbrace{\lambda_1 e(i,j)}_{\text{embedding}} + \underbrace{\lambda_2 \alpha_{i\to j}}_{\text{attention}} + \underbrace{\lambda_3 f(i,j)}_{\text{frequency}} \Big) \tag{6}$$

where $e(i,j)$ is the embedding similarity between API call embeddings i and j, $\alpha_{i\to j}$ is the attention score obtained from BERT's transformer layers, and $f(i,j)$ is the normalized empirical frequency of the transition $i \to j$. The weights $\lambda_1, \lambda_2, \lambda_3$ control the relative influence of these features in modeling transition likelihoods. These weights are not fixed or manually tuned. Instead, they are treated as learnable parameters and trained end-to-end with the rest of the model via gradient descent. At initialization, the weights are set equally:

$$\lambda_1 = \lambda_2 = \lambda_3 = \frac{1}{3}$$

During training, the model optimizes these parameters using backpropagation. To enforce the constraints $\lambda_1 + \lambda_2 + \lambda_3 = 1$ and $\lambda_i \geq 0$, the weights are parameterized by underlying logits and passed through a softmax function. This design allows the model to adaptively adjust the contribution of each component based on their effectiveness for classification. For example, if attention scores provide stronger signals for distinguishing malicious behavior than frequency or embedding similarity, the model will assign a higher λ_2.

The resulting score $\psi(a,b)$ is normalized by applying a softmax over all possible transitions from cluster a, producing a probabilistic transition matrix that forms the input graph structure for the Graph Neural Network classifier.

$$P_{a\to b} = \frac{\exp(\psi(a,b))}{\sum_{b'} \exp(\psi(a,b'))} \tag{7}$$

where $P_{a\to b}$ denotes the transition probability between clusters a and b. The learnable λ-weights allow GraphShield to adapt to diverse malware behaviors and highlight the most important signals in each scenario. The resulting graphs, generated by Algorithm 2, encode structural, semantic, and statistical patterns in the process behavior. These representations serve as the direct input for downstream graph neural network models, enabling the classifier to learn from both local execution patterns and global behavioral context. Specifically, this approach achieves three key outcomes:

1. **Standardizes variable-length API traces into fixed-size matrices** by representing each semantic cluster as a node and each inter-cluster transition as a weighted, normalized edge in an $n \times n$ matrix. This eliminates instability from fluctuating graph sizes and degree distributions, facilitating consistent batch processing and efficient GNN training.
2. **Captures semantic and temporal relationships** through the composite scoring function in Eq. (6), which integrates three distinct and complementary signals—BERT-based embedding similarity, transformer attention scores, and empirical transition frequencies—combined via learnable λ-weights. This ensures that edge weights reflect both the meaning of API calls and their behavioral dynamics over time.
3. **Quantifies behavioral patterns with probabilistic graphs** by applying a softmax normalization over outgoing transitions (Eq. (7)), yielding a Markovian transition matrix where each row encodes a probability distribution over next states. This probabilistic formulation allows the model to naturally model uncertainty, emphasize dominant malicious transitions, and maintain comparability across samples.

These design choices allow GraphShield to preserve critical execution semantics, stabilize input dimensions for deep learning architectures, and highlight the most discriminative behavioral signals for malware detection.

Algorithm 2 Generation of Feature Matrices for Malware and Goodware.

Input:
  Malware sequences S_{mal}, Goodware sequences S_{gw}
  Cluster count N (default: 100)
  Embedding vectors $\{\mathbf{e}_i\}$, attention weights $\{\alpha_{i\to j}\}$, frequencies $\{\#(i \to j)\}$
  Learnable weights $\lambda_1, \lambda_2, \lambda_3$ for Eq. (6)
Output:
  Feature matrices $\{M_i\}_{i=1}^{|S_{\text{mal}}|}$ for malware
  Feature matrices $\{G_i\}_{i=1}^{|S_{\text{gw}}|}$ for goodware
for all (S, label) in $\{(S_{\text{mal}}, \text{mal}), (S_{\text{gw}}, \text{gw})\}$ **do**
  for $i = 1$ to $|S|$ **do**
    $C \leftarrow$ parsed cluster sequence from $S[i]$ {Map API calls to cluster indices $[1, N]$}
    Initialize $F \leftarrow \mathbf{0}_{N\times N}$
    for $j = 1$ to $|C| - 1$ **do**
      $c_1 \leftarrow C[j]$, $c_2 \leftarrow C[j+1]$
      Compute $\psi(c_1, c_2)$ via Eq. (6) {Using $T_{c_1\to c_2}$}
      Compute $P_{c_1\to c_2}$ via Eq. (7)
      $F[c_1, c_2] \leftarrow P_{c_1\to c_2}$
    end for
    $F \leftarrow$ normalize(F) {Ensure rows sum to 1}
    if label = mal **then**
      $M_i \leftarrow F$
    else
      $G_i \leftarrow F$
    end if
  end for
end for

return $\{M_i\}, \{G_i\}$

3.3. Phase 3: Training and evaluation

We used multiple graph-based deep learning models to analyze cluster transition matrices for malware and goodware classification, each using distinct strengths in feature extraction and classification. These models exploit graph structures to capture intricate relationships within the data, enabling robust discrimination between malicious and benign software behaviors. Our framework begins with the Graph Convolutional Network (GCN), which learns node representations through spectral-based neighborhood aggregation. The layer-wise propagation rule is defined as:

$$H^{(l+1)} = \sigma\big(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\big), \tag{8}$$

where $\tilde{A} = A + I$ represents the adjacency matrix with added self-loops, $\tilde{D}$ is the corresponding degree matrix, and $W^{(l)}$ contains trainable parameters. While effective for capturing topological patterns, the GCN's fixed-weight aggregation motivates the need for more adaptive approaches. Extending beyond spectral methods, the Graph Attention Network (GAT) introduces dynamic neighborhood weighting through attention mechanisms. The node features are updated as:

$$H_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij} W H_j^{(l)}\right), \tag{9}$$

with attention coefficients computed as:

$$\alpha_{ij} = \text{softmax}_j\Big(\text{LeakyReLU}\Big(\mathbf{a}^T[W H_i^{(l)} \| W H_j^{(l)}]\Big)\Big). \tag{10}$$

This attention-based paradigm provides two key advantages over GCN: adaptive importance weighting for different neighbors, and implicit handling of edge importance without requiring modified adjacency matrices. For maximum discriminative power in graph separation tasks,

Table 1
Architectural Overview of the Employed Graph Neural Network Models.

Model Name	Encoder Type	Decoder Type	Layers	Activation	Pooling	Final Classification
GCN Model	GCNConv (Graph Convolutional Network)	None	3 GCNConv layers: 100→128→64→32, 1 FC layer	ReLU, BatchNorm, Dropout	Global mean pooling	Linear layer (32→1), output squeezed
GAT Model	GATConv (Graph Attention Network) with multi-head attention	None	3 GATConv layers with heads: (100→128x8), (1024→64x8), (512→32), 1 FC layer	ReLU, BatchNorm, Dropout	Global mean pooling	Linear layer (32→1), output squeezed
GIN-CNN Autoencoder	GINConv (Graph Isomorphism Network)	CNN (1D Conv)	3 GINConv layers with MLPs: 100→128→64→32; Decoder: 32→64→128→100	ReLU, BatchNorm, Dropout	Global mean pooling	Linear layer (32→1), outputs: classification + reconstruction
Transformer-CNN Autoencoder	Transformer Conv (Graph Transformer Network)	CNN (1D Conv)	3 TransformerConv layers: (100→128x4), (512→64x4), (256→32); Decoder: 32→64→128→100	ReLU, BatchNorm, Dropout	Global mean pooling	Linear layer (32→1), outputs: classification + reconstruction
GCN-CNN Autoencoder	GCNConv (Graph Convolutional Network)	CNN (1D Conv)	3 GCNConv layers: 100→128→64→32; Decoder: 32→64→128→100	ReLU, BatchNorm, Dropout	Global mean pooling	Linear layer (32→1), outputs: classification + reconstruction
GCN-LSTM Autoencoder	GCNConv (Graph Convolutional Network)	LSTM (2-layer)	3 GCNConv layers: 100→128→64→32; Decoder LSTM input: 32, output: 64, followed by FC (64→100)	ReLU, BatchNorm, Dropout	Global mean pooling	Linear layer (32→1), outputs: classification + reconstruction

we employ the Graph Isomorphism Network (GIN) with its theoretically powerful injective aggregation:

$$H_v^{(l+1)} = \mathrm{MLP}^{(l)}\left((1+\epsilon^{(l)})\cdot H_v^{(l)} + \sum_{u\in\mathcal{N}(v)} H_u^{(l)}\right), \tag{11}$$

where $\epsilon^{(l)}$ is a learnable parameter. The GIN's sum aggregation preserves distinct neighborhood distributions, making it particularly suitable for malware graphs where subtle structural differences are critical. Combining the strengths of attention mechanisms and convolutional processing, our Transformer-based Graph Autoencoder employs graph attention layers:

$$H^{(l+1)} = \mathrm{LayerNorm}\left(H^{(l)} + \mathrm{MultiHeadAttention}(H^{(l)}, H^{(l)}, H^{(l)})\right), \tag{12}$$

followed by position-wise feedforward networks. The CNN decoder then reconstructs the graph representation through hierarchical convolutions:

$$X' = \mathrm{ConvBlock}(\mathrm{ReLU}(\mathrm{Conv2D}(H))), \tag{13}$$

where ConvBlock denotes a sequence of convolution, batch normalization, and activation operations.

For capturing evolving malware behaviors, we integrate graph and sequence processing through two hybrid architectures. The GCN-LSTM Autoencoder combines GCN's spatial aggregation with LSTM's temporal modeling:

$$h_t = \mathrm{LSTM}(\mathrm{GCN}(X_t), h_{t-1}), \tag{14}$$

while the GCN-CNN Autoencoder fuses graph and grid-based processing:

$$X' = \mathrm{CNN\text{-}Decoder}(\mathrm{GCN\text{-}Encoder}(X)). \tag{15}$$

These models, as summarized in Table 1, collectively provide complementary perspectives on malware graph analysis - from spectral and attention-based methods to temporal and spatial hybrids. The GCN establishes baseline neighborhood aggregation, which GAT extends through attention mechanisms, while GIN provides theoretically grounded separation power. The transformer and hybrid models then integrate these graph operations with sequence and grid processing for comprehensive feature learning.

Classification Architecture: The graph-level representation is obtained by applying global mean pooling to the node embeddings generated by the final graph convolutional layer. Each node i has a learned embedding $\mathbf{h}_i \in \mathbb{R}^{32}$, where each embedding is a vector in a 32-dimensional space. The graph representation $\mathbf{h}_G$ is calculated as the average of these node embeddings.

$$\mathbf{h}_G = \frac{1}{N}\sum_{i=1}^{N} \mathbf{h}_i \tag{16}$$

where $N = 100$ denotes the number of nodes in the graph. This pooled representation is subsequently processed through a linear transformation followed by a sigmoid activation:

$$\mathrm{Logit} = \mathbf{W}_c \mathbf{h}_G + b_c \tag{17}$$

where $\mathbf{W}_c \in \mathbb{R}^{1\times 32}$ represents the weight matrix and $b_c \in \mathbb{R}$ the bias term. The malware probability p is computed as:

$$p(\mathrm{Malware}) = \sigma(\mathrm{Logit}) = \frac{1}{1+\exp(-\mathrm{Logit})} \tag{18}$$

The final classification decision is made by thresholding this probability:

$$\mathrm{Prediction} = \begin{cases} \mathrm{Malware} & \text{if } p > .5 \\ \mathrm{Goodware} & \text{otherwise} \end{cases} \tag{19}$$

This formulation provides a probabilistic interpretation of the model's predictions while maintaining a simple decision boundary for binary classification.

3.4. Phase 4: Explainability and model analysis

GNNs have become a powerful tool for malware analysis, particularly when modeling relationships between entities (e.g., system calls, control flow graphs, or behavioral dependencies) as graphs. Their message-passing mechanism-where nodes aggregate features from neighbors-enables effective pattern recognition in graph-structured malware data. However, the black-box nature of GNNs poses challenges in security-sensitive applications, where interpretability is crucial for trust, debugging, and adversarial robustness Agarwal et al. (2023).

GNN Explainer addresses this by identifying the most influential subgraphs and node features behind predictions, making it indispensable for malware analysis. It reveals malicious substructures (e.g., suspicious API call sequences or obfuscated code patterns), helping analysts validate detections, reduce false positives, and combat evasion tactics like graph perturbations Longa et al. (2025). Furthermore, explainability is often required for compliance in automated threat detection systems. By clarifying model decisions, GNN Explainer ensures GNNs remain both

powerful and interpretable in cybersecurity workflows. For a node v at layer l, this process can be formalized as:

$$h_v^{(l)} = \text{UPDATE}^{(l)}\left(h_v^{(l-1)}, \text{AGGREGATE}^{(l)}\left(\{h_u^{(l-1)} \mid u \in \mathcal{N}(v)\}\right)\right) \tag{20}$$

where $h_v^{(l)}$ represents the feature vector of node v at layer l, $\mathcal{N}(v)$ denotes the neighborhood of v, and the AGGREGATE and UPDATE functions perform feature transformation and combination. The GNNExplainer method identifies influential nodes and edges by optimizing a subgraph G_S that maximizes mutual information with the model's prediction:

$$\max_{G_S} \text{MI}(Y, G_S) = H(Y) - H(Y \mid G_S) \tag{21}$$

This is where GNN Explainer becomes indispensable. It enhances interpretability by pinpointing the most influential subgraphs and node features behind a model's predictions. In malware analysis, this means highlighting suspicious subgraphs-such as a particular cluster or sequence pattern-that contribute to a malicious classification. Such insights are crucial for security analysts, enabling them to validate detections, reduce false positives, and refine threat intelligence. This approach identifies which API clusters (nodes) and their interactions (edges) most significantly influence the classification decision between malware and goodware.

4. Results and discussion

This section evaluates the proposed model using multiple datasets and standard performance metrics.

4.1. Dataset

Table 2 describes the composition of the two datasets used for model training and evaluation. Each dataset consists of sequences of API calls paired with a binary label: malware or goodware. Dataset 1 is used for training and consists of 300,000 samples, evenly split between 150,000 malware and 150,000 goodware instances. All samples were collected between 2023 and 2024, ensuring the dataset reflects modern threat behavior and current software trends. The malware set spans a diverse range of families, including trojans, ransomware, cryptominers, and remote access tools (RATs), all verified through VirusTotal and multiple antivirus engines to ensure accurate labeling. The goodware set includes signed, up-to-date enterprise software, development tools, operating system utilities, and commonly used user applications. This 1:1 class ratio eliminates imbalance, reducing model bias and enhancing training stability. Following this, Dataset 2 is allocated strictly for testing and contains 200,000 samples-100,000 malware and 100,000 goodware-collected during the same timeframe but kept entirely separate from training and validation to ensure a fair, unbiased evaluation. The use of a large, recent, and balanced training set improves generalization, while the separate, unseen test set provides an accurate measurement of real-world performance. This experimental design supports rigorous, reproducible evaluation across both malicious and benign software contexts.

Dataset 3 is composed of a Brazilian malware dataset containing 50,820 executable files that were collected between 2012–2018, comprising 29,704 malware samples (trojans, ransomware, spyware, worms, backdoors, adware) and 21,116 benign samples Ceschin et al. (2018). The dataset's six-year collection period provides temporal diversity, while its Brazilian origin offers regional threat relevance. With almost balanced representation (58 % malicious, 42 % benign) and comprehensive threat coverage, this dataset enables robust evaluation of malware detection systems, particularly for assessing classifier performance against region-specific and evolving threats while maintaining global applicability. The dataset's size and composition make it valuable for machine learning applications, regional threat analysis, and studying malware evolution trends. Dataset 4, sourced from the APIMDS dataset Ki et al. (2015), comprised API call sequences extracted from 23,080 malware samples and 300 benign samples, encompassing 2727 distinct API calls. The dataset represented a diverse range of malware categories, including backdoors, worms, packed malware, potentially unwanted programs (PUPs), trojans, and other variants. This broad coverage enabled a comprehensive evaluation of model performance across multiple threat classifications. We employed this dataset exclusively for testing purposes to rigorously assess the models' generalization capabilities and their effectiveness in detecting various malware types.

Table 2
Dataset sample counts.

Dataset	Purpose	Size	Goodware	Malware
Dataset 1	Train	300,000	150,000	150,000
Dataset 2	Test	200,000	100,000	100,000
Dataset 3 Ceschin et al. (2018)	External Test	50,820	21,116	29,704
Dataset 4 Ki et al. (2015)		23,380	300	23,080

4.2. Evaluation metrics

To evaluate GraphShield's performance, we employed standard metrics including precision, recall, and the F1-score, which together assess the model's predictive accuracy and balance between sensitivity and specificity. The false positive rate (FPR) and false negative rate (FNR) were analyzed to quantify critical misclassification risks, while overall accuracy provided a measure of total prediction correctness. Finally, the area under the ROC curve was used to evaluate the model's discriminative power between classes. These metrics (Eqs. (22)–(28)) collectively offer a rigorous and interpretable assessment of the model's performance, highlighting both its reliability and potential limitations.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{22}$$

Where: TP referees to the True Positives and FP referees to the False Positives

$$\text{Recall} = \frac{TP}{TP + FN} \tag{23}$$

Where: TP referees to the True Positives and FN referees to the False Negatives

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{24}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{25}$$

Where: TN referees to the True Negatives

$$\text{FPR} = \frac{FP}{FP + TN} \tag{26}$$

$$\text{FNR} = \frac{FN}{FN + TP} \tag{27}$$

$$\text{AUC} = \int_{-\infty}^{\infty} \text{ROC curve} \tag{28}$$

These metrics, when used together, offer a comprehensive evaluation of model performance, highlighting its strengths and weaknesses in different areas.

4.3. Experimental setup

Technical Specifications: All experiments were conducted on a workstation equipped with an Intel Core i7-10700K CPU (3.8 GHz base frequency), NVIDIA GeForce RTX 2070 GPU (8GB GDDR6 VRAM), and 32GB DDR4 RAM. The software environment comprised Ubuntu 24.04 LTS OS with Python 3.9.18, PyTorch 2.0.1, and PyTorch Geometric 2.4.0, utilizing CUDA 11.8 acceleration for GPU-accelerated graph operations. The implementation leveraged NVIDIA's cuDNN 8.9.7 for optimized deep learning primitives benchmarks.

Table 3
Validation metrics for different DL models.

Model	Precision	Recall	F1 Score	Accuracy	FPR	FNR	AUC-ROC
CNN	0.9450	0.9450	0.9450	0.9450	0.0542	0.0558	0.9900
LSTM	0.9164	0.9161	0.9161	0.9161	0.0715	0.0961	0.9750
BiGRU	0.9190	0.9185	0.9184	0.9185	0.0635	0.0994	0.9800
GRU	0.9037	0.9036	0.9036	0.9036	0.0897	0.1029	0.9650
BiLSTM	0.9228	0.9212	0.9212	0.9212	0.0485	0.1090	0.9700
CNN-LSTM	0.9509	0.9506	0.9506	0.9506	0.0358	0.0630	0.9920
CNN-BiLSTM	0.9457	0.9454	0.9454	0.9454	0.0417	0.0674	0.9850
CNN-GRU	0.9413	0.9406	0.9405	0.9406	0.0391	0.0797	0.9880
CNN-BiGRU	0.9418	0.9417	0.9417	0.9417	0.0501	0.0665	0.9830

Table 4
Testing metrics for different DL models.

Model	Precision	Recall	F1 Score	Accuracy	FPR	FNR	AUC-ROC
CNN	0.8430	0.5705	0.5883	0.5705	0.5623	0.0062	0.8500
LSTM	0.8258	0.3768	0.3375	0.3768	0.8183	0.0012	0.7200
BiGRU	0.8403	0.5739	0.5924	0.5739	0.5560	0.0120	0.8450
GRU	0.8417	0.6279	0.6503	0.6279	0.4803	0.0272	0.8800
BiLSTM	0.8228	0.3211	0.2472	0.3211	0.8918	0.0002	0.7000
CNN-LSTM	0.8361	0.5492	0.5649	0.5492	0.5882	0.0130	0.8300
CNN-BiLSTM	0.8263	0.3710	0.3284	0.3710	0.8262	0.0004	0.7500
CNN-GRU	0.8443	0.6114	0.6327	0.6114	0.5057	0.0156	0.8900
CNN-BiGRU	0.8370	0.5179	0.5277	0.5179	0.6319	0.0049	0.8200

Experimental Design and Evaluation Framework: The models were trained using a stratified 70:30 training-validation split on Dataset 1 (300,000 samples) to maintain class balance, with Dataset 2 (200,000 samples) reserved as an untouched test set. This partitioning strategy ensures robust evaluation while mitigating bias in performance assessment. Building upon this data division, the study introduces a comprehensive evaluation framework designed to systematically compare traditional sequence-based deep learning approaches with modern graph-based architectures. To achieve this, the research methodology employs a rigorous two-phase experimental design, assessing model performance across multiple dimensions. In the first phase, conventional sequence models-including CNNs, LSTMs, GRUs, and their hybrid variants-are evaluated using raw sequential API call traces. Special emphasis is placed on their generalization capabilities when transitioning from validation data to real-world testing scenarios. Transitioning to the second phase, the focus shifts to GNNs such as GCN, GAT, and GIN-CNN, which process malware samples represented as structural graphs. This phase critically examines their ability to maintain robust detection performance on unseen samples, providing a direct comparison against sequence-based methods.

4.4. Baseline DL models based on original sequence data

This section evaluates a range of deep learning architectures, including CNNs, RNN variants (LSTM, GRU, BiLSTM), and hybrid combinations like CNN-LSTM and CNN-GRU. Each model type represents a unique approach to handling spatial, sequential, or contextual data. CNNs focus on spatial feature extraction. RNNs and their gated variants handle temporal sequences. Hybrid models attempt to combine the strengths of both. Comparing these models across validation and testing stages reveals how architectural choices influence performance, generalization, and robustness in practical conditions. The evaluation of baseline deep learning models provides critical insights into their performance characteristics and generalization capabilities. The comprehensive analysis reveals distinct patterns between model architectures when comparing the validation and testing results.

During validation, as described in Table 3, hybrid architectures demonstrate superior performance, with CNN-LSTM emerging as the top performer for all key metrics, including precision (0.9509), recall (0.9506), and F1-score (0.9506), while maintaining an exceptionally low false positive rate of 0.0358. The standalone CNN model shows comparable strength with consistent scores of 0.9450 in multiple metrics, suggesting robust feature extraction capabilities. Among recurrent models, bidirectional variants exhibit balanced performance, though with slightly elevated false negative rates. Testing scenarios present a markedly different landscape, where simpler architectures unexpectedly outperform their more complex counterparts. As described in Table 4, the GRU model demonstrates remarkable resilience in real-world conditions, achieving the highest testing recall (0.6279) and F1-score (0.6503) among all evaluated models. This performance contrasts sharply with bidirectional models like BiLSTM, which suffer from severe performance degradation in testing environments, evidenced by recall dropping to 0.3211 and false positive rates exceeding 0.89. The CNN-GRU hybrid maintains competitive performance with an AUC of 0.8900, suggesting that certain hybrid configurations may offer better generalization than pure recurrent architectures.

These findings highlight several critical considerations for model selection and development. The significant performance gap between validation and testing metrics, as described in Fig. 2, underscores the limitations of relying solely on validation results for model assessment. While complex architectures excel in controlled validation environments, their practical utility may be compromised by overfitting or sensitivity to data distribution shifts. The strong showing of simpler models like GRU in testing conditions suggests that architectural complexity does not always translate to better real-world performance. These insights form a valuable foundation for subsequent model optimization efforts, emphasizing the need to balance theoretical performance with practical robustness in real-world applications.

4.5. Impact of cluster granularity on classification accuracy

Our initial objective was to determine the optimal number of clusters for indexing individual APIs. Traditional methodology, such as the elbow method, suggested 20 clusters as the optimal count based on WSS reduction (see Section 3.1.3). However, initial evaluations using the training dataset revealed that this configuration resulted in subpar classification accuracy, as shown in Fig 3. This divergence high-

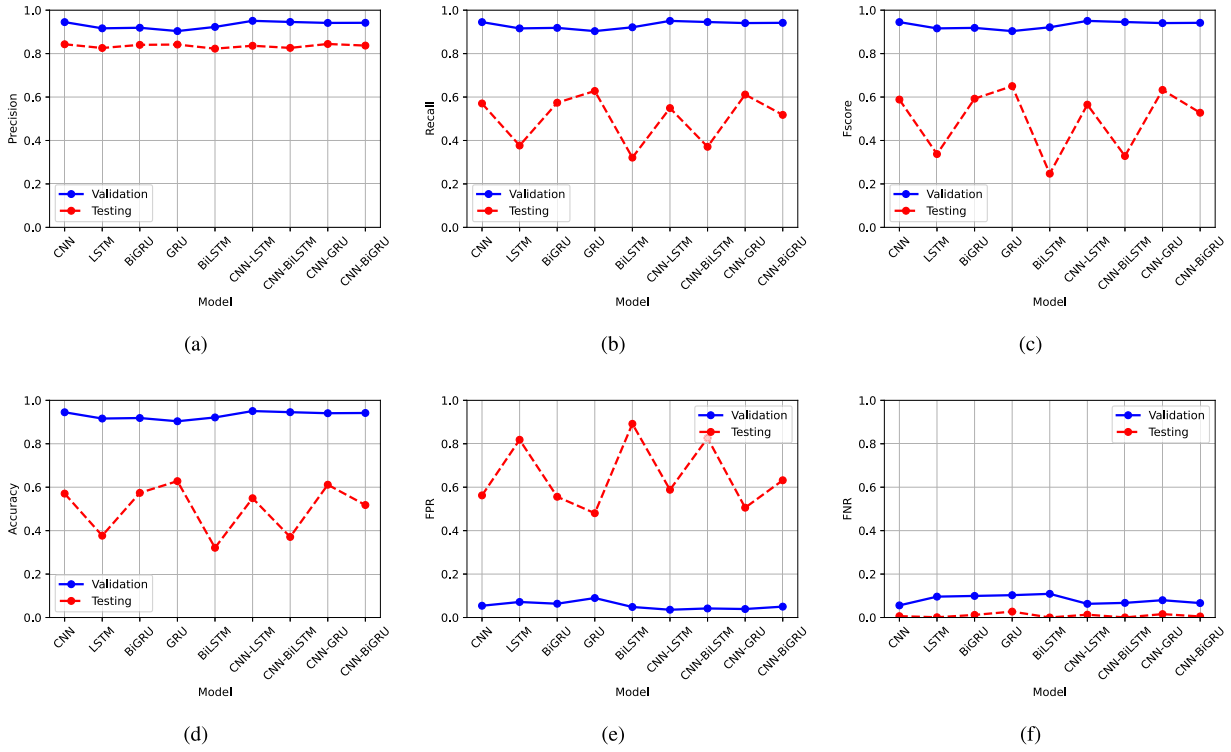


Fig. 2. Comparison of Validation and Testing metrics across different Deep Learning models: (a) Precision, (b) Recall, (c) Fscore, (d) Accuracy, (e) FPR, and (f) FNR.

Table 5
Sample API Cluster Assignments in Malware vs. Goodware.

API	Goodware Cluster	Malware Cluster
`VirtualAlloc`	5	18
`WriteProcessMemory`	74	18

lights that cluster count functions as a de facto hyperparameter in our pipeline-while unsupervised methods optimize for geometric cohesion, the downstream classification task requires tuning for discriminative power. Upon closer inspection, we found that API distributions across these clusters were highly imbalanced-some contained thousands of APIs, while others indexed only a few dozen. This imbalance caused the model to over-rely on a small subset of clusters, undermining its discriminative power. To address this, we treating cluster count as a tunable hyperparameter, deliberately increased the number of clusters beyond the elbow point, testing values from 20 to 120 in increments of 10.

As illustrated in Fig. 3, classification accuracy improved consistently across this range, peaking at 100 clusters. Beyond this point, further increases yielded no meaningful performance gains. Using KMeans++ to generate 100 clusters, we captured fine-grained API interaction patterns, achieving an optimal balance between granularity and interpretability. Rather than treating all API calls uniformly, clustering exposed subtle yet critical differences in how malware and goodware interact with system APIs-differences crucial for precise behavioral analysis. Moreover, this approach revealed deeper insights into API interactions. While some APIs appear harmless in isolation, they may exhibit malicious behavior when combined with others-a relationship effectively highlighted through clustering. For example, the `VirtualAlloc` → `WriteProcessMemory` sequence (discussed in Section 3.1.2) was found in distinct clusters for malware and goodware, as shown in Table 5. This divergence demonstrates how malware repurposes these APIs in ways that differ significantly from benign software.

Surprisingly, rather than plateauing at the theoretical optimum suggested by WSS, classification metrics continued to improve as cluster counts increased. This behavior mirrors supervised hyperparameter tuning, where optimal values often deviate from theoretical assumptions. The additional discriminative power from finer granularity led to notable gains in precision, recall, and F1-scores, particularly in the transition from small to medium cluster sizes (Fig. 3). While improvements slowed at higher cluster counts, the results clearly demonstrate that the optimal number of clusters for classification exceeds conventional recommendations, enabling more accurate and insightful malware detection.

4.6. Comparative evaluation of graph-Based malware detection architectures

In this section, we present a rigorous evaluation of the proposed graph-based architectures for malware detection, analyzing their validation and testing performance to assess both optimization efficacy and real-world generalization. The evaluation results, presented in Tables 6 and 7 and summarized in Fig. 4, provide a comprehensive comparative analysis of six graph-based architectures for malware detection, revealing critical insights into their performance characteristics and generalization capabilities. Across all models, validation metrics demonstrate exceptionally strong performance, with precision, recall, and F-scores consistently exceeding 0.995, suggesting that graph-based approaches are highly effective for malware classification on seen data. However, the testing results reveal more nuanced differences in model behavior that have significant implications for real-world deployment.

The GIN-CNN architecture emerges as the top-performing model, achieving the highest validation F-score (0.9985) and maintaining superior testing performance (F-score: 0.9951). This superior performance is particularly notable in its balanced precision-recall tradeoff, with testing precision of 0.9973 and recall of 0.9931. The model's consistently low false positive rate (FPR = 0.0028) and false negative rate (FNR = 0.0070) during testing suggest robust generalization capabilities. These results

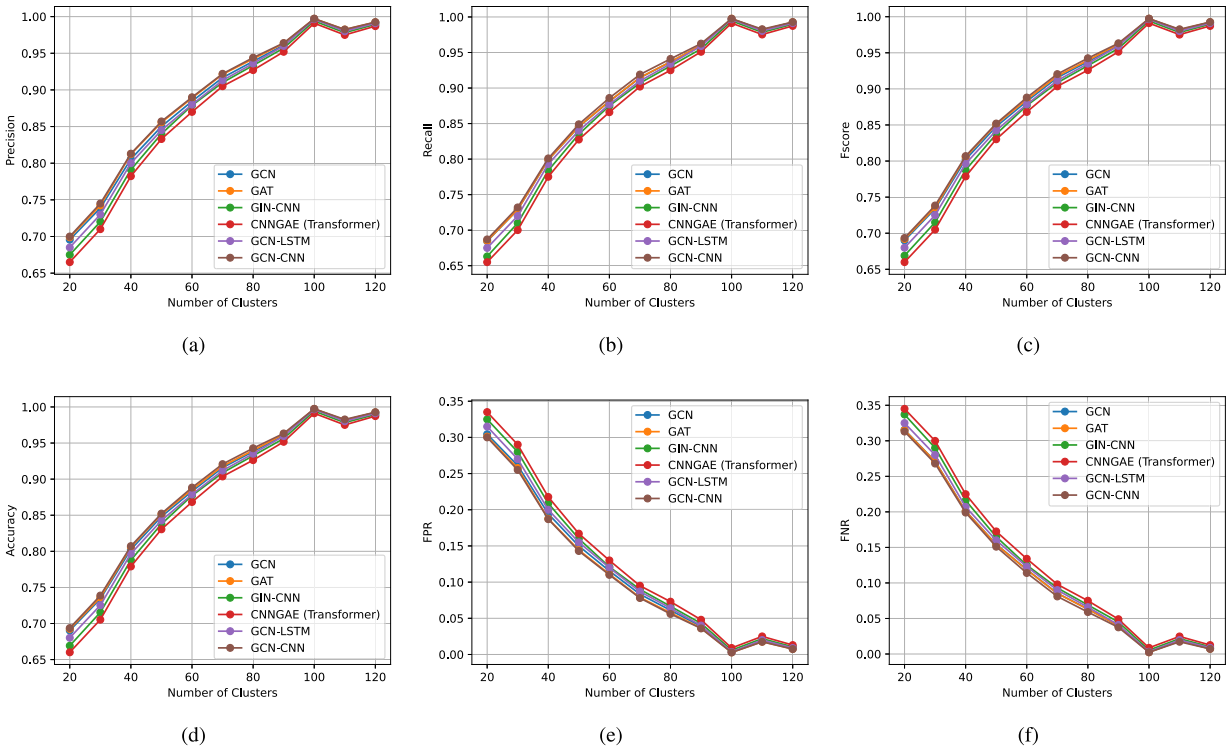


Fig. 3. Validation Performance of Models Metrics Across Increasing Cluster Counts: (a) Precision, (b) Recall, (c) Fscore, (d) Accuracy, (e) FPR, and (f) FNR.

Table 6
Validation metrics for different GNN models.

Model	Precision	Recall	F1 Score	Accuracy	FPR	FNR	AUC-ROC
GCN	0.9970	0.9974	0.9972	0.9972	0.0031	0.0026	0.9999
GAT	0.9971	0.9984	0.9978	0.9978	0.0030	0.0016	0.9996
GIN - CNN	0.9983	0.9986	0.9985	0.9985	0.0017	0.0014	0.9999
CNNGAE (Transformer)	0.9981	0.9983	0.9982	0.9982	0.0019	0.0017	0.9999
GCN-LSTM Autoencoder	0.9959	0.9981	0.9970	0.9970	0.0042	0.0019	0.9998
GCN-CNN Autoencoder	0.9970	0.9976	0.9973	0.9973	0.0031	0.0024	0.9997

Table 7
Testing Results for Different GNN Models Over Testing Dataset 2.

Model	Precision	Recall	F1 Score	Accuracy	FPR	FNR	AUC-ROC
GCN	0.9957	0.9915	0.9935	0.9936	0.0044	0.0085	0.9990
GAT	0.9941	0.9831	0.9881	0.9885	0.0061	0.0169	0.9989
GIN - CNN	0.9973	0.9931	0.9951	0.9951	0.0028	0.0070	0.9993
CNNGAE (Transformer)	0.9974	0.9844	0.9907	0.9909	0.0026	0.0156	0.9995
GCN-LSTM Autoencoder	0.9953	0.9942	0.9947	0.9947	0.0047	0.0058	0.9993
GCN-CNN Autoencoder	0.9962	0.9856	0.9907	0.9909	0.0039	0.0144	0.9995

indicate that the combination of graph isomorphism networks with convolutional operations may be particularly well-suited for capturing both local and global patterns in malware graph representations.

Performance consistency between validation and testing phases varies significantly across architectures. The GCN-LSTM Autoencoder demonstrates remarkable stability, with only a 0.0023 drop in F-score from validation (0.9970) to testing (0.9947). This suggests that the sequential processing capability of LSTM components may help maintain performance when encountering novel samples. In contrast, the GAT architecture shows the most substantial performance degradation, with recall dropping from 0.9984 in validation to 0.9831 in testing, accompanied by the highest testing false negative rate (FNR = 0.0169) among all models. This raises important questions about the robustness of attention mechanisms in security applications where false negatives carry high costs. The transformer-based CNNGAE exhibits an interesting performance pattern, maintaining high precision (0.9974 testing) but showing a notable recall decline (from 0.9983 to 0.9844). This suggests that while the model is highly confident in its positive predictions, it may become overly conservative when processing unseen samples. The standard GCN architecture demonstrates reliable performance (testing F-score: 0.9935), outperforming the more complex GAT while being slightly surpassed by GIN-CNN. This finding is particularly relevant for practical applications, as it suggests that increased model complexity does not always translate to better performance.

All models achieve near-perfect AUC scores (>0.9989) during testing, with GIN-CNN and CNNGAE reaching outstanding AUC values between 0.9993 and 0.9995, demonstrating the high efficiency and strong discriminative power of our approach in conventional malware detection; however, these exceptional scores suggest that current datasets may no longer sufficiently challenge advanced models, and the absence

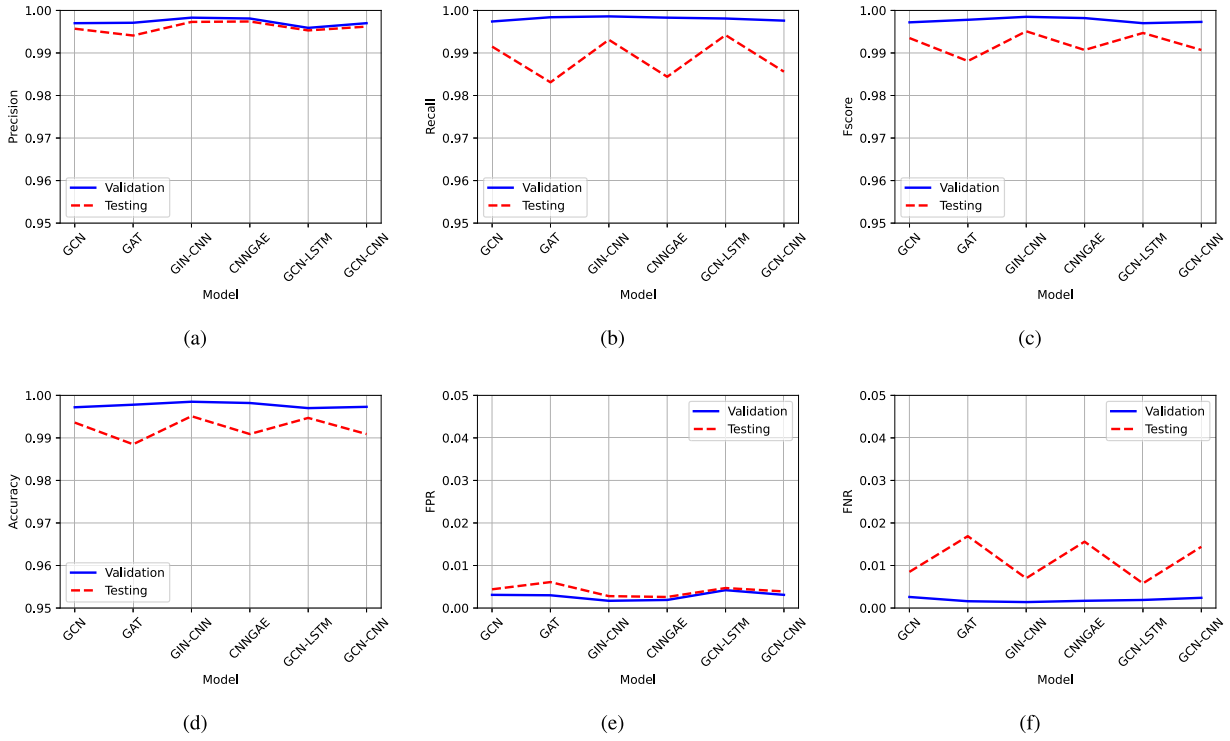


Fig. 4. Comparison of Validation and Testing metrics across different models: (a) Precision, (b) Recall, (c) Fscore, (d) Accuracy, (e) FPR, and (f) FNR.

of adversarial samples in the evaluation set could artificially inflate performance metrics, potentially masking vulnerabilities, indicating that while our method sets a new benchmark under standard conditions, future evaluations must incorporate harder, more realistic datasets to drive further progress. From an architectural perspective, the results reveal several important insights. The strong performance of GIN-CNN supports the effectiveness of neighborhood aggregation strategies combined with CNN operations for graph classification tasks. The consistency of GCN-LSTM suggests that sequential processing can enhance model stability, potentially by better capturing temporal or structural dependencies in the graph data. The comparative under-performance of GAT in testing scenarios indicates that attention mechanisms, while powerful in theory, may require additional regularization or training strategies to achieve reliable generalization in security applications. These findings have significant implications for both research and practice in malware detection. The demonstrated performance variations underscore the importance of rigorous testing methodologies that go beyond standard validation metrics. Future work should focus on developing more challenging evaluation benchmarks, incorporating adversarial samples and concept drift scenarios. Additionally, the results suggest promising directions for architectural innovations, particularly in combining the strengths of different approaches - for instance, integrating GIN's powerful graph representation capabilities with LSTM's sequential processing advantages.

For practical deployment, the choice of model involves important trade-offs. GIN-CNN offers the best overall accuracy but may require more computational resources. GCN-LSTM provides excellent stability for operational environments where consistent performance is critical. Standard GCN remains a strong baseline option when computational efficiency is prioritized. These considerations highlight the need for context-aware model selection in real-world security operations. Our work makes several important contributions to the field of graph-based malware detection. It provides the first comprehensive comparison of these six architectures using consistent evaluation metrics, establishing clear performance benchmarks. The analysis of generalization patterns offers valuable insights into model robustness that were not previously documented. Furthermore, the identification of specific strengths and limitations for each architecture guides future research toward the most promising development directions. These findings advance the understanding of how graph neural networks can be effectively applied to cybersecurity challenges while highlighting critical areas needing further investigation.

4.7. Comparison of DL and graph-Based models

The evaluation of deep learning (DL) models and graph-based models reveals significant differences in generalization capability across validation and testing phases. While DL architectures such as CNN-LSTM and CNN achieve strong validation performance (F-scores of 0.9506 and 0.9450, respectively, with AUC $>$ 0.98), their testing performance degrades substantially—e.g., CNN's recall falls from 0.9450 to 0.5705, and its false positive rate (FPR) rises from 0.0542 to 0.5623. This suggests that while these models fit the validation data well, they exhibit limited generalization to unseen test conditions, potentially due to their reliance on local feature representations that may not capture broader data dependencies. In contrast, graph-based models (e.g., GCN, GIN-CNN) demonstrate remarkable consistency between validation and testing. They maintain near-optimal metrics in both phases, with testing F-scores $\geq$ 0.9881, AUC $\geq$ 0.9989, and FPR/FNR rates remaining low ($\leq$ 0.0169). This stability highlights their superior ability to generalize, likely attributable to their explicit modeling of relational structures, which enables robust feature learning even under distribution shifts. Hybrid architectures like GCN-LSTM autoencoders further enhance reliability, achieving testing F-scores above 0.9907 without significant degradation.

The comparative evaluation between graph neural networks (GNNs) and conventional deep learning approaches demonstrates substantial improvements in malware detection performance. Using the standard improvement metric described in Eq. (29):

$$\text{Improvement } \% = \left(\frac{\text{GNN Score} - \text{DL Score}}{\text{DL Score}} \right) \times 100 \tag{29}$$

Table 8
Quantitative Performance Comparison Between Best DL and GNN Models.

Metric	DL (GRU)	GNN (GIN-CNN)	Δ	Improvement %
Precision	0.8417	0.9973	+0.1556	18.48
Recall	0.6279	0.9931	+0.3652	58.16
F1-Score	0.6503	0.9951	+0.3448	53.02
Accuracy	0.6279	0.9951	+0.3672	58.50
FPR	0.4803	0.0028	-0.4775	-99.42
FNR	0.0272	0.0070	-0.0202	-74.26
AUC	0.8800	0.9993	+0.1193	13.56

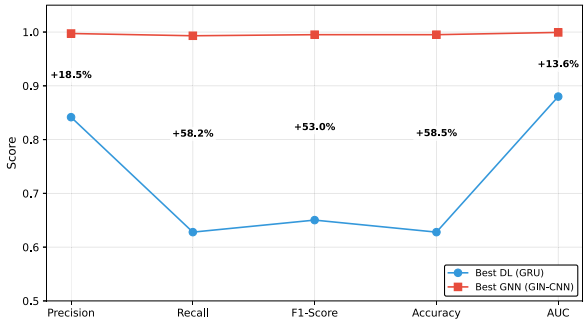


Fig. 5. Performance Improvement: Best DL vs. Best GNN Model.

We quantify the advantages of the best-performing GNN model (GIN-CNN) over the optimal deep learning model (GRU), as shown in Table 8 and illustrated in Fig. 5. The most notable enhancement appears in recall, with a 58.16 % improvement from 0.6279 to 0.9931, indicating superior detection of true malware cases. This is particularly significant for security applications, where minimizing false negatives is critical.

The false positive rate (FPR) shows a remarkable 99.42 % reduction, decreasing from 0.4803 to 0.0028. This demonstrates that GNN architectures achieve near-perfect specificity while maintaining high sensitivity, addressing a key limitation of traditional machine learning in security contexts where false alarms incur significant operational costs. The comprehensive 58.50 % improvement in accuracy, increasing from 0.6279 to 0.9951, confirms that GNNs deliver balanced performance gains across all classification metrics. The near-perfect AUC score of 0.9993 suggests that GNN-based approaches achieve nearly ideal separation between malware and benign samples, representing a qualitative leap beyond conventional deep learning methods. This performance stems from the inherent architectural advantages of graph-based learning in capturing structural and relational patterns in malware binaries. The consistent superiority across all GNN variants (Table 7) indicates these benefits are fundamental to the GNN paradigm rather than implementation-specific optimizations.

These results establish GNN architectures as the new baseline methodology for malware detection research and deployment. The improvements in both detection rates (recall) and operational efficiency (FPR reduction) meet the dual requirements of security applications: comprehensive threat identification with minimal false alarms. The magnitude of improvement across all metrics suggests that graph-based learning represents a paradigm shift rather than incremental progress in malware detection. The disparity in generalization suggests that graph-based methods are inherently more suited for tasks where structural relationships are critical. DL models, despite strong validation performance, may struggle with test data variability due to their limited inductive biases for relational reasoning. Thus, graph architectures emerge as the preferred choice for real-world applications requiring consistent and generalizable performance. Further investigation into dataset characteristics and training protocols could help clarify the root causes of DL models' generalization gaps, but the empirical advantage of graph-based approaches remains clear.

4.8. Progressive performance analysis

In this section, we analyze the performance of six advanced malware detection models to identify their strengths, weaknesses, and suitability for real-world deployment. It aims to guide security architects in selecting appropriate detection architectures by comparing precision, recall, stability, and operational factors critical to effective and sustainable threat detection.

Dataset 2 was partitioned into 10 non-overlapping, temporally ordered chunks of 20,000 samples each (totaling the 200,000-sample holdout set), with each chunk preserving the original 1:1 malware-to-goodware class distribution. This partitioning strategy served three critical purposes: (1) enabling progressive evaluation of model performance under sequential data exposure to detect temporal degradation, (2) simulating real-world deployment scenarios where security systems analyze samples in discrete batches, and (3) monitoring the model behavior for detecting performance fluctuations between chunks. The chronological ordering of samples within chunks preserved authentic behavioral evolution patterns, while the fixed chunk size ensured consistent computational load across evaluation phases. This structured evaluation approach provided the foundation for analyzing each model's behavior under real-world conditions. With data processed in chronological order and under consistent constraints, the results reflected not only static accuracy metrics but also how models held up over time and scale. The experimental evaluation of six advanced malware detection architectures reveals critical insights into their operational characteristics and practical deployment considerations.

According to Fig. 6, the Transformer model establishes itself as the precision leader, achieving remarkable consistency with its peak precision of 0.9992, representing near-perfect positive predictive value. This exceptional performance comes from the model's self-attention mechanism, which appears particularly adept at learning definitive malware signatures while maintaining an impressively low false positive rate of just 0.0008. Such performance characteristics make the Transformer architecture ideally suited for security operations centers where alert validation resources are constrained, or for endpoint protection solutions where false positives directly impact user experience and productivity.

However, this precision advantage is counterbalanced by the Transformer's recall degradation, which manifests as a concerning upward trend in false negative rates. The model's FNR progression from 0.0105 to 0.0255 suggests diminishing effectiveness against novel attack vectors and evolving malware techniques. This limitation likely stems from the Transformer's tendency to form conservative decision boundaries that prioritize precision over comprehensive threat coverage. In practical terms, while the model excels at identifying clear-cut malware cases, it may struggle with polymorphic code, adversarial samples, or newly emerging threat families that deviate from its learned patterns.

The LSTM-GAE architecture presents a compelling alternative with fundamentally different operational characteristics. Its superior recall performance, peaking at 0.9956, demonstrates exceptional sensitivity to malicious patterns while maintaining false negative rates as low as 0.0044. This capability stems from the model's dual architecture - the LSTM component effectively captures temporal and sequential patterns in malware behavior, while the graph autoencoder processes structural relationships within the code. The synergy between these components allows LSTM-GAE to maintain high detection rates even as malware evolves, making it particularly valuable for network perimeter defense and critical infrastructure protection, where missed detections carry severe consequences.

GIN-GAE emerges as the most stable performer across all evaluation metrics, with its graph isomorphism network foundation providing consistent discrimination capability throughout all data chunks. While its peak recall of 0.9947 slightly trails LSTM-GAE, the model's performance envelope remains remarkably tight, with less than 0.007 variation in any primary metric across all chunks. This stability suggests that GIN-GAE's message-passing architecture generalizes particularly well to un-

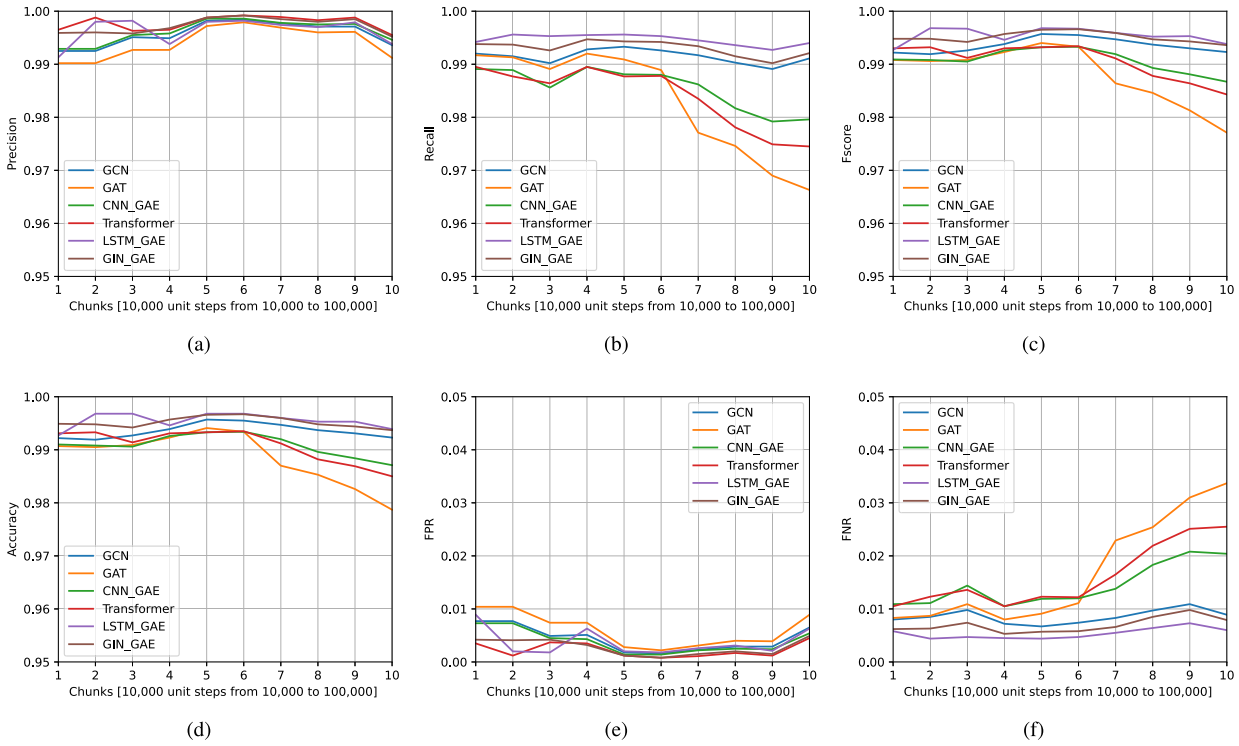


Fig. 6. Comparison of metrics across different models:: (a) Precision, (b) Recall, (c) Fscore, (d) Accuracy, (e) FPR, and (f) FNR.

seen samples, likely due to its theoretically grounded approach to graph representation learning. For security teams prioritizing predictable performance and operational consistency, GIN-GAE offers an attractive balance of capability and reliability. The comparative analysis of CNN-GAE and basic GCN models reveals important architectural limitations. CNN-GAE's performance degradation in later chunks, particularly in recall (declining from 0.9891 to 0.9796), suggests that its convolutional approach may lack the adaptability needed for long-term deployment without frequent retraining. The basic GCN architecture shows even more pronounced variability, with its false positive rate reaching 0.0089 in some chunks, nearly an order of magnitude higher than the Transformer's best performance. This instability likely stems from the GCN's simpler message-passing mechanism and lack of specialized components for handling the unique characteristics of malware detection graphs.

The AUC metrics, while uniformly excellent across all models, provide an interesting case study in the limitations of single-number performance assessments. The Transformer's theoretical advantage (0.9998 peak AUC) belies its practical recall limitations, while LSTM-GAE and GIN-GAE's slightly lower AUC values (0.9996 and 0.9998, respectively) correspond with more balanced real-world performance. This discrepancy underscores the critical importance of multi-dimensional evaluation in security applications, where different error types carry substantially different operational consequences. Temporal analysis reveals three distinct performance phases across the evaluation period. The initial chunks (1–4) serve as a capability baseline, showing all models performing at or near their peak levels. The middle phase (chunks 5–7) represents an optimal operating window where architectural differences become most pronounced, with the Transformer's precision advantage and LSTM-GAE's recall superiority clearly visible. The final phase (chunks 8–10) acts as a stress test, exposing vulnerabilities in certain architectures while validating the robustness of others. This progression mirrors real-world deployment scenarios where models must maintain effectiveness against both current threats and emerging attack vectors.

From an implementation perspective, the computational requirements and inference characteristics of each model present additional considerations. The Transformer's memory-intensive self-attention mechanism may limit its deployment in resource-constrained environments, while LSTM-GAE's sequential processing could introduce latency challenges in high-throughput scenarios. GIN-GAE's efficient graph processing offers a compelling middle ground, with inference characteristics that scale well to large enterprise environments. These operational factors must be weighed alongside pure detection metrics when selecting architectures for production deployment. The evaluation also highlights important considerations for model maintenance and lifecycle management. The performance degradation observed in some architectures suggests that static deployments may become increasingly ineffective over time. This finding supports the adoption of continuous learning frameworks, where models periodically incorporate new threat intelligence while preserving knowledge of historical attack patterns. The stability demonstrated by LSTM-GAE and GIN-GAE suggests these architectures may be particularly amenable to such approaches, potentially offering extended service life with minimal performance decay.

For security architects designing comprehensive protection systems, these findings suggest several strategic approaches. Precision-critical applications such as Email filtering or endpoint protection would benefit from Transformer deployments, possibly augmented with secondary verification mechanisms to mitigate its recall limitations. Network monitoring and perimeter defense systems would derive greater value from LSTM-GAE's comprehensive detection capabilities, particularly when paired with robust alert correlation systems to manage its slightly higher false positive rate. Enterprise-scale deployments might favor GIN-GAE for its balance of performance and operational stability, particularly in heterogeneous environments with diverse protection requirements. The research also identifies several promising directions for future work. Hybrid architectures combining the Transformer's precision with LSTM-GAE's recall capabilities could potentially achieve best-of-both-worlds performance. Investigation of attention mechanisms in GIN-GAE

might yield improvements in interpretability without sacrificing stability. Additionally, the development of specialized continual learning approaches for malware detection models could significantly extend their operational lifespan and adaptability to evolving threats.

In conclusion, this comprehensive evaluation demonstrates that modern graph-based malware detection architectures offer powerful capabilities, but with distinct performance characteristics that must be carefully matched to operational requirements. While LSTM-GAE emerges as the most capable general-purpose solution, the Transformer and GIN-GAE each fulfill critical niche requirements. The findings underscore that effective malware detection system design requires equal consideration of detection metrics, operational constraints, and maintenance requirements to achieve optimal long-term protection efficacy.

5. Cross-Dataset generalization and robustness

The ability of machine learning models to generalize across diverse datasets serves as a critical measure of their robustness and real-world applicability. Our comparative analysis of model performance on external Datasets 3 and 4 (Tables 9 and 10), as illustrated in Fig. 7, reveals varying generalization capabilities across different data distributions.

Dataset 3 results demonstrate strong model performance, with precision and recall values consistently close to 0.99 and F1 scores showing comparable results. Accuracy remains robust, generally ranging between 0.99 and 0.992. False positive and false negative rates remain below 0.01 and 0.011, respectively, indicating minimal classification errors. The AUC-ROC values between 0.996 and 0.997 further validate the models' effective generalization capabilities on this dataset. Dataset 4 presents a more challenging testing environment, as evidenced by performance declines across key metrics. Precision, recall, and F1 scores decrease to approximately 0.98-0.985, with corresponding reductions in accuracy. False positive rates rise to 0.0095, while false negative rates approach 0.02 in some cases. AUC-ROC values show a modest decline, ranging from 0.9918 to 0.9950. This shift suggests the dataset introduces greater variability or noise that impacts model effectiveness. Notably, certain architectures like CNNGAE (Transformer) exhibit more pronounced performance drops, revealing heightened sensitivity to data distribution shifts. Across both datasets, the GIN-CNN model distinguishes itself by consistently delivering the highest precision, recall, and AUC-ROC scores, underscoring its superior robustness and generalization capability. Other models, including GCN and GCN-based autoencoders, demonstrate stable but slightly inferior performance, indicating reliable yet less optimal adaptation to different data characteristics.

6. Model explainability and comprehensive model-Specific analysis

The interpretability of GNNs is crucial for building trust and ensuring their responsible deployment in security-sensitive applications. To address this need, GNN Explainer provides a powerful framework for enhancing model transparency by generating human-understandable explanations for GNN predictions. Specifically, it identifies the most influential subgraph structures and node features driving model decisions, effectively demystifying the often opaque reasoning processes of GNNs. Our analysis examines the top-20 highest-scoring nodes identified by each model, revealing consistent patterns in API usage that distinguish malicious from benign behavior.

The GCN model effectively identifies local API call patterns that distinguish malware from legitimate software. As shown in Fig. 8a, Node 18 emerges as particularly significant for malware detection. This node captures critical APIs including `VirtualAllocEx`, `WriteProcessMemory`, and `CreateRemoteThread`—a combination strongly associated with process injection techniques. The model reveals meaningful connections between Node 18 and neighboring nodes like Node 75 and Node 65, which contain complementary APIs such as `OpenProcess` and `VirtualProtect`. Together, these patterns paint a clear picture of malicious memory manipulation behaviors.

In the goodware classification shown in Fig. 8b, Node 11 stands out with its characteristic benign APIs. Functions like `GetSystemTime`, `GetVersionEx`, and `LoadLibrary` represent typical operations for legitimate system queries and library loading. The GCN's neighborhood aggregation mechanism successfully differentiates these harmless patterns from their malicious counterparts by analyzing the contextual relationships between nodes.

The Graph Attention Network (GAT) brings a dynamic perspective to node relationship analysis through its attention mechanisms. For malware detection in Fig. 9a, the model highlights Node 70 and Node 44 as key indicators. The former contains privilege escalation APIs like `OpenProcessToken` and `AdjustTokenPrivileges`, while the latter focuses on persistence mechanisms through registry operations. These nodes show strong associations with Node 47, which includes anti-debugging checks, revealing how GAT captures coordinated malicious behavior across different attack phases. When examining goodware patterns in Fig. 9b, the attention weights distribute more evenly across nodes like Node 95, which handles routine system operations. This contrasts sharply with the concentrated attention on specific malicious patterns, demonstrating GAT's ability to recognize the more diverse API usage profiles of legitimate software.

Table 9
Model Performance Comparison Over Testing Dataset 3.

Model	Precision	Recall	F1 Score	Accuracy	FPR	FNR	AUC-ROC
GCN	0.9890	0.9900	0.9895	0.9900	0.0055	0.0095	0.9960
GAT	0.9875	0.9885	0.9880	0.9885	0.0050	0.0112	0.9962
GIN-CNN	0.9910	0.9920	0.9915	0.9920	0.0035	0.0085	0.9970
CNNGAE (Transformer)	0.9900	0.9895	0.9897	0.9900	0.0045	0.0108	0.9968
GCN-LSTM Autoencoder	0.9898	0.9905	0.9901	0.9905	0.0048	0.0098	0.9965
GCN-CNN Autoencoder	0.9895	0.9900	0.9897	0.9902	0.0047	0.0105	0.9967

Table 10
Model Performance Comparison Over Testing Dataset 4.

Model	Precision	Recall	F1 Score	Accuracy	FPR	FNR	AUC-ROC
GCN	0.9817	0.9812	0.9814	0.9817	0.0082	0.0188	0.9925
GAT	0.9809	0.9817	0.9813	0.9818	0.0075	0.0183	0.9930
GIN-CNN	0.9842	0.9857	0.9849	0.9845	0.0047	0.0143	0.9950
CNNGAE (Transformer)	0.9805	0.9802	0.9803	0.9805	0.0095	0.0198	0.9918
GCN-LSTM Autoencoder	0.9827	0.9815	0.9821	0.9827	0.0062	0.0185	0.9937
GCN-CNN Autoencoder	0.9812	0.9817	0.9814	0.9817	0.0065	0.0183	0.9933

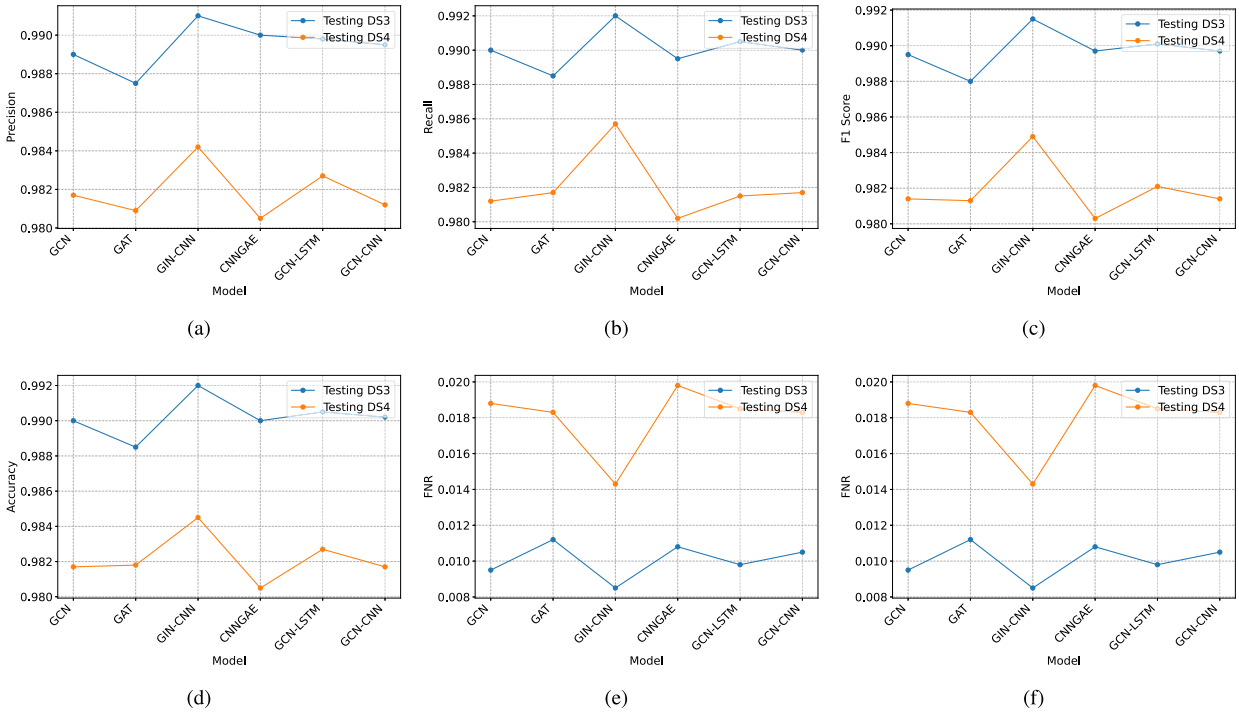


Fig. 7. Comparison of Models' Performance Over External Testing Datasets: (a) Precision, (b) Recall, (c) F1 Score, (d) Accuracy, (e) False Positive Rate (FPR), and (f) False Negative Rate (FNR).

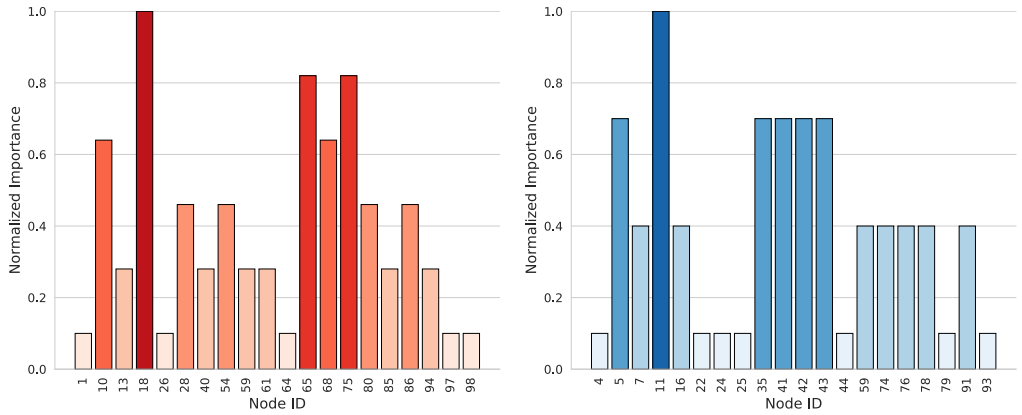
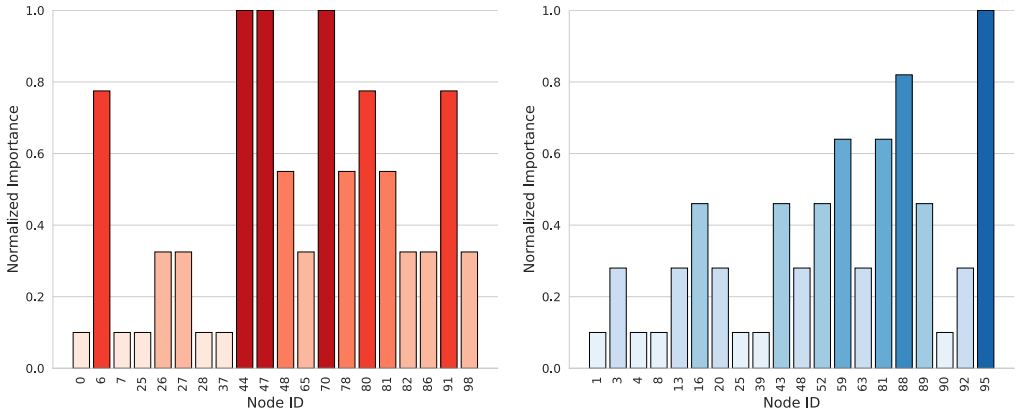


Fig. 8. GCN model explainability results showing (a) malware-related nodes and (b) goodware-related nodes through graph transition matrices.

The Graph Isomorphism Network (GIN) model, particularly when combined with a Graph Autoencoder (GAE), provides a unique perspective on analyzing API call graphs for malware detection. Unlike other models that focus on local node relationships or attention mechanisms, GIN excels at capturing the broader structural patterns that distinguish malicious behavior from legitimate software operations. In the malware classification case shown in Fig. 10a, Node 62 stands out for its inclusion of functions such as `CreateToolhelp32Snapshot`, `Process32FirstW`, and `Process32NextW`. These APIs are commonly used for enumerating running processes-an early-stage reconnaissance tactic in many malware strains. GIN also assigns high importance to Node 89, which includes `OpenSCManagerW` and `StartServiceCtrlDispatcherW`, both involved in manipulating or interfacing with Windows services. These operations often appear in malware that persists by creating or abusing background services. Node 41 contains `RegCreateKeyExW` and `RegSetValueExW`, indicating an intent to modify the Windows Registry-a frequent tactic used to establish persistence or modify system behavior. GIN's focus on these structurally related nodes uncovers a comprehensive view of the malware lifecycle, spanning reconnaissance, privilege usage, and persistence.

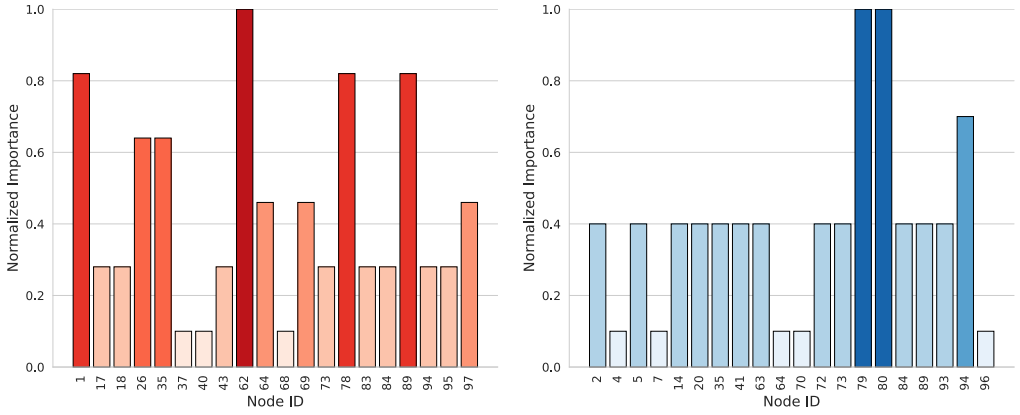
In contrast, Fig. 10b illustrates how GIN identifies benign software through nodes that contain functions associated with standard application behavior. Node 10 includes APIs such as `GetCurrentDirectoryW`, `SetCurrentDirectoryW`, and `PathCombineW`, all of which are used in safe file path handling and environment setup. These are typical during application initialization or resource loading. Additional benign indicators are found in neighboring nodes like Node 28 and Node 33, which feature `QueryPerformanceCounter`, `GetTickCount64`, and `GetLocalTime`. These functions provide basic timing or system state information, often used in application monitoring or UI rendering. GIN's structure-aware aggregation allows it to differentiate these benign sequences from the more targeted and disruptive operations found in malware samples.

Transformer-based models excel at identifying long-range dependencies in API call graphs. Fig. 11a reveals Node 92 as central to malware communication patterns, with its inter-process communication APIs.



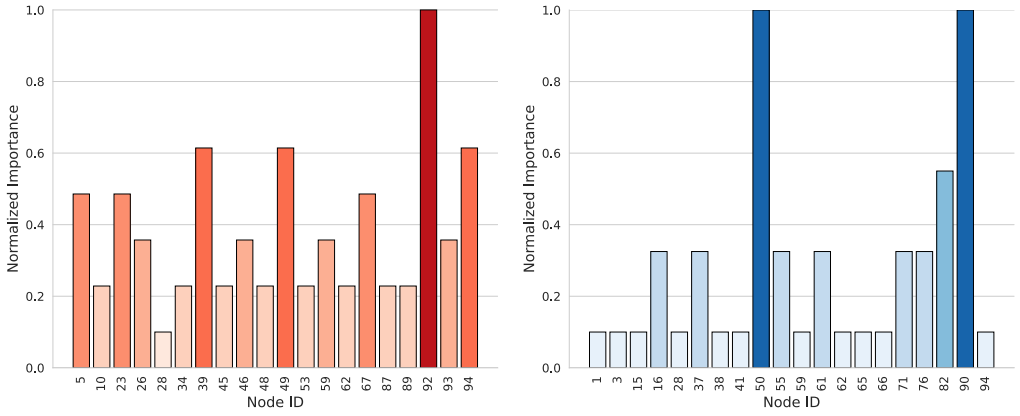
(a) (b)

Fig. 9. GAT model explainability results showing (a) malware-related nodes and (b) goodware-related nodes through graph transition matrices.



(a) (b)

Fig. 10. GIN model explainability results showing (a) malware-related nodes and (b) goodware-related nodes through graph transition matrices.

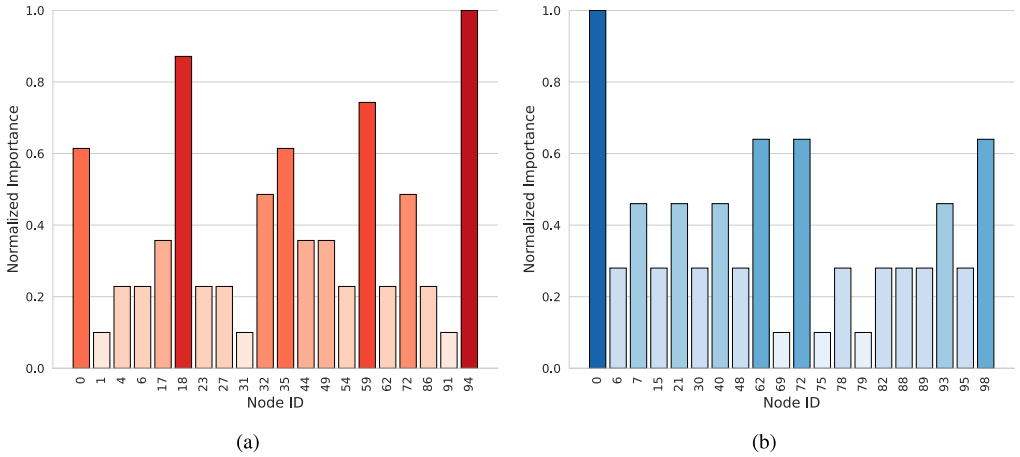


(a) (b)

Fig. 11. Transformer model explainability results showing (a) malware-related nodes and (b) goodware-related nodes through graph transition matrices.

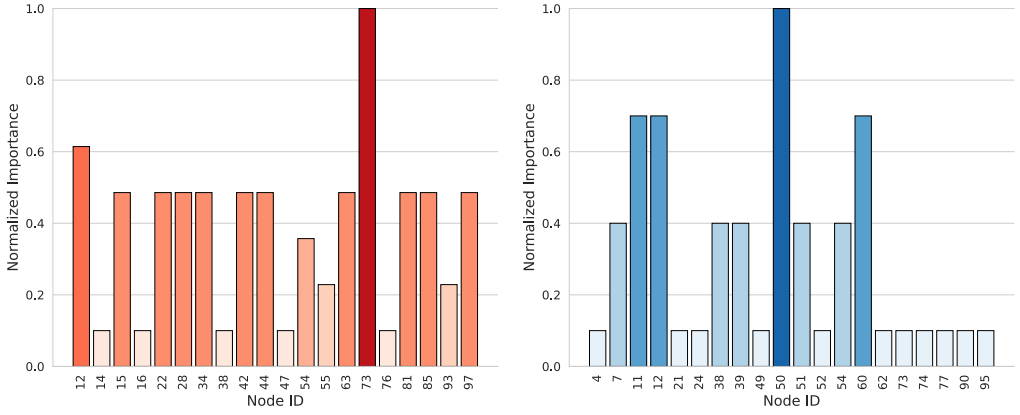
This connects meaningfully with Node49ı̀s process manipulation functions and Node94ı̀s persistence mechanisms, showing how Transformers capture comprehensive attack sequences that span multiple execution contexts. The goodware analysis in Fig. 11b presents a different picture, where nodes like Node 90 handle standard initialization routines. The model's global attention mechanism effectively distinguishes these legitimate operational flows from malicious ones by recognizing their fundamentally different structural patterns.

LSTM-based Graph Autoencoders provide unique insights into temporal API call sequences. The malware analysis in Fig. 12a shows Node94ı̀s registry operations following Node18ı̀s injection techniques and Node59ı̀s process creation—a classic malware lifecycle progression. Conversely, goodware sequences in Fig. 12b feature nodes like Node 0 handling benign system interactions, with temporal patterns reflecting normal software operation flows rather than malicious sequences.



(a) (b)

Fig. 12. LSTM-GAE model explainability results showing (a) malware-related nodes and (b) goodware-related nodes through graph transition matrices.



(a) (b)

Fig. 13. CNN-GAE model explainability results showing (a) malware-related nodes and (b) goodware-related nodes through graph transition matrices.

The CNN-GAE model, as illustrated in Fig. 13a, identifies Node 73 as a critical malware indicator. This node's structural relationships with neighboring nodes highlight memory manipulation behaviors. For goodware, Fig. 13b shows Node 50 as a central hub for benign operations, such as system initialization and resource management, demonstrating the model's ability to differentiate between malicious and benign structural patterns.

Expanded Cross-Model Insights and Implications

Several nodes repeatedly emerge as critical indicators of malicious activity across multiple graph-based neural network models. Node 59, for example, is consistently identified as important in GCN (Fig. 8a), Transformer (Fig. 11a), and LSTM-GAE (Fig. 12a) architectures. Similarly, Node 94 shows high relevance in Transformer (Fig. 11a), CNN-GAE (Fig. 13a), and LSTM-GAE (Fig. 12a) models. These overlaps suggest that structural or behavioral patterns tied to these nodes are strongly linked to malware-regardless of the model used.

These nodes rarely operate in isolation. Node 59 often forms tight intra-subgraph connections with Node 94 in attention-based architectures, suggesting a shared role in localized malicious behavior. Their recurrent co-activation across different models implies that they participate in synchronized operations such as process injection or memory manipulation. Node 26 further supports this observation. It emerges as a high-importance node in GCN (Fig. 8a), GAT (Fig. 9a), and GIN-GAE (Fig. 10a) models and frequently appears in conjunction with Nodes 28 and 18, forming dense clusters with high internal connectivity. These subgraph formations imply that these nodes may collectively encode specific tactics or stages within an attack sequence.

Beyond internal structure, inter-node relationships reveal additional dynamics. Nodes like 26 and 59 often act as bridges between otherwise separate functional subgraphs. For instance, Node 26 may connect registry operations with memory allocation flows, acting as a pivot across system functionalities. These cross-subgraph connections indicate that certain nodes are not only critical within their local neighborhoods but also central to broader communication across the malicious process graph. This behavior reflects a modular but coordinated malware design, where key nodes manage transitions between discrete functional stages like setup, execution, and persistence.

This view challenges the assumption that node importance is strictly local. When mapping how these nodes link subgraphs, a clearer picture of behavioral architecture emerges. Their ability to coordinate cross-functional activity suggests that they encode not just signals but operational roles. Nodes that exhibit both high intra-connectivity and strong inter-graph linkage become prime candidates for generalizable detection signatures. They reveal how malware maintains robustness against structural variation in different system contexts.

In contrast, nodes associated with benign behavior follow a different relational logic. Node 59, while sometimes flagged as malicious, is also verified as benign in three models when it appears in structurally distinct contexts. When Node 59 is embedded within sparse and predictable subgraphs alongside Nodes 7, 4, or 16, its behavior aligns with routine system operations. This shows how context, not just identity, defines a

node's behavioral profile. Node 7 consistently appears in GCN (Fig. 8b), GIN-GAE (Fig. 10b), CNN-GAE (Fig. 13b), and LSTM-GAE (Fig. 12b), anchoring benign subgraphs that include Nodes 4, 5, and 16. These nodes exhibit low-entropy, repetitive patterns typical of benign processes such as file access or system queries. Their intra-subgraph connections are stable but not densely interwoven, and they rarely bridge to distant graph regions. This relative isolation suggests that benign subgraphs operate within narrow behavioral bounds. They are less structurally integrated into global process flows and more constrained in their functional range.

This structural separation between benign and malicious clusters reflects fundamental differences in execution context and operational intent. Malicious subgraphs favor modular, densely connected, and strategically bridged architectures. Benign clusters remain localized, sparse, and functionally narrow. Recognizing this contrast can sharpen detection strategies by combining node importance with structural position and relational role. The implications for security operations are immediate. High-importance nodes like 18, 26, 59, and 94 should not only be monitored individually but also analyzed in terms of their relational patterns. Behavioral signatures must include recurring API subsequences and co-occurrence relationships-such as OpenProcess followed by VirtualAllocEx and WriteProcessMemory for injection, or RegCreateKeyEx with ShellExecute for persistence. These sequences often span multiple interconnected nodes, and their detection depends on preserving the integrity of the graph structure. Detection systems that focus solely on individual node scores miss the broader picture. Effective detection requires decoding the full behavioral graph-identifying not just which nodes matter, but how they connect, collaborate, and transition through malicious operations.

7. Operational impact and deployment value

The findings of this study have direct operational relevance for a range of stakeholders, including cybersecurity teams, enterprise IT departments, malware analysts, and national security organizations. While GraphShield demonstrates technical advances in detection accuracy and model design, its real-world value lies in its ability to address persistent pain points in malware defense-alert fatigue, generalization gaps, and black-box decision-making. For security operations centers (SOCs) and endpoint protection platforms, GraphShield offers a high-precision behavioral detection framework that maintains a false positive rate below 1 %. This level of reliability is critical in large-scale enterprise environments, where even modest error rates can result in thousands of false alerts. By capturing both structural and semantic relationships between API calls, the system detects evasive and staged malware more effectively than conventional sequence-based approaches, reducing triage time and improving analyst trust.

Building on this, cybersecurity vendors and dynamic analysis tool developers can integrate GraphShield into their existing pipelines to enhance threat visibility. Its modular design allows for deployment within automated sandbox environments, where it can generate interpretable detections on-the-fly. The inclusion of the GNNExplainer module further enables explainable AI workflows, giving security teams visibility into which behavioral signals triggered a detection-an increasingly important capability under compliance standards such as GDPR and NIST AI risk frameworks.

In a broader policy context, these findings underscore the need to move beyond static signature-based detection at the national and organizational level. As attackers adopt fileless, obfuscated, and context-aware techniques, behavioral detection grounded in graph learning offers a scalable, forward-compatible defense strategy. Policymakers and infrastructure regulators can use this insight to guide investment in AI-driven cybersecurity tools that prioritize adaptability and resilience. Lastly, for researchers and system architects, GraphShield demonstrates how combining language models, graph structures, and hybrid GNNs can create detection systems that generalize across malware variants while remaining transparent and auditable. Its architecture serves as a practical blueprint for future systems that must operate under adversarial, high-volume, and real-time conditions. Therefore, these implications highlight the role of GraphShield not only as a research contribution but as a possible deployable and policy-aligned solution to one of the most urgent challenges in cybersecurity.

8. Limitations and future directions

The proposed GraphShield model demonstrates strong performance in malware detection, but several limitations must be acknowledged. First, the GraphShield relies heavily on API call sequences as behavioral indicators, which may not capture all malicious activities. Some advanced malware employs evasion techniques such as API obfuscation, indirect system calls, or direct kernel manipulation, potentially bypassing detection. Additionally, while the GraphShield model performs well on unseen test data, its ability to detect truly novel (zero-day) threats depends on the diversity of the training dataset. Adversarial attacks that manipulate API call sequences could exploit this limitation.

Another challenge is computational overhead. The preprocessing pipeline-including BERT embeddings, clustering, and graph construction-introduces significant complexity, which may hinder real-time deployment in resource-constrained environments. Furthermore, while dynamic analysis (API call tracing) is more resilient to obfuscation than static analysis, it requires execution in controlled environments such as sandboxes, introducing latency and scalability constraints.

Dataset bias also poses a concern. The GraphShield's performance is contingent on the quality and representativeness of labeled malware and goodware samples. If certain malware families are overrepresented in training data, detection accuracy for underrepresented variants may suffer. Finally, interpretability remains a challenge. Graph neural networks, while powerful, often function as black-box models, making it difficult to explain why a specific API sequence is flagged as malicious. This lack of transparency could complicate forensic analysis and incident response.

Several promising research directions can be explored to address these limitations and further enhance malware detection. One key area is hybrid analysis techniques, which combine static features (e.g., code structure, entropy) with dynamic behavioral graphs. Graph-based fusion methods could integrate multiple detection signals, improving robustness against evasion tactics. Another critical direction is adversarial robustness. Developing adversarial training mechanisms-potentially using generative adversarial networks (GANs) to simulate malicious API call patterns-could harden the model against evasion attacks. Efficiency improvements are also essential. Lightweight graph learning techniques, such as simplified GNN architectures or knowledge distillation, could reduce computational overhead without sacrificing accuracy. Additionally, continuous learning frameworks could help models adapt to evolving threats without catastrophic forgetting. Online learning or memory-augmented networks might enable the retention of historical threat knowledge while incorporating new attack patterns.

Future work should prioritize explainability by improving interpretability using attention visualization, subgraph importance scoring, and rule extraction to help analysts understand model decisions, build trust, and enhance forensic use. Expanding model support to platforms like Android and Linux by adapting to their system call patterns would increase practical impact. Assigning semantic labels to nodes based on observed APIs can further clarify threat context and support deeper behavioral analysis. Research should also explore hybrid GNN architectures, like combining GAT with LSTM, to better capture both structural and temporal patterns. Adversarial testing must assess resilience against obfuscation and polymorphism. Cross-model agreement on high-risk API clusters supports the use of ensemble detection strategies and advanced behavior analysis: these steps will raise detection accuracy and operational reliability.

Finally, real-time detection optimizations should be explored. Streaming graph learning techniques could enable immediate threat response by processing API call sequences on the fly. Integrating large-scale threat intelligence-such as network indicators of compromise (IOCs) and attacker tactics, techniques, and procedures (TTPs)-could further enhance detection capabilities. By addressing these challenges, future iterations of GraphShield could achieve even greater accuracy, efficiency, and practical utility in cybersecurity operations.

9. Conclusion

GraphShield is a graph-based malware detection framework that uses dynamic behavioral analysis to overcome the weaknesses of traditional signature and sequence-based methods. It converts API call sequences into structured graphs, allowing the system to capture both local and global behavioral patterns. This enables robust detection of polymorphic, fileless, and environment-aware malware. The framework introduces a context-aware graph construction method that addresses the limitations of linear analysis. By using BERT embeddings to encode the semantics of API calls, attention mechanisms to focus on critical interactions, and Markov models to standardize sequences, GraphShield preserves both the structural and semantic relationships often missed by conventional approaches. This design improves detection accuracy and reduces false positives.

GraphShield incorporates two hybrid hierarchical GNN models: GIN-CNN and GCN-LSTM autoencoders. GIN-CNN combines Graph Isomorphism Networks with CNNs to model both the structure of the API call graph and localized patterns. GCN-LSTM merges graph convolutions with LSTM units to detect temporal evolution in staged or delayed malware behavior. Both models use attention mechanisms and autoencoder structures to detect anomalies through reconstruction loss. Among them, GIN-CNN achieved the best performance, with an F1-score of 0.9951 and very low false positives. The system was tested on over 500,000 samples, split into 300,000 for training and 200,000 for testing. GraphShield outperformed traditional deep learning models, achieving a 58 % improvement in detection accuracy. GNN-based models maintained false positive rates under 1 %, a critical metric for real-world deployment where operational cost and analyst fatigue are major concerns. By contrast, models like CNN-LSTM and GRU showed a steep drop in performance when exposed to unseen malware, confirming the superior generalization of graph-based approaches. A key feature of GraphShield is its explainability. It includes a GNNExplainer module that highlights the specific subgraphs and features that influenced the model's decisions. This transparency allows security analysts to understand, verify, and act on detection outcomes with greater confidence. The system can identify precise API patterns responsible for malware classification, which helps with compliance, auditing, and faster response. Experimental results reinforce the stability of structural approaches. For instance, CNN's recall dropped from 0.945 during validation to 0.570 in testing. In contrast, GIN-CNN maintained an F1-score of 0.9951 and an AUC of 0.9993, demonstrating resilience against diverse malware types and evasion tactics.

Despite its advantages, GraphShield introduces computational overhead due to the complexity of graph construction and processing. Real-time deployment in high-throughput or resource-limited environments will require further work on model optimization, compression, and scalable inference strategies. It also remains vulnerable to tactics like benign API injection or adversarial graph manipulation, which subtly alter the graph structure without changing malicious intent. To counter these challenges, future directions include adaptive learning techniques such as federated learning for privacy-preserving distributed training and self-supervised learning to reduce dependence on labeled data. Integrating GraphShield with memory forensics or network activity monitoring can further enhance threat correlation across different system layers. GraphShield represents a shift in malware detection strategy. By focusing on structure, context, and adaptability, it provides a scalable and explainable defense model that performs consistently across a wide range of threats. Its ability to maintain performance and interpretability makes it a strong candidate for next-generation security systems.

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the author(s) used ChatGPT to proofread. After using this tool, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

CRediT authorship contribution statement

Eslam Amer: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Visualization; **Shaker El-Sappagh:** Conceptualization, Methodology, Writing – review & editing; **Tamer Abuhamad:** Methodology, Validation, Formal analysis, Investigation, Writing – review & editing; **Bander Ali Saleh Al-Rimy:** Validation, Writing – review & editing; **Alaa Mohasseb:** Validation, Formal analysis, Investigation, Writing – review & editing.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request, and can be provided upon submitting a request through the project's GitHub page at https://github.com/EAMER-79/GraphShield-Project.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Abd-Elaziz, M. M., El-Rashidy, N., Elfetouh, A. A., & El-Bakry, H. M. (2025). Position-context additive transformer-based model for classifying text data on social media (vol. 15). UK London: Nature Publishing Group.

Abdelmotaleb, H., Mcneile, C., & Wojtyś, M. (2025). A comparative study of word embedding techniques for classification of star ratings. Elsevier.

Afianian, A., Niksefat, S., Sadeghiyan, B., & Baptiste, D. (2019). Malware dynamic analysis evasion techniques: A survey. *ACM Computing Surveys (CSUR)*, *52*(6), 1–28.

Agarwal, C., Queen, O., Lakkaraju, H., & Zitnik, M. (2023). Evaluating explainability for graph neural networks. *Scientific data*, *10*(1), 144.

Amer, E. (2023). Identification of malware mimicry attacks using process escalating visualization. In *2023 intelligent methods, systems, and applications (imsa)* (pp. 99–104). IEEE.

Amer, E., Al-Rimy, B. A. S., & El-Sappagh, S. (2025). Strengthening ics defense: Modbus-nfa behavior model for enhanced anomaly detection. *Journal of Information Security and Applications*, *89*, 103990.

Amer, E., Al-Rimy, B. A. S., El-Sappagh, S., Almalki, S., & Alghamdi, T. (2024). From black boxes to transparent insights: Enhancing industrial control systems anomaly detection with deep autoencoder models. In *2024 international mobile, intelligent, and ubiquitous computing conference (miucc)* (pp. 380–386). IEEE.

Amer, E., & Elboghdadly, T. (2024). Evaluating machine learning techniques for ics security: Insights from dataset limitations and classifier performance. In *2024 international mobile, intelligent, and ubiquitous computing conference (miucc)* (pp. 368–373). IEEE.

Amer, E., Samir, A., & Mostafa, H. (2022). Malware detection approach based on the swarm-based behavioural analysis over api calling sequence. In *2022 2nd international mobile, intelligent, and ubiquitous computing conference (miucc)* (pp. 27–32). IEEE. Amer Mohamed, and Mohamed Amin.

Amer, E., & Zelinka, I. (2020). A dynamic windows malware detection and prediction method based on contextual understanding of api call sequence. *Computers & Security*, *92*, 101760.

Amer, E., Zelinka, I., & El-Sappagh, S. (2021). A multi-perspective malware detection approach through behavioral fusion of api call sequence. *Computers & Security*, *110*, 102449.

Anderson, H. S. (2023). EMBER Benchmark.

Andriesse, D., Chen, X., Der, V. V., Veen, A., Slowinska, H., & Bos (2016). An in-Depth analysis of disassembly on full-Scale x86/x64 binaries. In *25th usenix security symposium (usenix security 16)* (pp. 583–600).

Arthur, D., & Vassilvitskii, S. (2007). K-means: The advantages of careful seeding. In *Proceedings of the 18th annual acm-siam symposium on discrete algorithms (soda)* the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) (pp. 1027–1035).

Artuso, F., Mormando, M., Giuseppe, A., Luna, D., & Querzoni, L. (2024). Binbert: Binary code understanding with a fine-tunable and execution-aware transformer.

Bai, J., Zhu, J., Song, Y., Zhao, L., Hou, Z., Du, R., & Li, H. (2021). A3t-Gcn: Attention temporal graph convolutional network for traffic forecasting. *ISPRS international journal of geo-information*, *10*(7), 485.

Bilot, T., Madhoun, N. E., Agha, K. A., & Zouaoui, A. (2024). A survey on malware detection with graph representation learning. *ACM computing surveys*, *56*(11), 1–36.

Bouritsas, G., Frasca, F., Zafeiriou, S., & Bronstein, M. M. (2022). Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE transactions on pattern analysis and machine intelligence*, *45*(1), 657–668.

Ceschin, F., Botacin, M., Heitor, M., Gomes, F., Pinagé, L. S., Oliveira, A., & Grégio (2023). Fast & furious: On the modelling of malware detection as an evolving data stream. *Expert systems with applications*, *212*, 118590.

Ceschin, F., Pinage, F., Castilho, M., Menotti, D., Oliveira, L. S., & Gregio, A. (2018). The need for speed: An analysis of brazilian malware classifiers. *IEEE Security & Privacy*, *16*(6), 31–41.

Chen, C., Li, K., Wei, W., Zhou, J. T., & Zeng, Z. (2021). Hierarchical graph neural networks for few-shot learning. *IEEE Transactions on Circuits and Systems for Video Technology*, *32*, 240–252.

Chen, S., Lang, B., Liu, H., Chen, Y., & Song, Y. (2024a). Android malware detection method based on graph attention networks and deep fusion of multimodal features. *Expert systems with applications*, *237*, 121617.

Chen, T., Zeng, H., Lv, M., & Zhu, T. (2024b). Ctimd: Cyber threat intelligence enhanced malware detection using api call sequences with parameters. *Computers & Security*, *136*, 103518.

Chen, Y.-H., Lin, S.-C., Huang, S.-C., Lei, C.-L., & Huang, C.-Y. (2023). Guided malware sample analysis based on graph neural networks. *IEEE Transactions on Information Forensics and Security*, *18*, 4128–4143.

Costamagna, C. V., Zheng, H., & Huang (2018). Identifying and evading android sandbox through usage-profile based fingerprints. In *Proceedings of the first workshop on radical and experiential security* the first workshop on radical and experiential security (pp. 17–23).

CrowdStrike Intelligence Team, (2023). Global Threat Report. CrowdStrike Inc

Cui, L., Cui, J., Ji, Y., Hao, Z., Li, L., & Ding, Z. (2023). Api2vec: Learning representations of api sequences for malware detection. In *Proceedings of the 32nd acm sigsoft international symposium on software testing and analysis* the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 261–273).

Dabas, N., Sharma, P., & et al. (2023). Malanalyser: An effective and efficient windows malware detection method based on api call sequences. *Expert systems with applications*, *230*, 120756.

Demetrio, L., Coull, S. E., Biggio, B., Lagorio, G., Armando, A., & Roli, F. (2021). Adversarial exemples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection. *ACM Transactions on Privacy and Security (TOPS)*, *24*(4), 1–31.

Demirkıran, F., Çayır, A., Ünal, U., & Dağ, H. (2022). An ensemble of pre-trained transformer models for imbalanced multiclass malware classification. *Computers & Security*, *121*, 102846.

Dowd, M., Mcdonald, J., & Schuh, J. (2006). The art of software security assessment: Identifying and preventing software vulnerabilities. Pearson Education.

Duan, G., Lv, H., Wang, H., Feng, G., & Li, X. (2024). Practical cyber attack detection with continuous temporal graph in dynamic network system. In *Ieee transactions on information forensics and security*.

Feng, B., Wang, Y., & Ding, Y. (2021). Uag: Uncertainty-aware attention graph neural network for defending adversarial attacks. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*, 7404–7412.

Fields, J., Chovanec, K., & Madiraju, P. (2024). A survey of text classification with transformers: how wide? how large? how long? how accurate? how expensive? how safe? (vol. 12). IEEE.

Filippo, M., Bianchi, D., Grattarola, L., Livi, C., & Alippi (2020). Hierarchical representation learning in graph neural networks with node decimation pooling. *IEEE transactions on neural networks and learning systems*, *33*(5), 2195–2207.

Galli, A., Valerio, L., Gatta, V., Moscato, M., Postiglione, G., & Sperlì (2024). Explainability in ai-based behavioral malware detection systems. *Computers & Security*, *141*, 103842.

Gao, Y., Hasegawa, H., Yamaguchi, Y., & Shimada, H. (2022). Malware detection by control-flow graph level representation learning with graph isomorphism network. *IEEE Access*, *10*, 111830–111841.

Gurevin, D., Shan, M., Huang, S., Hasan, C. M. A., Ding, O., & Khan (2024). Prunegnn: Algorithm-architecture pruning framework for graph neural network acceleration. In *2024 ieee international symposium on high-performance computer architecture (hpca)* (pp. 108–123). IEEE.

Institute, A.-T. (2023). Technical Report Malware Statistics Report.

Kara, I. (2023). Fileless malware threats: Recent advances, analysis approach through memory forensics and research challenges. *Expert systems with applications*, *214*, 119133.

Kargarnovin, O., Sadeghzadeh, A. M., & Jalili, R. (2024). Mal2gcn: A robust malware detection approach using deep graph convolutional networks with non-negative weights. *Journal of Computer Virology and Hacking Techniques*, *20*(1), 95–111.

Khoshraftar, S., & An, A. (2024). A survey on graph representation learning methods. *ACM transactions on intelligent systems and technology*, *15*(1), 1–55.

Ki, Y., Kim, E., Kang, H., & Kim (2015). A novel approach to detect malware based on api call sequence analysis. *International Journal of Distributed Sensor Networks*, *11*(6), 659101.

Kim, J.-Y., & Cho, S.-B. (2022). Obfuscated malware detection using deep generative model based on global/local features. *Computers & Security*, *112*, 102501.

Kirat, D., Vigna, G., & Kruegel, C. (2011). Barebox: Efficient malware analysis on bare-metal. In *Proceedings of the 27th annual computer security applications conference* the 27th Annual Computer Security Applications Conference (pp. 403–412).

Koromilas, L., Vasiliadis, G., Athanasopoulos, E., & Ioannidis, S. (2016). GRIM: Leveraging GPUs for kernel integrity monitoring (vol. 19). Paris, France. Springer.

Lagraa, S., Husák, M., Seba, H., & Vuppala, S. (2024). Radu state, and moussa ouedraogo. a review on graph-based approaches for network security monitoring and botnet detection. *International Journal of Information Security*, *23*(1), 119–140.

Li, C., Lv, Q., Li, N., Wang, Y., Sun, D., & Qiao, Y. (2022a). A novel deep framework for dynamic malware detection based on api sequence intrinsic features. *Computers & Security*, *116*, 102686.

Li, Q., Han, Z., & Wu, X.-M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. *Proceedings of the AAAI conference on artificial intelligence*, *32*.

Li, S., Zhou, Q., Zhou, R., & Lv, Q. (2022b). Intelligent malware detection based on graph convolutional network. *The Journal of supercomputing*, *78*(3), 4182–4198.

Li, W., Tang, H., Zhu, H., Zhang, W., & Liu, C. (2024). Ts-mal: Malware detection model using temporal and structural features learning. *Computers & Security*, *140*, 103752.

Ling, X., Wu, L., Deng, W., Qu, Z., Zhang, J., Zhang, S., Ma, T., Wang, B., Wu, C., & Ji, S. (2022). Malgraph: Hierarchical graph neural networks for robust windows malware detection. In *Ieee infocom 2022-ieee conference on computer communications* (pp. 1998–2007). IEEE.

Liu, C., Li, B., Zhao, J., Liu, X., & Li, C. (2023). Malaf: Malware attack foretelling from runtime behavior graph sequence. *IEEE transactions on dependable and secure computing*, *21*(4), 1951–1966.

Liu, W., Luo, X., Liu, Y., & Zhang, D. (2021). Multi-head graph attention networks for malware detection. *Computers & Security*, *110*, 102437.

Liu, Y., Tantithamthavorn, C., Li, L., & Liu, Y. (2022). Explainable ai for android malware detection: Towards understanding why the models perform so well? In *2022 ieee 33rd international symposium on software reliability engineering (issre)* (pp. 169–180). IEEE.

Longa, A., Azzolin, S., Santin, G., Cencetti, G., Liò, P., Lepri, B., & Passerini, A. (2025). Explaining the explainers in graph neural networks: A comparative study. *ACM computing surveys*, *57*(5), 1–37.

Manzoor, E., Sadegh, M., Milajerdi, L., & Akoglu (2016). Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 1035–1044).

MITRE Corporation, (2023). ATT&CK Matrix .

Mohanta, A., & Saldanha, A. (2020). Malware analysis and detection engineering: A comprehensive approach to detect and analyze modern malware. Springer.

Nasereddin, M., & Al-Qassas, R. (2024). A new approach for detecting process injection attacks using memory analysis. *International Journal of Information Security*, *23*(3), 2099–2121.

NIST, SP 800-215 2023.

Or-Meir, O., Nissim, N., Elovici, Y., & Rokach, L. (2019). Dynamic malware analysis in the modern era-a state of the art survey. *ACM Computing Surveys (CSUR)*, *52*(5), 1–48.

Owoh, N., Adejoh, J., Hosseinzadeh, S., Ashawa, M., Osamor, J., & Qureshi, A. (2024). Malware detection based on api call sequence analysis: A gated recurrent unit-generative adversarial network model approach. *Future Internet*, *16*(10), 369.

Parry, H., Xun, L., Sabet, A., Bi, J., Hare, J., & Merrett, G. V. (2021). Dynamic transformer for efficient machine translation on embedded devices. In *2021 acm/ieee 3rd workshop on machine learning for cad (mlcad)* (pp. 1–6). IEEE.

Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J., & Cavallaro, L. (2019). Tesseract: Eliminating experimental bias in malware classification across space and time. In *Proceedings of the 28th usenix security symposium* the 28th USENIX Security Symposium.

Pitropakis, N., Panaousis, E., Giannetsos, T., Anastasiadis, E., & Loukas, G. (2019). A taxonomy and survey of attacks against machine learning. *Computer Science Review*, *34*, 100199.

Ranjani, B., & Chinnadurai, M. (2024). Sparse attention with residual pyramidal depthwise separable convolutional based malware detection with optimization mechanism. *Scientific reports*, *14*(1), 24414.

Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., & Bronstein, M. (2020). Temporal graph networks for deep learning on dynamic graphs. *Proceedings of the 37th International Conference on Machine Learning, series Proceedings of Machine Learning Research*, *119*(PMLR).

Sammouda, R., & El-Zaart, A. (2021). An optimized approach for prostate image segmentation using k-means clustering algorithm with elbow method. *Computational intelligence and neuroscience*, *2021*(1), 4553832.

Schranko, A., Oliveira, D., & Sassi, R. J. (2023). Behavioral malware detection using deep graph convolutional neural networks. Authorea Preprints.

Shaukat, K., Luo, S., & Varadharajan, V. (2023). A novel deep learning-based approach for malware detection. *Engineering applications of artificial intelligence*, *122*, 106030.

Shoaib, M., Akhtar, & Feng, T. (2022). Malware analysis and detection using machine learning algorithms. *Symmetry*, *14*(11), 2304.

Sun, L., Dou, Y., Yang, C., Zhang, K., Wang, J., Yu, P. S., He, L., & Li, B. (2022). Adversarial attack and defense on graph data: A survey. *IEEE transactions on knowledge and data engineering*, *35*(8), 7693–7711.

Tajudeen, A., & Nureni, A. (2025). A Survey On Categorization Of Threat Intelligence And Trust-Based Sharing Strategies On Cyber Attack. Technical Report 2. Vokasi Unesa Bulletin of Engineering.

Tay, Y., Dehghani, M., Bahri, D., & Metzler, D. (2022). Efficient transformers: A survey. *ACM computing surveys*, *55*(6), 1–28.
Velickovic, P., Cucurull, G., Casanova, A., Romero, A., & Lio, P. (2017). Yoshua bengio, et al. graph attention networks. *Stat*, *1050*(20), 10–48550.
Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks.
Virustotal (2023). Technical Report Enterprise Analytics Report.
Wang, X., Li, P., & Zhang, M. (2025). Improving graph neural networks on multi-node tasks with the labeling trick. *Journal of Machine Learning Research*, *26*(23), 1–44.
Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2020). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, *32*, 4–24.
Xu, K., & et al. (2019). How powerful are graph neural networks?
Yan, B., Yang, C., Shi, C., Fang, Y., Li, Q., Ye, Y., & Du, J. (2023). Graph mining for cybersecurity: A survey. *ACM transactions on knowledge discovery from data*, *18*(2), 1–52.
Zhang, X., & Zitnik, M. (2020). Gnnguard: Defending graph neural networks against adversarial attacks. *Advances in neural information processing systems*, *33*, 9263–9275.
Zhang, Z., Li, Y., Wang, W., Song, H., & Dong, H. (2022). Malware detection with dynamic evolving graph convolutional networks. *International Journal of Intelligent Systems*, *37*(10), 7261–7280.
Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., Deng, M., & Li, H. (2019a). T-Gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, *21*(9), 3848–3858.
Zhao, Y., Bo, B., Feng, Y., Xu, C., & Yu, B. (2019b). A feature extraction method of hybrid gram for malicious behavior based on machine learning. Security and Communication Networks.