Vol-4 Issue-1 2026
Scientific Research Journal of Science, Engineering and Technology
ISSN: 2584-0584, Peer Reviewed Journal

Behavioral Drift–Aware Malware Detection: A Survey of Graph Neural Networks for Explainable and Operational Deployment

Yogesh S, M.Tech Computer Science and Engineering, St. peter's Institute of Higher Education and Research, Avadi, Chennai, India projfeb10@gmail.com
Ms. E. Sheela, Assistant professor,Dept of Computer Science and Engineering,St. peter's Institute of Higher Education and Research,Avadi, Chennai, India,Sheela.cse@spiher.ac.in

ABSTRACT

Modern malware uses more and more advanced evasion techniques, like packing, polymorphism, fileless execution, reflective loading, and living-off-the-land behaviors. This makes traditional signature-based and static detection methods much less effective. Dynamic behavioral analysis—recording API and system-call traces, process interactions, memory activities, and network events—offers deeper semantic understanding of runtime malware behavior, yet generates highly structured, noisy, and evolving data. To tackle this issue, graph-based representations such as API-call graphs, control-flow graphs, system dependency graphs, and heterogeneous process-resource graphs have proven to be effective abstractions for modeling malware behavior.Graph Neural Networks (GNNs), which include convolutional, attention-based, temporal, and self-supervised types, have shown that they can find new types of malware and variants that act similarly to existing ones. Nonetheless, behavioral and conceptual drift, adversarial graph manipulation, restricted explainability, scalable graph extraction, and real-time deployment limitations persist as significant obstacles to operational implementation.This survey methodically examines GNN-based malware detection research published from 2020 to 2025, focusing specifically on drift-aware learning, explainable GNN models, and deployment-oriented factors. We classify current research based on graph construction strategies, GNN architectures, self-supervised and contrastive learning methodologies, robustness mechanisms, and evaluation protocols. Lastly, we talk about open research problems and suggest ways to move forward in creating GNN-based malware defense systems that are strong, easy to understand, and can be used in the real world.

Keywords: Graph Neural Networks (GNNs); Malware Detection; Behavioral Drift; Concept Drift; Dynamic MalwareAnalysis; Explainable AI; Adversarial Malware; Graph Learning; Security Analytics

1. INTRODUCTION

Malware is still changing quickly, using more and more advanced ways to avoid detection, like packing, polymorphism, fileless execution, reflective loading, and living-off-the-land strategies. These methods make traditional signature-based and static analysis methods, which depend on known attack patterns and syntactic features, much less effective [1], [9]. Because of this, modern malware detection systems need to think about how things work at runtime instead of how they look in code.As a result, dynamic malware analysis is now an important part of modern defense systems. Dynamic analysis collects semantically rich behavioral information that is less likely to be hidden or packed by looking at execution traces like API and system calls, process hierarchies, memory interactions, and network communications [6], [12]. But these runtime signals are noisy, high-dimensional, and structurally complex by nature, which makes it hard to use standard machine learning techniques to analyze them.Graph-based representations have become a potent abstraction for systematically and

articulately modeling malware behavior. Control-flow graphs (CFGs), API-call graphs, system dependency graphs, and heterogeneous process–resource graphs represent both the structural and causal relationships between execution events, allowing for detailed behavioral reasoning [3], [6], [12]. These graph representations naturally fit with how malware works and provide a solid basis for detection based on learning.Graph Neural Networks (GNNs) have shown to be the best at many graph learning tasks in the last few years, such as finding malware and analyzing intrusions [1], [2], [7]. GNNs can learn discriminative representations from complex malware graphs while keeping topological and semantic information by using message passing and neighborhood aggregation. Graph Convolutional Networks (GCNs) [19], Graph Attention Networks (GATs) [20], Gated Graph Neural Networks (GGNNs) [22], and Graph Isomorphism Networks (GINs) [21] are some examples of variants that have worked well on both static and dynamic malware graphs. More recent extensions include temporal modeling [7], [24], heterogeneous graph learning [8], and self-supervised or contrastive objectives to lessen label dependence in large-scale malware datasets [4], [15].Even with these improvements, behavioral drift is still a major and not well-studied problem for real-world malware detection systems. Behavioral drift is when malware runs differently over time because of things like adversarial evolution, software updates, and changes in the environment, and defensive pressure [11]. When detection models trained on past data are used in changing operational environments, this kind of drift causes performance to go down. Although concept drift has been thoroughly examined in conventional malware classification [11], its interplay with graph-based representations and GNN-based detectors remains inadequately explored. Another major problem that makes it hard for GNN-based malware detectors to be used in real life is that they are not easy to understand. Security analysts need clear and understandable decisions to figure out what happened, check alerts, and help with incident response workflows. Nevertheless, the majority of GNN-based methodologies function as black-box models, offering minimal transparency regarding the specific nodes, edges, or sub graphs that influence a classification decision [3], [16], [28], [29]. Trust in these kinds of systems is even harder because it's hard to trust explanations when behavior changes and people try to trick them. Additionally, operational deployment constraints, such as scalable graph extraction, sandbox fidelity, inference latency, and integration with Security Operations Centers (SOCs), are frequently neglected in academic research [17], [6]. Many GNN-based solutions claim to have high detection accuracy in controlled experiments, but we don't know if they will work in real-time or on a large scale. To deal with these problems, we need to look at them all at once, taking into account learning robustness, explain ability, and system-level deployment issues. While numerous surveys have examined malware detection through graph representations or deep learning methodologies [5], [17], current research does not specifically address the integrated challenges of behavioral drift, interpretable GNN models, and practical implementation. In particular, the lack of a drift-aware perspective makes it hard to use previous surveys to real-world malware defense systems that need to work all the time in hostile and changing environments. What This Survey Adds. To fill this gap, this paper offers a thorough survey of behavioral drift-aware malware detection utilizing Graph Neural Networks, emphasizing explain ability and operational readiness. The primary contributions of this survey are as follows:

We present a structured summary of graph construction methods for modeling malware behavior, encompassing static, dynamic, and hybrid techniques, and evaluate their resilience in the face of behavioral drift. We systematically examine GNN architectures and learning paradigms utilized in malware detection, encompassing convolutional, attention-based, temporal, heterogeneous, and self-supervised models. We look at current explainability methods for GNN-based malware detection, pointing out their pros, cons, and how well they work when conditions change or when there are attacks. We look at adversarial attacks and ways to make graph-based malware detectors more robust. We talk about the problems that come up when deploying operations, such as scalability, real-time inference, continuous learning, and SOC integration. We pinpoint open research challenges and prospective pathways for developing resilient, interpretable, and deployable GNN-based malware defense systems.

This survey aims to serve as a reference for researchers and practitioners who want to design next-generation malware detection systems that are not only accurate, but also strong, easy to understand,

and useful in real life. It does this by bringing together recent advances from 2020 to 2025.

2. BACKGROUND AND MOTIVATION

A. How Malware Behaviors Have Changed

Over the past ten years, the malware landscape has changed quickly and constantly, mostly because attackers have adapted to defensive technologies. Signature-based antivirus systems worked well against early malware, which used relatively simple signatures and static payloads. As a result, modern malware uses more advanced evasion methods like packing, encryption, polymorphism, metamorphism, reflective code loading, fileless execution, and living-off-the-land (LotL) strategies that abuse legitimate system utilities to hide its true purpose [1], [9], [26]. These methods make it much harder to find malicious artifacts in static binaries and make it harder to use traditional feature extraction methods.Also, modern malware families often have modular architectures, delayed execution logic, behavior that changes based on the environment, and command-and-control (C2) interactions that only happen when certain conditions are met [6], [17]. Consequently, malware behavior is no longer static or predictable; it fluctuates across execution contexts, operating systems, and deployment environments. This variety in behavior directly leads to behavioral drift, which is when detection patterns that were learned in the past become useless over time.

B. Static, Dynamic, and Hybrid Malware Analysis

There are three main types of malware analysis techniques: static, dynamic, and hybrid. Static analysis looks at program binaries without running them and pulls out things like opcode sequences, imported libraries, control-flow graphs (CFGs), and byte-level representations [9], [10]. Static methods are fast and can be used on a large scale, but they are very easy to hide, pack, and encrypt, which is common in modern malware [1].Dynamic analysis looks at how a program works while it is running in a safe place, like a sandbox or a virtual machine. It records things that happen while a program is running, like API and system-call sequences, process creation events, file-system access, registry changes, memory interactions, and network traffic [6], [12], [17]. Dynamic analysis gives us a deeper understanding of malicious intent and is better at dealing with static evasion techniques. But it has problems like limited execution coverage, sandbox evasion, noisy behavioral traces, and a lot of extra work for the computer.The goal of hybrid analysis is to use both static and dynamic methods to take advantage of their strengths. Hybrid methods can make static representations better by adding runtime behavior graphs or dynamic execution traces to static feature spaces. This makes them more robust and accurate at finding things [6], [26]. Graph-based modeling has become an intrinsic foundation for hybrid analysis, facilitating the simultaneous representation and learning of diverse information sources.

C. Behavioral and Concept Drift in Malware Detection

Behavioral and concept drift is a big but often ignored problem in malware detection. It means that the statistical properties and semantic meaning of malware behavior change over time. Adversarial evolution, software updates, changes in operating environments, and defensive pressure from detection systems all cause behavioral drift [11]. Concept drift happens when the relationship between the features that were seen and the class labels changes. This makes models trained on old data less accurate. In real-world situations, malware detectors run all the time for long periods of time, during which new families of malware appear and existing families change how they act to avoid being found [11], [13]. Graph-based representations are especially vulnerable to drift because even small changes in execution logic or interaction patterns can have a big effect on the structure and topology of the graph. Even though drift is common in operational settings, most current malware detection methods assume that the data distribution is stable and test performance in static experimental conditions.

D. Limitations of Existing Detection Approaches

When faced with changing adversarial behavior, traditional machine learning and deep learning-based malware detection systems have a number of problems. Signature-based systems and static classifiers do not work with new types of malware and can be easily avoided by using obfuscation and packing [1]. Deep learning models that are trained on static or sequential features often break easily, are hard to understand, and are sensitive to changes in the data [5]. Graph-based models and Graph Neural Networks (GNNs) exhibit enhanced expressive capabilities and detection precision [1, 2, 7]; however, they present novel challenges. These encompass scalability challenges in graph extraction, susceptibility to adversarial graph manipulation [14], restricted explainability [3], [16], and deterioration under behavioral drift [11]. Additionally, numerous GNN-based methodologies are assessed in regulated laboratory settings and fail to sufficiently consider real-time limitations, sandbox accuracy, or integration with operational security frameworks [6], [17].

E. Research Gaps Addressed by This Survey

Several surveys have looked at using deep learning or graph-based methods to find malware [5], [17], but none of them have looked specifically at the problems that come up when you combine behavioral drift, explainable graph learning, and operational deployment. The relationship between behavioral drift and GNN-based malware detection is still not well understood, and the reliability of explainability methods when adversarial conditions change is also not well understood. This survey aims to fill in these gaps by giving a full and organized overview of how to use Graph Neural Networks to find malware that is aware of behavioral drift. We focus on explainability, robustness, and deployment-oriented factors, giving a single view that connects theoretical progress with real-world needs. This work brings together recent research from 2020 to 2025 to help create next-generation malware detection systems that are resistant to drift, easy for analysts to understand, and work well in real-world situations.

3. Malware Behavior Graph Construction Techniques

Graph construction is an important part of graph-based malware detection because it decides how Graph Neural Networks (GNNs) abstract, structure, and learn from program behavior. How well a GNN-based detector works depends a lot on how well the malware behavior graph can express, be strong, and grow. This part talks about static, dynamic, and hybrid graph construction methods. Then it talks about graph granularity and feature encoding, with a focus on how well they work when behavior changes.

A. Static Graph Construction

Static graph construction techniques get graph representations directly from program binaries without running them. These methods are fast and can be used on a large scale, which makes them good for large-scale malware analysis pipelines. Control-Flow Graphs (CFGs) show how a program's logic works. Each node is a basic block, and each edge shows a possible control-flow transition [3], [17]. CFGs help us think about the logic and complexity of a program's structure. They are often used in static malware analysis and finding similarities. But CFG-based representations are very sensitive to techniques that modern malware often uses to hide code, pack it, and flatten control flow [1], [9]. Call graphs show how functions interact with each other by showing functions as nodes and invocation relationships as edges. Call graphs are more abstract than CFGs and can show how different procedures interact with each other [17]. Indirect calls, dynamic loading, and reflection can all make call graphs much less complete and accurate, which makes them less reliable in adversarial settings. Static graphs are useful and easy to understand, but they aren't as good at adapting to changing malware tactics and behavioral drift because they can't show how things change over time.

B. Dynamic Graph Construction

Dynamic graph construction techniques simulate malware behavior by monitoring runtime execution in sandboxed or virtualized environments. These methods gather semantically rich behavioral data and are less likely to be fooled by static evasion methods. API-call graphs show API calls at runtime as nodes, with edges showing the order in which they happened or how they depend on each other [1], [6]. These graphs do a good job of showing how systems interact with each other at a high level, and they have been very good at finding new and behaviorally similar malware variants. But API-call graphs can be noisy and affected by things like the version of the operating system and the configuration of the sandbox. By modeling dependencies between system calls based on shared resources or data flows [12], [27], system-call dependency graphs give us a lower-level view of how things work. These graphs give you a lot of detail about bad actions, and malware has a harder time getting around them without changing how it works. Their main problem is that they can't be scaled up, since long execution traces can make graphs that are big and complicated. Process-resource interaction graphs enhance dynamic modeling by integrating various entity types, such as processes, files, registry keys, memory objects, and network sockets [8]. Typed interactions like read, write, execute, or communicate are shown by edges. These kinds of heterogeneous graphs give a complete picture of how malware works and are great for dvanced GNN architectures. However, they make graph building and feature engineering more complicated. Dynamic graph-based methods are usually stronger than static ones, but they can still be tricked by sandbox evasion, incomplete execution coverage, and runtime noise [17].

C. Hybrid Graph Construction

Hybrid graph construction techniques combine static and dynamic data to make the most of their strengths. These methods try to make systems more resistant to obfuscation while keeping their ability to grow and their behavior accurate. Hybrid methods can add dynamic annotations to static graphs or combine static structural features with runtime interaction graphs [6], [26]. For instance, you can add dynamic API-call frequencies to static CFG nodes, or you can use static control-flow information to limit dynamic graphs. This kind of integration makes semantic modeling more complex and makes it easier to generalize to new types of malware. Hybrid graphs are especially useful when it comes to being able to handle drift. Static features provide structural stability, while dynamic features capture changing behavior patterns, which helps models adapt better to changes in behavior and concepts [11]. As a result, operational malware detection systems are using hybrid graph representations more and more.

D. Graph Granularity and Feature Encoding

Graph granularity is the level of detail at which malware behavior is modeled. It can be fine-grained system-call graphs or coarse-grained function or process-level graphs. Fine-grained graphs give you a lot of information about behavior, but they are hard to scale and take a lot of computing power. Coarse-grained graphs, on the other hand, make things more efficient but less expressive [6], [12]. Feature encoding has an effect on how well detection works and how strong it is. Node and edge attributes can consist of call frequencies, temporal sequences, argument classifications, access rights, resource identifiers, and contextual metadata [1], [7]. Temporal encoding is very important for modeling behavior that changes over time and for supporting drift-aware learning in dynamic GNN architectures [24]. Choosing the right graph granularity and feature representations means finding a balance between expressiveness, robustness, and deployment needs. Badly chosen abstractions can make noise louder, make drift more sensitive, or make it harder to deploy in real time

Graph Type	Analysis Type	Nodes	Edges	Strengths	Limitations	Drift Robustness
Control-Flow Graph (CFG)	Static	Basic blocks	Control-flow transitions	Structural insight, scalable	Vulnerable to obfuscation	Low
Call Graph	Static	Functions	Invocation relationships	High-level abstraction	Incomplete under dynamic loading	Low
API-Call Graph	Dynamic	API calls	Temporal / dependency	Semantically rich behavior	Noisy, environment-dependent	Medium
System-Call Dependency Graph	Dynamic	System calls	Data/resource dependency	Fine-grained behavior capture	Scalability challenges	Medium–High
Process–Resource Graph	Dynamic	Processes & resources	Typed interactions	Holistic behavior modeling	High complexity	High
Hybrid Graph	Static + Dynamic	Multi-level	Multi-type	Robust and expressive	Construction overhead	**High**

4. GNN TECHNIQUES FOR MALWARE DETECTION

Graph Neural Networks (GNNs) are a structured way to learn from graph-structured malware representations by modeling node attributes, edge relationships, and the overall shape of the graph at the same time. GNNs learn task-specific representations through iterative message passing and aggregation, which is different from traditional machine learning models that rely on handcrafted features. This makes them perfect for complex malware behavior graphs [18], [19]. This part looks at the main GNN architectures used to find malware and talks about their pros and cons as well as how well they work when behavior changes.

A. Graph Convolutional Networks (GCN)

Graph Convolutional Networks apply convolutional operations to non-Euclidean graph domains by consolidating normalized features from adjacent nodes [19]. Graph-level classification tasks like API-call graphs, control-flow graphs, and system-call dependency graphs have used GCNs a lot for malware detection [1], [6].

For large malware datasets, GCNs are a good choice because they are fast to compute and not too hard to set up. They accurately capture local structural patterns and have shown strong baseline performance in spotting bad behavior. But standard GCNs depend on fixed aggregation schemes and

assume that the data distribution is mostly stable, which makes them less reliable when behavior and concept drift [11] occur. Also, GCNs may have trouble modeling different types of nodes and edges without adding to their architecture.

B. Graph Attention Networks (GAT)

Graph Attention Networks use attention mechanisms in neighborhood aggregation to give different weights to neighboring nodes based on how relevant they are [20]. This adaptive
Weighting is especially useful for malware graphs because not all interactions lead to bad behavior in the same way.

In heterogeneous and noisy settings, like process–resource interaction graphs and dynamic behavior graphs [8], GAT-based models have outperformed GCNs. Attention mechanisms also help with interpretability by focusing on important nodes and edges. However, GATs necessitate increased computational resources and may exhibit susceptibility to adversarial manipulation of attention weights, particularly in dynamic threat landscapes [14].

C. Graph Isomorphism Networks (GIN)

Graph Isomorphism Networks are built to get the most expressive power out of message-passing GNNs by closely following the Weisfeiler–Lehman graph isomorphism test [21]. GINs are especially good at telling apart malware graphs that look similar but act in slightly different ways.

When it comes to finding malware, GINs have been used for graph classification tasks like CFGs and API-call graphs. They are better at telling the difference between different types of graphs than simpler architectures [7]. But the fact that GINs can express more often means that they are more sensitive to noise and drift, which means that careful regularization and strong training strategies are needed in operational settings.

D. Gated Graph Neural Networks (GGNN)

Gated Graph Neural Networks use recurrent gating mechanisms like Gated Recurrent Units (GRUs) in message passing to find long-range dependencies and sequential behavior [22]. Because of this, GGNNs are great for modeling execution flows and dependency chains in graphs that show how malware behaves.

GGNNs have worked well on system-call dependency graphs and control-flow representations, where the order of events and cause-and-effect relationships are very important [3], [12]. But because they happen over and over again, they make the computations more difficult and the time it takes to make inferences longer, which could make real-time detection systems less scalable.

E. Temporal and Dynamic GNNs

Temporal and dynamic GNNs enhance conventional architectures by explicitly representing graph structures that evolve over time. These models keep track of how node attributes, edge relationships, and graph topology change over time, which makes them very useful for studying how malware behaves over time [7], [24].

Temporal GNNs have been utilized for sequence-based API-call graphs and dynamic interaction graphs, facilitating enhanced identification of malware variants whose behavior develops progressively during execution. Temporal modeling is important because it lets models change as behavior patterns change, which helps drift-aware learning. Even though temporal GNNs have some benefits, they need to be carefully planned so that they can be expressive while still being efficient with memory and processing power.

F. Comparative Analysis of GNN Architectures

Different GNN architectures have different trade-offs when it comes to how expressive, robust, scalable, and easy to understand they are. Attention-based and isomorphism-based models are better at telling the difference between things, but they cost more to compute. GCNs are good at providing a baseline level of performance. Temporal and gated architectures make modeling even better, but they are hard to use in real-time environments.

From a behavioral drift point of view, architectures that use attention mechanisms, temporal modeling, or adaptive learning objectives are usually better able to deal with malware that changes its behavior [11], [24]. However, no one architecture works better than all the others in all situations. This shows how important it is to choose GNN models based on the type of graph, the limitations of the deployment, and the changing nature of threats.

5. LEARNING PARADIGMS IN GNN-BASED MALWARE DETECTION

Learning paradigms are very important for figuring out how adaptable, strong, and scalable GNN-based malware detection systems are. Because malware behaviors change all the time because of obfuscation, polymorphism, and adversarial adaptation, learning strategies need to be able to deal with limited labels, distribution shifts, and concept drift. This section looks at the different types of supervised, semi-supervised, self-supervised, and continual learning that are used in graph-based malware detection. It focuses on how well they work for drift-aware and operational deployment.

A. Supervised Learning

Supervised learning is still the most common way to use GNNs to find malware. In this method, models are trained on both good and bad samples that have been labeled. In this context, graph representations obtained from static or dynamic analysis serve as inputs to graph-level classifiers [1], [10].
Supervised GNNs have shown great results on benchmark datasets because they can learn representations that are specific to known types of malware. But their dependence on large, well-labeled datasets makes them hard to scale and hard to use in the real world. Also, supervised models usually assume that the data distribution stays the same, which makes them easy targets for behavioral and concept drift as malware authors come up with new ways to get around [11]. To keep performance up, you have to retrain often, which can be expensive and hard to do.

B. Semi-Supervised Learning

Semi-supervised learning solves the problem of not having enough labeled malware samples by using a small set of labeled graphs and a larger set of unlabeled data. GNNs are especially good for semi-supervised settings because they can spread label information across graph structures [19].
Semi-supervised GNNs have been used for graph node classification and graph-level tasks in malware detection. This makes it easier to generalize when there isn't much labeled data [6]. This approach cuts down on the cost of annotating data and lets models take advantage of structural similarities between malware samples. Still, semi-supervised methods are still sensitive to label noise and changes in distribution, which can hurt performance over time as behavior changes.

C. Self-Supervised and Contrastive Learning

Self-supervised learning has become a promising approach for detecting malware, especially when there isn't much labeled data available or when it quickly goes out of date. These methods learn representations by completing pretext tasks like guessing masked nodes, rebuilding graph structures, or telling the difference between different views of the same graph [4], [13].
Graph contrastive learning methods make positive and negative graph pairs by carefully adding to the graphs. This lets models learn invariant representations that show semantic behavior instead of just surface patterns. Self-supervised GNNs have shown better resistance to obfuscation and better ability

to generalize to new malware families in malware detection [4], [9]. Self-supervised methods provide a scalable solution for drift-resilient malware detection by separating representation learning from downstream classification.

D. Continual and Drift-Aware Learning

Continual learning paradigms deal with behavioral and concept drift by letting models slowly adjust to new malware behaviors without losing everything they have learned. When it comes to GNN-based malware detection, continual learning strategies include incremental graph updates, rehearsal-based training, and regularization techniques that keep knowledge that has already been learned [11], [24]. When drift-aware learning mechanisms see big changes in feature distributions or graph structures, they adapt the model. Temporal GNNs and online learning frameworks help models keep changing in environments where threats are always changing. Even though they hold a lot of promise, it is still hard to design stable and scalable continual learning systems because data distributions change, feedback is limited, and people try to manipulate them.

E. Comparative Discussion

Different learning paradigms offer different trade-offs between accuracy, flexibility, and how hard it is to use. Supervised learning excels in addressing known threats; however, self-supervised and continual learning paradigms are more appropriate for long-term implementation in dynamic environments. Combining contrastive pretraining with supervised fine-tuning has become a good way to balance performance and robustness [4], [10].

From an operational standpoint, learning paradigms that reduce reliance on labeled data and facilitate adaptive updating are crucial for the large-scale implementation of GNN-based malware detection systems.

6. Explainable AI (XAI) for Graph-Based Malware Detection

As graph neural networks get more complicated, their black-box nature makes it hard to use them in environments where security is important. When it comes to finding malware, it's hard to trust, audit, and use decisions made by opaque models, especially when false positives can mess up normal system operations. Explainable AI (XAI) techniques try to fill this gap by giving analysts understandable information about model predictions, so they can see why a sample is labeled as malicious. This section looks at techniques for making graph-based malware detection easier to understand and talks about how they can help with behavioral drift and operational deployment.

A. Node, Edge, and Subgraph Attribution

Attribution-based explainability methods find the nodes, edges, or subgraphs that had the biggest impact on a model's prediction. In malware behavior graphs, such elucidations may underscore pivotal API calls, system calls, or process-resource interactions that contribute to malicious classification [3], [8].

Node-level attribution gives each node an importance score, and edge-level explanations show important paths of interaction. Subgraph-level attribution is very useful for analyzing malware because it can find small patterns of behavior that are linked to bad actions like privilege escalation or network exfiltration [18]. These localized explanations help analysts understand things better and help with forensic investigations.

B. Graph Reduction and Summarization

The goal of graph reduction techniques is to make complex malware graphs easier to read while keeping their meaning. Reduced graphs give short explanations that are easier to understand and check [8] by getting rid of nodes and edges that aren't needed.

Graph summarization helps security analysts quickly find chains of bad behavior in operational settings without having to look at full execution traces. Also, reduced graphs can make explanations more stable by cutting down on noise caused by benign or environment-specific behaviors. This is especially important when behaviors are changing.

C. Attention-Based Explanations

GNN architectures have built-in attention mechanisms that naturally make them easier to understand by giving weights to nodes or edges during message passing. Graph Attention Networks (GATs) and similar models let you see attention scores, which show which parts of the malware graph affect the final prediction [20].

In malware detection, attention-based explanations can bring attention to important execution paths or interactions between resources. Recent studies, however, have demonstrated that attention weights do not consistently align with causal significance, thereby eliciting concerns regarding the reliability and consistency of explanations [18]. This limitation requires supplementary explanatory methodologies and stringent validation.

D. Stability and Consistency of Explanations

Stability of explanations means that they stay the same even when the input data or model parameters change slightly. Unstable explanations can make analysts less likely to trust malware detection and make it harder to use in the long term [18].

Behavioral drift makes it even harder to explain things because the way malware behaves may change over time, which could change how attribution works. Recent studies stress the importance of explanation methods that work well even when things change over time or when someone tries to mess with them. To keep trust in drift-aware malware detection systems, we need stability-aware explanation frameworks.

E. Role of XAI in Security Operations

Explainability is not just a feature that makes things clear; it's a necessary part of operational malware detection systems. XAI techniques help security operations by making it easier to follow rules and audits, speeding up incident triage, and making false-positive analysis better.
Explainable models help analysts see how malware behavior changes over time and tell the difference between real threats and harmless behavioral changes. Because of this, XAI is very important for putting GNN-based malware detectors into real-world Security Operations Centers (SOCs) and Endpoint Detection and Response (EDR) platforms.

7. Adversarial Attacks & Robustness in GNN-Based Malware Detection

Graph-based malware detection systems work in environments that are naturally hostile, where attackers change the way malware behaves to avoid being found. As Graph Neural Networks (GNNs) become a key part of advanced malware analysis pipelines, it's important to know their weaknesses and build strong defenses so they can be used safely. This part looks at adversarial threat models that target GNN-based malware detectors and talks about ways to make them more robust against attacks that try to get around, change, or poison them.

A. Adversarial Threat Models

Adversarial attacks on GNN-based malware detection systems can be grouped into broad categories based on what the attacker can do and what they want to do. Evasion attacks, in which malware

behavior is changed at inference time to avoid detection, and poisoning attacks, in which malicious samples are added during training to corrupt the learned model [17], are two common threat models.

In graph-based environments, attackers can change the properties of nodes, add nodes that look harmless, or change the structure of the graph while keeping the malicious functionality intact. These attacks are especially hard to stop because they take advantage of how flexible graph representations are and how GNNs depend on structural and relational information.

B. Graph Manipulation and Evasion Attacks

Graph manipulation attacks try to change malware behavior graphs in ways that trick GNN classifiers without changing how the malware is supposed to work. Some examples are adding extra API calls, changing the order of system calls, or adding safe control-flow paths that make bad patterns less clear [17].

Recent research has shown that even small, well-planned changes to control-flow graphs or system-call dependency graphs can make detection much less effective [17]. These kinds of attacks are very similar to behavioral drift because using the same evasion techniques over and over again can slowly change the behavior distributions of malware, making it even harder to find.

C. Adversarial Training Strategies

Adversarial training makes models more robust by exposing them to samples that have been changed in a hostile way during training. In the realm of GNN-based malware detection, this could entail creating perturbed graphs that mimic authentic evasion strategies and integrating them into the training regimen [14].

GNNs can be more resistant to changes in structure and features if they learn to classify both original and adversarial graphs. Adversarial training, on the other hand, raises the cost of computation and may fit too closely to certain attack patterns, making it harder to generalize to new adversarial strategies.

D. Robust GNN Architectures

Architectural changes can make a model more robust by making it less sensitive to changes that are meant to hurt it. Attention regularization, edge-weight normalization, and hierarchical graph modeling are some of the methods that have been suggested to lessen the effects of bad graph changes [20], [22].

Temporal and dynamic GNNs make systems even more robust by modeling how behavior changes over time. This makes it harder for attackers to avoid detection by making small, temporary changes. Still, architectures that focus on robustness must find a way to balance expressiveness and efficiency in order to be useful in real time.

E. Certified and Provable Defenses

Certified defenses seek to offer formal assurances regarding model behavior in the presence of constrained adversarial perturbations. In graph-based malware detection, these methods try to make sure that the results of classification stay the same when graph structures or attributes are changed within certain limits [18].

Certified defenses for GNNs are still in the early stages of development, but they are promising. However, they have trouble scaling up when they are used on large, diverse malware graphs. Still, they are an important area of research for finding malware with high assurance in hostile environments.

F. Discussion: Strength in the Face of Behavioral Drift

Adversarial attacks and behavioral drift are closely related. Repeated attempts to avoid detection may cause gradual shifts in the distribution that look like natural drift, making it hard to tell the difference between adversarial manipulation and benign evolution. To stay effective over time, strong GNN-based detectors must have both adversarial awareness and drift-adaptive learning systems.

Combining robustness strategies with continuous and self-supervised learning paradigms presents a promising approach to developing resilient malware detection systems that can function in dynamic threat environments.

8. Datasets, Benchmarks, and Evaluation Protocols

The choice of datasets, benchmarking strategies, and evaluation protocols has a big effect on how reliable and reproducible malware detection research is. These things are especially important for graph-based malware detection with GNNs because there are so many different ways to represent graphs, malware behaviors change over time, and there is behavioral drift. This part looks at datasets, benchmarking methods, and evaluation metrics that are often used, with a focus on how well they work for malware detection systems that are aware of drift and can be used in real life.

A. Public Malware Datasets

Publicly accessible malware datasets establish a basis for assessing and contrasting GNN-based detection methodologies. Many people have used static malware datasets like EMBER and DREBIN to test machine learning models [14, 15]. EMBER looks at static PE features, while DREBIN looks at Android malware using features based on permissions and APIs. These datasets are useful for making baseline comparisons, but they don't show how things change over time or how they behave at runtime.

Dynamic analysis datasets obtained from sandbox execution logs facilitate the creation of API-call graphs, system-call dependency graphs, and process-resource interaction graphs [1], [6], [12]. These datasets better show how malware works in the real world, but they often have problems with limited execution coverage, environmental bias, and a lot of noise. Recent studies have underscored the necessity for datasets that reflect long-term behavioral evolution to facilitate drift-aware evaluation [9].

B. Benchmarking Strategies

When comparing GNN-based malware detection models, you need to think carefully about the type of graph, the learning paradigm, and the threat model. Some common ways to benchmark are classifying malware families, classifying binaries as malicious or benign, and finding anomalies [10].

Temporal splits, continual learning benchmarks, and behavioral drift are becoming more popular ways to test robustness. In these cases, models are trained on old data and tested on newer samples in the order they were collected, which is similar to how they would be used in the real world [11]. Cross-dataset evaluation tests models trained on one dataset against samples from a different source or environment to see how well they generalize.

To find the best balance between accuracy, scalability, and robustness, it is important to compare GNN architectures and learning paradigms.

C. Evaluation Metrics and Protocols

Common ways to measure how well malware detection works are accuracy, precision, recall, F1-score, and area under the ROC curve (AUC). However, these metrics alone may not fully reflect operational performance, especially in datasets with significant imbalance where false positives incur considerable costs.

For graph-based and drift-aware detection systems, other metrics like detection latency, model stability over time, and explanation consistency are becoming more important [18]. Robustness assessment may also include the quantification of performance decline in response to adversarial perturbations or simulated evasion attacks [17].

It is very important to have clear evaluation protocols, such as repeated experiments, ablation studies, and tests for statistical significance, to make sure that studies can be fairly compared and repeated.

Dataset	Analysis Type	Graph Support	Temporal Coverage	Limitations
EMBER	Static	No	Limited	No runtime behavior
DREBIN	Static (Android)	Partial	Limited	Outdated malware
Sandbox API Traces	Dynamic	Yes	Medium	Environment bias
System-Call Logs	Dynamic	Yes	Medium–High	Scalability issues
Hybrid Datasets	Static + Dynamic	Yes	High	Limited availability

9. OPERATIONAL DEPLOYMENT CONSIDERATIONS

In research settings, GNN-based malware detection methods work very well. However, using them in real-world security settings comes with more problems and limitations. When putting something into use, you have to think about how to detect it in real time, how reliable the sandbox is, how much extra processing power it needs, and how well it works with existing security systems. This part talks about important operational factors that affect how well GNN-based malware detection systems work in real-world settings.

A. Limitations on Real-Time Detection

To stop threats that spread quickly from doing damage, operational malware detection systems often need inference that happens almost in real time or with low latency. But building a graph and using GNN can take a lot of computing power, especially for big dynamic behavior graphs [10].

Fine-grained system-call or process-resource graphs may yield extensive behavioral insights, yet they result in considerable processing delays. To solve this problem, many practical systems use coarse-

grained graph representations, incremental graph updates, or early-exit classification strategies that let them find problems quickly while still being accurate [12, 24].

B. Sandbox Evasion and Trustworthiness

Sandbox environments are very important for dynamic malware analysis because they let you see how the malware behaves while it's running. Modern malware often uses sandbox evasion techniques like environment fingerprinting, delayed execution, and trigger-based payload activation to hide bad behavior during analysis [16]. When a sandbox runs at a low fidelity, it can create behavior graphs that are incomplete or misleading, which makes GNN-based detection less effective. To make behavior graphs that are reliable enough to use [1], [17], you need to make the sandbox more realistic, add multi-stage execution, and combine sandbox observations with endpoint telemetry.

C. Extra work for computers and ability to grow

Scalability is still a big problem when it comes to using GNN-based malware detectors on a large scale. Building graphs, extracting features, and passing messages all add a lot of computational and memory overhead, especially when there are a lot of samples to process [6].

To make these problems easier to deal with, deployment-oriented solutions use graph pruning, feature compression, model distillation, and hardware acceleration. We also look at lightweight GNN architectures and batching strategies to find a balance between detection performance and limited resources [20], [22].

D. Working with SOC and EDR systems

To be useful, GNN-based malware detection systems need to work well with current Security Operations Center (SOC) workflows and Endpoint Detection and Response (EDR) platforms. For integration to work, there need to be standard output formats, alerts that can be acted on, and explanations that security analysts can easily understand [8], [18]. Explainability mechanisms are very important for deployment because they let analysts check alerts, look into incidents, and respond in the right way. Additionally, feedback from SOC analysts can be used to support ongoing learning and drift-aware adaptation, which will make the system more effective in the long run [11].

E. Deployment under Behavioral Drift

Behavioral drift is a constant problem in operational environments because malware behaviors change all the time. If you don't update or change static deployment models, they might get worse over time. So, to keep detection accuracy high, drift-aware monitoring, periodic retraining, and online learning mechanisms are all necessary [11], [24].

To avoid false alarms or unexpected model behavior, operational deployment needs to find a balance between being flexible and being stable. For long-term use in the real world, systems that combine ongoing learning with decision-making that can be explained are better.

10. OPEN RESEARCH CHALLENGES AND FUTURE DIRECTIONS

Even though Graph Neural Networks (GNNs) have made great strides in detecting graph-based malware, there are still some open research questions that make it hard to use them reliably, on a large scale, and for a long time. To make malware detection systems that can handle behavioral drift, adversarial manipulation, and operational limits, we need to deal with these problems. This part lists important open problems and suggests some good areas for future research.

A. Building a Scalable Graph

Building graphs quickly is still a big problem, especially for dynamic and hybrid malware representations. Fine-grained behavior graphs that come from system calls or interactions between processes and resources often get very big very quickly, which uses a lot of memory and processing power [6], [12].
Researchers should look into adaptive graph construction strategies that change the level of detail based on the situation and the level of threat. To make things more scalable while keeping their meaning, we also need automated noise filtering and graph summarization methods. Standardized graph extraction pipelines would also make it easier to reproduce results and compare studies [9].

B. Robustness against attacks

Adversarial attacks on GNN-based malware detectors are still a real and growing threat. Adversarial training and strong architectures have shown promise, but many current defenses don't work well against attack strategies that haven't been seen before [17], [18].

Future directions involve the creation of certified robustness guarantees specifically designed for malware behavior graphs and the development of threat-aware GNN architectures that integrate domain constraints. Combining adversarial robustness with drift-aware learning methods may make systems even more resilient when they are used for a long time.

C. Trust and Explainability

Even though explainable AI techniques have made it easier to understand AI, making sure that explanations stay stable and consistent even when behavior changes is still a problem [18]. Analysts may lose trust and not use the system if the explanations are not clear or are wrong.Future work should focus on explanation methods that fit with how people think and how security analysts do their jobs. To make malware detection systems that people can trust, we need to do quantitative research on the quality of explanations and long-term studies on how people explain things when threats change [8].

D. Deployment in Real Time and at the Edge

Because of the high computational and memory needs, it is still hard to use GNN-based malware detection models in real time or on devices with limited resources, like endpoints or edge devices [10].

Research endeavors ought to investigate lightweight GNN architectures, model compression methodologies, and hardware-aware optimization strategies. Event-driven and incremental inference mechanisms could enhance low-latency detection while ensuring resilience to changing behaviors [24].

E. Variety in datasets and generalization

Malware datasets that are already out there often don't have enough variety, have old samples, or are biased toward certain environments. These constraints impede generalization and diminish confidence in practical performance [14], [15].

Future research should focus on developing extensive, longitudinal datasets that document the changing behaviors of malware across various platforms, environments, and temporal contexts. To move the field forward, we also need benchmarking protocols that clearly test behavioral drift and cross-domain generalization [11].

F. Working with Threat Intelligence and SOC Feedback

Many current malware detection systems that use GNN work alone and don't use information about threats from outside sources or feedback from analysts. Combining threat intelligence feeds, rule-based

knowledge, and feedback from the SOC can help you be more aware of your surroundings and adapt [16].
To make detection more accurate and cut down on false positives, future systems should include human-in-the-loop learning and model updates based on feedback. This kind of integration will be necessary to create malware defense ecosystems that can change and work together.

10. CONCLUSION

This survey presented a comprehensive review of graph-based malware detection techniques leveraging Graph Neural Networks (GNNs), with a particular focus on behavioral drift, explainability, adversarial robustness, and operational deployment. As modern malware increasingly adopts obfuscation, polymorphism, and adaptive execution strategies, traditional static and signature-based approaches have become insufficient. Graph-based representations, including static, dynamic, and hybrid behavior graphs, offer a powerful abstraction for capturing the structural and semantic characteristics of malware behavior.

We systematically examined malware behavior graph construction techniques, state-of-the-art GNN architectures, and diverse learning paradigms ranging from supervised to self-supervised and continual learning. Special emphasis was placed on explainable AI (XAI) methods, which are essential for building trust and enabling effective integration of GNN-based detectors into real-world security operations. Furthermore, we analyzed adversarial attacks targeting graph-based detectors and reviewed robustness-enhancing strategies necessary for sustaining long-term effectiveness under evolving threat landscapes.

Through a detailed discussion of datasets, evaluation protocols, and deployment considerations, this survey highlighted the gap between academic research and practical deployment. Finally, we identified key open research challenges and outlined future directions aimed at building scalable, interpretable, and resilient malware detection systems capable of adapting to behavioral drift and adversarial manipulation. We hope this survey serves as a valuable reference and roadmap for researchers and practitioners working toward operationally deployable GNN-based malware defense solutions.

REFERENCES

[1] C. Li, X. Zhang, Y. Wang, and J. Liu, “DMalNet: Dynamic malware analysis based on API features and graph neural networks,” *Computers & Security*, vol. 114, Art. no. 102580, 2022.

[2] J. Busch, L. Payer, and K. Rieck, “Network flow graph neural networks for malware detection,” in *Proc. Int. Conf. on Detection of Intrusions and Malware*, 2021.
[3] J. D. Herath, “CFGExplainer: Explaining graph neural network-based malware classification,” in *Proc. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, 2022.
[4] Y. Gao, Q. Wang, and Z. Li, “Malware self-supervised graph contrastive learning with data augmentation,” in *Proc. IEEE INFOCOM Workshops*, 2023.
[5] A. Bensaoud, “A survey of malware detection using deep learning,” *Journal of Information Security and Applications*, vol. 76, Art. no. 103671, 2024.
[6] H. Haddadpajouh, A. Dehghantanha, and R. M. Parizi, “A deep learning-based approach for malware detection using behavioral graphs,” *Computers & Security*, vol. 105, Art. no. 102208, 2021.
[7] W. Li, Y. Chen, and Z. Zhang, “TS-Mal: Malware detection using temporal and structural features,” *Computers & Security*, vol. 130, Art. no. 103268, 2024.
[8] W. Wang, Y. Shang, and X. Li, “Heterogeneous graph neural networks for malicious behavior detection,” *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3211–3224, 2022.

[9] D. Arp, M. Spreitzenbarth, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. NDSS*, 2014.
[10] H. S. Anderson and P. Roth, "EMBER: An open dataset for training static PE malware machine learning models," *arXiv preprint arXiv:1804.04637*, 2018.
[11] R. Jordaney, K. Sharad, S. Dash, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification," in *Proc. USENIX Security Symposium*, 2017.
[12] M. Alasmary, A. Alhaidari, and A. Alzahrani, "Dynamic malware analysis using system-call dependency graphs," *Journal of Information Security and Applications*, vol. 54, Art. no. 102547, 2020.
[13] H. Peng, J. Li, and Y. Zhang, "Evading control flow graph–based graph neural network malware detectors," *Scientific Reports*, vol. 15, Art. no. 11234, 2025.
[14] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks," in *Proc. ACM SIGKDD*, 2019.
[15] Y. You, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," in *Proc. NeurIPS*, 2020.
[16] M. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you? Explaining the predictions of any classifier," in *Proc. ACM SIGKDD*, 2016.
[17] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware analysis techniques," *ACM Computing Surveys*, vol. 44, no. 2, pp. 1–42, 2012.
[18] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. NeurIPS*, 2017.
[19] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. ICLR*, 2017.
[20] P. Velickovic et al., "Graph attention networks," in *Proc. ICLR*, 2018.
[21] K. Xu et al., "How powerful are graph neural networks?" in *Proc. ICLR*, 2019.
[22] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *Proc. ICLR*, 2016.
[23] S. Wu et al., "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021.
[24] Z. Chen et al., "Temporal graph neural networks for dynamic malware detection," *Future Generation Computer Systems*, vol. 140, pp. 34–45, 2023.
[25] Y. Zhen, L. Wang, and H. Zhang, "A novel malware detection method based on audit logs and graph neural networks," *Computers & Security*, vol. 132, Art. no. 103312, 2025.
[26] F. Ahmed, H. Hameed, M. Z. Shafiq, and M. Farooq, "Using spatio-temporal information in API calls for malware detection," in *Proc. ACM CCS*, 2009.
[27] U. Bayer et al., "Scalable, behavior-based malware clustering," in *Proc. NDSS*, 2009.
[28] A. Rosenberg et al., "Explainable graph neural networks for cybersecurity applications," *IEEE Security & Privacy*, vol. 20, no. 3, pp. 58–67, 2022.
[29] S. Hou et al., "Explainable malware detection with graph neural networks," *Pattern Recognition*, vol. 138, Art. no. 109370, 2023.
[30] A. Singhal and X. Ou, "Security risk analysis of enterprise networks using probabilistic attack graphs," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 3, pp. 417–430, 2011.