LLMGRAPH: Label-Free Detection Against APTs in Edge Networks via LLM and GCN

Tianlong Yu, Gaoyang Liu, Chen Wang, *Senior Member, IEEE*, and Yang Yang

***Abstract*—In the growing trend of remote working, millions of edge networks (e.g., homes or branch offices) are increasingly threatened by Advanced Persistent Threats (APTs), because of the weakened segmentation between business and non-business devices in remote working environment. Despite the fact that numerous APT detection mechanisms have been proposed, all of them are struggling to handle the *complex structure*, the *massive scale* and the *diverse topology* of edge networks. *Can recent machine-learning advances tackle these APT detection pain points in edge networks?* The GNNs (Graph Neural Networks) seems to be suited to capture the *complex structure*, but its adjacency matrix fails to capture key network flow context. Additionally, GNNs require extensive manual labeling, which is not scalable. LLMs (Large Language Models) have the potential to provide automatic labeling for the GNNs, but they lack the supplementary security context needed for effective labeling. To address these gaps, we present LLMGRAPH, which incorporates extended GCNs (Graph Convolutional Networks) and domain-specific RAG (Retrieval-Augmented Generation) pipeline to achieve label-free detection against APTs in edge networks. LLMGRAPH's extended GCNs model can capture network flow context and direction. LLMGRAPH's domain-specific RAG pipeline can supplement key security contexts, including device vulnerability and network flow, for effective labeling. Additionally, LLMGRAPH provides an LLM aggregator to augment and merge the *diverse topology* of the edge networks. Compared to the state-of-the-art mechanisms, LLMGRAPH proves *effective* and *scalable*, improving the F1-score by at least 46.9%, and the training time for 1 million edge networks is within 1000 s.**

***Index Terms*—Network security, APT detection, LLM, GCN.**

Received 19 November 2024; revised 16 July 2025; accepted 28 July 2025. Date of publication 5 August 2025; date of current version 4 November 2025. This study was supported in part by the National Natural Science Foundation of China under Grant 62302177, Grant 62272183, Grant 62106069, Grant 62102136, and Grant 62002104, in part by the National Key Research and Development Program of China under Grant 2022YFA0911800, in part by the Key R&D Program of Hubei Province under Grant GJHZ202500049, in part by the Open Project Funding of the Key Laboratory of Intelligent Sensing System and Security, Ministry of Education under Grant KLISSS202401. *(Corresponding author: Yang Yang.)*

Tianlong Yu and Yang Yang are with the Key Laboratory of Intelligent Sensing System and Security (Ministry of Education), School of Artificial Intelligence, Hubei University, Wuhan 430061, China (e-mail: tommyyu21@163.com; yangyang@hubu.edu.cn).

Gaoyang Liu and Chen Wang are with the Hubei Key Laboratory of Internet of Intelligence, School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: gaoyangliu2020@gmail.com; chenwang@hust.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TDSC.2025.3596092, provided by the authors.

Digital Object Identifier 10.1109/TDSC.2025.3596092

I. INTRODUCTION

THE security landscape of the work environment had shifted greatly in the post-pandemic era. Millions of people are working more flexible through edge networks that interconnect homes, branch offices, or shared workspaces. The boundary between heavily secured enterprise networks and vulnerable edge networks is disappearing. Now, business devices (e.g., working laptops) are sharing the same network with more vulnerable non-business devices (e.g., smart TVs, cameras, coffee machines). This trend encourages Advanced Persistent Threat (APT), which is a multi-stage, stealthy, and continuous cyberattack, typically carried out by a highly organized group, that targets a specific entity with the goal of gaining and maintaining unauthorized access to systems and data over an extended period. In APTs, the attacker can first exploit non-business devices, and use the non-business devices as attack launch pads to compromise the business devices. In this process, APTs leverage multiple techniques to circumvent the current detection mechanisms. First, APTs adopt evasive techniques such as 0-day, convert channel or low frequency access [2]. Second, APTs leverages recent advances in LLM-based exploitation tools, such as hacker-GPT [17] and worm-GPT [14], to scale up their attack against the edge networks.

Current detection mechanisms includes signature-based detection detection [11], [54], anomaly-based detection [24], [35], [53] and graph-based detection [40], [44]. Signature-based detection includes legacy Firewall/IDS/IPS [11], [54]. Anomaly-based detection, such as Kitsune [53], focuses on anomalies such as abnormal flow statistic. Graph-based detection includes attack graph analysis [44], provenance-based detection [40], [43], [47], [60], and series of GNN-based detection such as E-GraphSAGE [49] and StrucTemp-GNN [26].

Current detection mechanisms are ill-suited to detect APTs among millions of edge networks due to three pain points:

- *Complex structure:* The structure of the edge networks are complex with the various devices, network flows and security events. Current signature-based detection [11], [54] and anomaly-based detection [24], [35], [53], only covers certain vantage points. Signature-based detection [11], [54] can be evaded by zero-day exploits or physical covert channels (e.g., BYOD). Anomaly-based detection [24], [35], [53] focuses on statistical anomalies and can miss out the actual APTs path.
- *Massive scale:* The scale of the devices and network flows in millions of edge networks makes it hard to perform the

1545-5971 © 2025 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.

manual labeling task. Current attack graph analysis [44], [55] requires the labeling of potential vulnerabilities on each device. The provenance-based detection [40], [43], [60] requires labeling provenance data. GNN-based detection [26], [49] requires labeling the benign and malicious devices in the training stage. None of the *manual labeling* process can scale to millions of edge networks.

- *Diverse topology:* The topology and communication pattern of the edge networks are so diverse that it is hard to apply one set of detection rules or one trained model and effectively apply it to all the edge networks. Therefore, it is hard to generate one attack graph [44], [55], one provenance graph [40], [43], [60] or one GNN model [26], [49] that is effective for all edge networks.

In this paper, we aim to address the key question: "Can recent machine-learning advances effectively tackle the APT detection pain points in edge networks?". The series of GNN models, such as Spectral GNN [28], GCN [46] GAT [65] and Graph Sage [41], are commonly used to capture graph structures, which seems to be suitable to capture the edge networks with the devices abstracted as nodes and network flows abstracted as edges. However, conventional GNN architectures (e.g., GCN, GAT, GraphSAGE) are fundamentally constrained by their reliance on adjacency matrices [23], rendering them unable to capture network flow context or directional relationships. While costly extensions (e.g., those incorporating Hodge Laplacians) can partially embed edge information, they remain unsuitable for network flow analysis, where a single edge often carries multiple flows [58], [67]. This intrinsic limitation prevents effective modeling of the complex dependencies inherent in flow-based network data. Additionally, GNNs requires a subset of well-labeled nodes to be effective (around 5% of the nodes as stated in GCN [46]), which is hard to obtain in the edge networks. The LLM do not require manual labels, but faces effectiveness and scalability issues if the LLM alone is applied for detecting all devices and network flows [32]. Another thought is to use LLMs to generate a subset of well-labeled nodes for GNNs, and use GNNs to perform scalable training and detection. However, LLMs are trained on generic data and lack the supplementary security context needed for effective labeling.

To address the above gaps, we propose LLMGRAPH, which incorporates extended GCN (Graph Convolutional Networks) and domain-specific RAG (Retrieval-Augmented Generation) pipeline to perform label-free detection against APTs in edge networks. LLMGRAPH provide several improvements over existing GCN and LLM frameworks:

- *Flow-supportive GCN:* To address the *complex structure* issue, LLMGRAPH provides extended GCN with virtual nodes to fully capture the context of multiple network flow on the same edge.
- *LLM Annotator:* To address the *massive scale* issue, LLMGRAPH provides the LLM annotator with a RAG (Retrieval Augmented Generation) pipeline to automatically annotate and select a subset of well-labeled devices and network flows.
- *LLM Aggregator:* To address the *diverse topology* issue, LLMGRAPH provide a LLM-based aggregator that can cluster and merge diverse edge networks to scale up the GCN training.

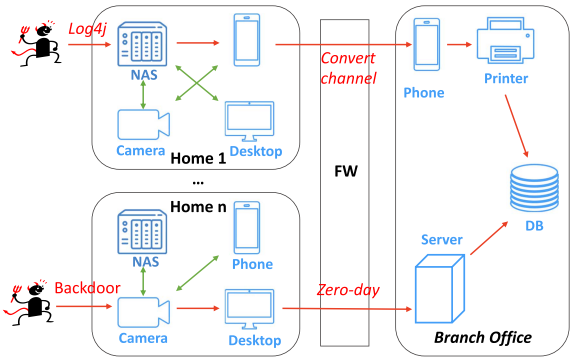

Fig. 1. APTs can breach the edge networks.

We evaluate LLMGRAPH using real edge network datasets (e.g., IoT-23 [39], IoT Sentinel [52], VirusTotal [6] and Tranco [56]), and compare it to state-of-the-art detection mechanisms, including signature-based detection [11], [54], anomaly-based detection [53] and graph-based detection (i.e., attack-graph analysis [44], [55], provenance-based detection [40], [43], [47], [60] and GNN-based detection [41]). We demonstrate that LLMGRAPH outperforms existing solutions in terms of *effectiveness* and *scalability*. LLMGRAPH is *effective*. Compared to signature-based [11], anomaly-based [53] and graph-based detection (including attack-graph analysis [44], [55], provenance-based detection [40] and GNN-based detection [41]), LLMGRAPH increases the F1-Score by at least 46.9%; LLMGRAPH is *scalable*. LLMGRAPH's extended GCN model's training time for one network with 35 000 devices and services is less than 0.5 s. The training time for 1 million edge networks (each with 240 devices and services) is under 1000 s; LLMGRAPH can handle the *diversity* of edge networks. Compared to the one-model-per-network approach (n-model) and one-model-fit-all approach (1-model), LLMGRAPH reduces the minimum FNR by 83.3% and FPR by 61.4%. Also, LLMGRAPH's components improve the effectiveness and scalability. Compared with manual labeling, the LLM-based annotator can improve accuracy by 27.09%. The extended GCN can increase the F1-Score by 9.4%. The LLM-based aggregator can increase the F1-Score by 69.9%, and also reduce the GCN training time by 78%.

II. MOTIVATION

This section begins with a detailed explanation of edge networks and APTs. We then illustrate the limitations of current defenses in securing these networks. Finally, we present the opportunity offered by GCN and LLM to address these limitations and protect the edge networks against APTs.

A. Edge Networks are Increasingly Threatened by APTs

Fig. 1 illustrates the edge networks of two employee homes and a branch office. The edge networks are connected to the business network with remote working supports. Such remote working supports includes business devices such as working

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.

desktops and laptops, VPNs and edge security routers such as Cisco Meraki [4], Amazon Eero [3] and Netgear Orbi [5]. Between the edge networks and the business networks, the perimeter defense such as the gateway firewall is still in place. Such edge networks break the heavily fortified boundaries between business devices and the non-business devices. Business devices share the same network with more vulnerable non-business devices, creating significant security risks. For instance, Fig. 1 shows two APTs attacks. In the attack on home 1, the attacker exploited a vulnerable non-business NAS device via the log4j vulnerability. Then, the attacker uses the NAS to compromise the smart phone. When the smart phone was brought into the branch office, this forms a physical convert channel that further compromises the printer and the DB in the branch office. Here, the structural context is that the NAS is normally only accessed by the camera. If the NAS could be automatically flagged, then such structural context can help to flag the phone as well. In the attack on home 2, the attacker exploited a D-Link Camera via a debug-mode backdoor. Then, the attacker uses the camera to compromise the desktop. The compromised desktop then use a zero-day exploit to compromise the server and database in the branch office. Here, the structural context is that the Camera is normally only accessed by the phone, not the Desktop.

B. Pain Points of APT Detection in Edge Networks

Existing detection approaches, including signature-based detection [11], [54], anomaly-based detection [53] and graph-based detection (i.e., attack-graph analysis [44], [55], provenance-based detection [40], [60] and GNN-based detection [41]) are not adequately equipped to protect edge networks from APTs because of the following pain points.

Complex structure: The structure of the edge networks are complex with the various devices, network flows and security events. Signature-based detection [11], [54] only covers a single topological point and can be evaded by zero-day exploits or physical covert channels. For example, the Firewall FW in Fig. 1 is bypassed by a zero-day exploit on server and a convert channel of a BYOD phone. Anomaly-based detection [24], [35], [53] focuses on abnormal statistics and can be evaded by APT attacker's stealthy strategies such as low frequency access. For example, Fig. 2 presents the result of the anomaly-based detection called Kitsune [53] on Mirai C&C and SambaCry. The Mirai C&C causes significant statistical anomalies due to multiple control message exchanges in the C&C channel. In contrast, SambaCry code injection (with low frequency SMB commands) is stealthy and does not cause significant anomalies.

Massive scale: The scale of the devices and network flows in millions of edge networks makes it hard to perform the manual labeling task. Current graph-based detection mechanisms, including attack-graph analysis [44], [55], provenance-based detection [40], [60] and GNN-based detection [41], require manual labeling to be effective. For example, in Fig. 1, attack-graph analysis [44], [55] requires the admin to label each device's vulnerabilities (e.g., Camera's Backdoor). The provenance-based detection [40], [60] requires the admin to label each device's logs to construct the provenance graph. GNN-based detection [41] requires manually labeling (e.g., compromised NAS) for model training.

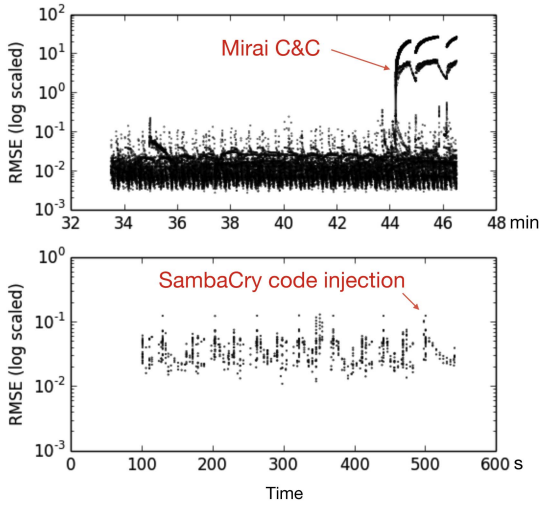

Fig. 2. The statistic anomalies caused by Mirai C&C and SambaCry code injection for Kitsune [53].

Diverse topology: The topology and communication pattern of the edge networks are so diverse that it is hard to apply one set of detection rules or one trained model and effectively apply it to all the edge networks. For example, in Fig. 1, it is hard to generate one attack graph [44], [55], one provenance graph [40], [43], [60] or one GNN model [26], [49] that is effective for thousands of different employee home networks.

C. Opportunity: GCN and LLM

Recent advances in GCN and LLM offers great opportunity to address the above issues. The intuition is to use GCN to capture the structural context, and use LLM to automatically generate a subset of well-labeled nodes for GCN training.

GCN: The spectrum of Graph Neural Networks, such as Spectral GNN [28], GCN [46] GAT [65] and Graph Sage [41], can capture structural context. Compared to GCN, other graph-based machine learning mechanisms [27], [41], [46], [65] are either less efficient or less practical. For instance, Spectral GNN [27] is not localized, meaning that the feature representation of one node can be influenced by all other nodes in the graph, which is inefficient as the training iteration evolves all all other nodes, and is impractical as the APT attacker can exploit such influence. Graph Attention Networks (GAT) [65] are not as efficient because each node needs to compute its neighbors' aggregation coefficients. Graph Sage [41] is not practical as it requires sampling and aggregation based on numeric data, potentially missing key security contexts that are non-numeric. We opted for GCN [65] because it is both efficient and practical. It has constant neighbor coefficients and does not require sampling or numeric aggregation.

LLM: Recent advances in LLMs have showcased impressive zero-shot capabilities. This means that such LLMs has the potential to annotate on devices and network flows with labels and remove the requirement for manual labeling [32]. Such LLMs can be used to automatically generate labels for

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.

TABLE I
PROMPT USED FOR LABELING LOG4J FLOWS

Input: Given the context that: "Log4j vulnerability was reported to apache by Chen Zhaojun of the Alibaba cloud security team on 24th November 2021 and published in a tweet on 9th December 2021. Apache software foundation assigned a maximum severity score of 10/10. The vulnerability allows attackers to remote code execution and the payload string looks like jndi:ldap://attacker.com/a. Lots of organization-affected services include Cloudfare, apple iCloud, Minecraft: java edition, stream, Tencent QQ, and Twitter" There are three categories: [benign, malicious, suspicious] What is the category of the following network flows? Flow 1: 172.16.141.131,,172.16.141.132, /,jndi:ldap://172.16.141.131:1389/yeicai Provide your 3 best guesses and a confidence number (0 to 100). **Ouput:** Let's analyze the network flows based on the context. Flow 1: The payload contains the string jndi:ldap://, which is indicative of a malicious intent. LDAP (Lightweight Directory Access Protocol) is commonly used for directory services, but in this context, it could be exploited for remote code execution. Category: Malicious, Confidence: 90

GCN, which requires adequate high-quality labels. To demonstrate LLM's annotation potential in security detection, we use Llama-7B model to label the network flow of Log4j attack [13], as shown in Table I. In this experiment, we ask the LLM model to annotate the log4j network flow into three categories - benign, malicious and suspicious. Note that vulnerability information of Log4j is also given for effective annotation. In the output part, we can see that the LLM model accurately label the log4j network flow as malicious. The LLM model even provide an reasonable analysis for the log4j flow's payload and pointed out it could be exploited for remote code execution. This demonstrates LLM's potential for effective device and network flow annotation.

Potential LLM-GCN-based APT detection: A key observation is that - APTs are inherently multi-stage attacks where concealing all stages via zero-day exploits or covert channels is extremely difficult (especially considering that a zero-day exploit on a specific device would take months to uncover). The LLM can be used to annotate some APT stages and devices with known exploits or existing security anomalies, which is label-free - no manual label is required. GCNs then correlate these LLM-generated annotations across different attack stages and propagate the identified suspicion along potential full APT paths. For instance, in Fig. 1, the first APT (upper red path) has the LLM annotate the known Log4j exploit and the anomalous NAS access to a BYOD phone; the GCN correlates these and propagates suspicion to the covert channel of the BYOD phone. Similarly, for the second APT (lower red path), the LLM flags the anomalous camera access to the desktop, and the GCN propagates this suspicion to the desktop's later zero-day exploit against the server.

III. OVERVIEW

In this section, we present an overview of LLMGRAPH, which combines LLM and GCN to identify *compromised devices* and *malicious network flows* of APTs in edge networks.

Threat model: In this part, we define the threat model:

1) The attacker can launch APT attacks against edge networks with evasive techniques including 0-day exploits, physical covert channels and low frequency accesses.

2) The defense system, including the host-based detection systems and edge security gateways, is able to monitor the host logs and network flows.

3) For encrypted network traffic, the defense system cannot inspect the content, but can analyze the metadata (e.g., source, destination, traffic volume, etc.).

4) The attacker cannot compromise the detection itself.

Our intuition: In Section II, we can see that GCN can be used to capture the structural context in edge networks. However, GCN requires a subset of nodes with high-quality labels to be effective. Meanwhile, LLM has the potential to automatically label the devices and network flows in edge networks. But LLM cannot be trained or directly applied to utilize the graph structural context and perform detection on edge networks effectively and efficiently. Therefore, our idea is to integrate GCN and LLM, leveraging the GCN to capture the graph structural context, and leveraging the LLM to automatically label the devices and network flows for GCN training.

Basic LLMGRAPH *workflow:* Fig. 3 shows the basic LLMGRAPH workflow. In LLMGRAPH, the edge networks are represented in the form of Text-Attributed Graph (TAG) as $G_T = (V, A, T, X, Y)$. $V = \{v_1, v_2, \ldots, v_n\}$ is the set of n nodes paired with text attributes $T = \{t_1, t_2, \ldots, t_n\}$. The adjacency matrix $A \in \{0, 1\}^{n \times n}$ represents graph connectivity. The feature matrix $X = \{x_1, x_2, \ldots, x_n\}$ describes the encoded feature for the nodes V and corresponding text attributes T. The label matrix $Y = \{y_1, y_2, \ldots, y_n\}$ represents the labels (i.e., benign, malicious, suspicious) for the nodes V.

As the first step of the workflow, LLMGRAPH's LLM Annotator will perform labeling on nodes V (i.e., devices and network flows). In this process, the LLM Annotator will convert the text attributes T into feature matrix X, and automatically generate the label matrix Y_{train}. After that, LLMGRAPH provides a LLM-based Aggregator to process different edge networks. Finally, the edge networks and their devices and network flows will be processed by the GCN module. The GCN module will perform a graph spectral convolution $Z = f(X, A)$ to encode the graph input into signals in *Fourier space* [28], [33], [46]. Then, the input signal will transverse through multiple hidden layers with activation functions (e.g., ReLU) and a softmax function. In the training stage, GCN will generate the parameters W of the above GCN structure based on the label matrix Y_{train}. In the prediction stage, GCN will use the trained parameters W to make the prediction Y_{pred} on nodes V_{pred} with feature X_{pred} expected to be predicted (i.e., benign, malicious, suspicious).

However, there are three key challenges to make the above LLMGRAPH workflow *practical* for edge networks:

Challenge 1: Generic LLM lacks labeling effectiveness. GCN needs a subset of well-labeled nodes to be effective [46]. However, generic LLM lacks the labeling effectiveness to provide such subset of well-labeled nodes. First, current LLM models, such as ChatGPT [8], MS copilot [9], Llama [64], Vicuna [12] or Red Pajama [10], are trained on generic dataset and lacks

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.

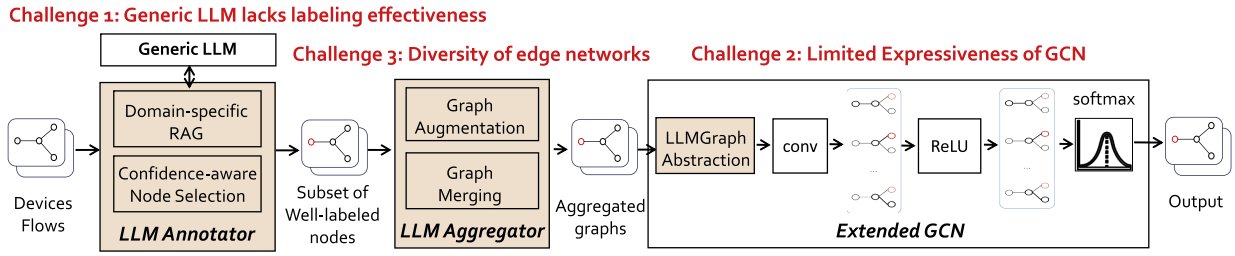

Fig. 3. LLMGRAPH workflow.

TABLE II
GENERIC LLM (GPT-4O 2025-06-25) LACKS CONTEXT AND CANNOT LABEL CISCO-IOS-BACKDOOR FLOW AS MALICIOUS

Input: There are three categories:
[benign, malicious, suspicious]
What is the category of the following network flows?
Flow 1: 192.168.1.3,,192.168.1.4, /,/level/16/exec/
Ouput: This flow should be categorized as **suspicious**, as the endpoint /level/16/exec/ suggests a potentially risky action (executing commands at a high access level), **but additional context or evidence of exploitation is needed to confirm it's malicious**. It could be part of a legitimate system, but it's definitely worth investigating further.

the domain-specific security data (e.g., CVE information) to provide accurate labels. For example, in Table II, the generic LLM (GPT-4o 2025-06-25-version) lacks context data (e.g., vulnerability description) and cannot label Cisco-IOS-backdoor flow as malicious. Another issue is that the generic LLM is not capable of selecting the subset of well-labeled nodes for GCN.

Challenge 2: Limited expressiveness of GCN. In the context of APTs in edge networks, existing learning models have limitations in representing key domain-specific concepts such as network flows and protocol fields. For instance, graphic learning models [28], [33], [46] rely on an adjacency matrix that only records connectivity and does not capture the detailed context of network flows (e.g., firmware update `URI`).

Challenge 3: Insufficient data and various networks. Edge networks exhibit a high degree of diversity, with each network having a unique set of devices and communication patterns. It is challenging to apply a one-size-fits-all model that adequately covers all edge networks. Alternatively, if each edge network trains its own model, the data available within each edge network is often insufficient for model training. For instance, there are new or infrequently used devices that have limited communication records.

In the following part, we will discuss LLMGRAPH's design and how LLMGRAPH addresses the above challenges. In Section IV, we will demonstrate how LLMGRAPH's annotator employs a domain-specific RAG pipeline and a confidence-aware node selection mechanism to address the *lack of labeling effectiveness* issue. In Section V, we will provide an extended GCN pipeline with virtualized nodes to address basic GCN's *limited expressiveness* issue. In Section VI, we will provide a LLM-based aggregator with graph augmentation and clustering to scale up the GCN detection on diverse edge networks with

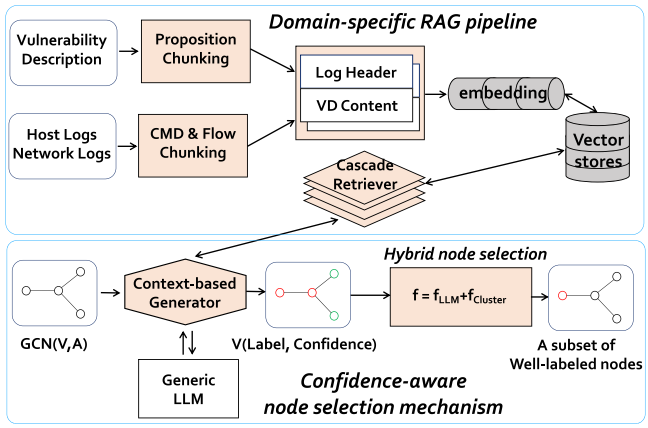

Fig. 4. The workflow of the LLM annotator.

massive scale. In Section VIII, we will provide the evaluation of LLMGRAPH in terms of effectiveness and scalability. We will walk through the related work in Section IX and discuss other issues in Section X.

IV. LLM ANNOTATOR

GCN requires a small subset of well-labeled nodes (i.e., around 5% of the nodes) to be effective [32], [46]. To fulfill this requirement, LLMGRAPH provide a LLM-based Annotator to first label all the nodes into three categories - benign, malicious and suspicious, and then select a subset of well-labeled nodes for GCN training. However, this is not a easy task and there are two key issues: 1) *Lack of domain-specific data* - current LLM models [8], [9], [10], [12], [64] are trained on generic dataset and lack the domain-specific data needed to effectively label the devices and network flows, as shown in Table II; 2) *Require proper node selection* - the effectiveness of GCNs relies on a small subset of well-labeled nodes [46]. While we can use LLM to label all nodes, selecting an appropriate portion of well-labeled nodes remains critical. To address these issues, the LLM Annotator's workflow is as shown in Fig. 4. There are two parts - the domain-specific RAG pipeline and the confidence-aware node selection.

A. Domain-Specific RAG Pipeline

To address *lack of domain-specific data* issue, the LLM Annotator provides a RAG (Retrieval-Augmented Generation)

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.

pipeline to indexing and retrieve relevant security and network data (e.g., CVE), as shown in Fig. 4. RAG is an emerging technique that enhances the capabilities of LLM by plugging in other data sources, in addition to the knowledge that LLM itself has learned. RAG's workflow includes two key steps - indexing and retrieval. The indexing step is crucial for obtaining accurate and context-aware answers. It begins with extracting and cleaning data from various file formats. The cleaned text is then split into smaller chunks to avoid context limitations within LLMs. Each chunk is transformed into a numeric vector via embedding, and an index is built to store these chunks and their corresponding embeddings. In the retrieval step, the query is converted into a vector representation using the same embedding. Similarity scores between the query vector and the vectorized chunks are calculated, and the system retrieves the top K chunks most similar to the query.

RAG systems face two key challenges in these steps - 1) indexing effectiveness; 2) retrieval efficiency. Generic indexing divide text into meaningful segments via fixed size chunking. However, in LLMGRAPH's case, the effectiveness of such generic indexing is limited. This is because the vulnerability description and the host & network logs are not suitable for fixed size chunking. Then, in the retrieval process, query is converted into a vector representation and similarity scores between the query vector and vectorized chunks are calculated. However, given the massive scale of the vulnerability descriptions and the host & network logs, the size of the vectorized chunks are huge and the retrieval efficiency is greatly limited. To address the indexing effectiveness issue, LLMGRAPH's LLM annotator provides the hybrid chunking. To address the retrieval efficiency issue, LLMGRAPH's LLM annotator provides the cascade retriever.

Hybrid chunking: To improve the indexing effectiveness, LLMGRAPH's LLM annotator adopts the hybrid chunking to customize and synthesize the chunking of different inputs including the vulnerability description and the host & network logs. More specifically, for vulnerability description, the LLM annotator adopts the proposition chunking [31]. Propositions are atomic expressions within text, each encapsulating a distinct factoid and presented in a concise, self-contained natural language format. The three principles below define propositions as atomic expressions of meanings in text. Each proposition should represent a distinct piece of meaning in the text, collectively embodying the semantics of the entire text. A proposition must be minimal and cannot be further divided into separate propositions. A proposition should contextualize itself and be self-contained, encompassing all the necessary context from the text to interpret its meaning. Then, for host & network logs, LLMGRAPH's LLM annotator adopts semantic chunking based on command and network flows. After that, LLMGRAPH's LLM annotator will synthesize the vulnerability description chunks and the host & network logs chunks together. In the synthesize stage, the host & network logs chunks will be used as the header and the vulnerability description chunks will be used as the content.

Cascade retriever: To enhance retrieval efficiency, the LLM annotator employs a cascade retriever. The core concept involves leveraging host and network log headers as a fast matching structure. The historical host and network log headers are shorter than the vulnerability description, and is more likely to match the current host and network log part in the current query. When a new query Q with host and network logs arrives, the cascade retriever will calculate the min cosine distance d_i and max cosine distance d_j between the query Q and the chunk headers C_H. If the ratio d_i/d_j is greater than threshold T (0.01 unless other specified), then all the host and network log header matches yield low scores, and the query Q will continue to match the content of the chunks C_N, which are the vulnerability descriptions. After that, the prompt P will be constructed by combining the query Q with the best matched chunk with index i. Additionally, the chunks are reranked using a Least Recently Used (LRU) mechanism to improve future matches.

Algorithm 1: Confidence-Aware Node Selection.

Require: Edge networks $G(V, E)$, Devices V with labels L_V, Flows E with labels L_E
Ensure: Devices V^s with labels L_V^s, Flows E^s with labels L_E^s
1: $n_{cluster} \leftarrow 3$
2: $C \leftarrow KMeans(n_{cluster}, V, L_V)$
3: **for all** $v, v \in V$ **do**
4: $f_{llm}(v) \leftarrow LLM(v)$
5: $f_{cluster}(v) \leftarrow d(v, c_v)$
6: $f(v) = c_0 f_{llm}(v) + c_1 f_{cluster}(v)$
7: **end for**
8: $(V^s, L_V^s) \leftarrow Rank(V, f(V))$

B. Confidence-Aware Node Selection

To address the *require proper node selection* issue, the LLM Annotator provides a confidence-aware annotation to select a subset of well-labeled nodes for GCN training. The confidence-aware node selection mechanism combines two factors: 1) the confidence score from LLM, and 2) the distance between the labeled node and the cluster center.

To calculate the confidence score from LLM, LLMGRAPH employs the template shown in Table I in Section II. In the labeling process, each device or network flow is treated as a node in GCN. The nodes are classified into three categories: benign, malicious, and suspicious. Each node receives a confidence number from 0 to 100. For example, the log4j flow with "jndi" information is labeled as malicious with a confidence score of 90, as shown in Table I.

To calculate the distance between the labeled node and the cluster center, LLMGRAPH needs to identify the cluster center. However, calculating the distance between labeled nodes and the cluster center is challenging in unweighted and non-localized GCN graphs. To address this, we use the clustering coefficient of the unweighted graph to identify the center. For an unweighted graph, the clustering coefficient of a node u is defined as:

$$f_{cluster}(u) = \frac{2T(u)}{\deg(u)(\deg(u) - 1)} \tag{1}$$

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.

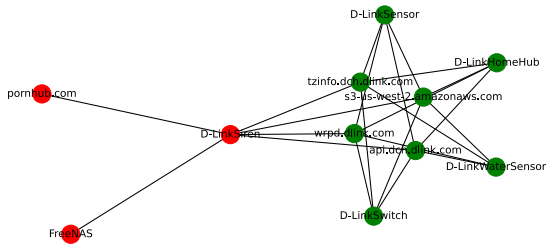

Fig. 5. LLMGRAPH's abstraction for a home network.

where $T(u)$ is the number of triangles through node u, and deg(u) is the degree (number of neighbors) of node u.

Finally, we combine these two factors together to realize the confidence-aware node selection:

$$f = c_0 f_{llm} + c_1 f_{cluster} \tag{2}$$

where f_{llm} is the LLM confidence score and $f_{cluster}$ is the cluster confidence score, c_0 and c_1 are balancing coefficients.[1]

V. EXTENDED GCN

Creating an *expressive* learning model for network security contexts presents a significant challenge. In this section, we will walk through the extended GCN, and demonstrate how we can tackle this challenge.

A. Communication Graph Abstraction

Benign and malicious communications within edge networks occur across various devices and services, and under diverse contexts. Therefore, it's crucial to have a succinct abstraction that can extract the key information from these communications. In LLMGRAPH, we introduce a *communication graph abstraction* designed to capture the essential information of benign and malicious communications. In LLMGRAPH's abstraction $G(V, E)$, each node $v \in V$ is either an internal device d or an external service s. Internal devices are devices within the gateway of the edge networks, and external services are services on the internet. LLMGRAPH distinguishes internal devices (e.g., camera) and external services (e.g., website or IP address), because internal devices are part of the edge networks, are more critical and more manageable for the security admins. Each internal device d is composed of its MAC address d_{mac}, IP address d_{ip}, device type d_{type}, device vendor d_{vendor} and device model d_{model}.[2] Each external service s is composed of its IP address s_{ip}, domain name s_{domain}, url s_{url}. Each edge $e \in E$ is a group of network flows between two nodes. Each edge e includes the source-destination pair (e_{src}, e_{dst}). Each edge e also includes a list of edge contexts (e.g., protocol type) or protocol-specific information (e.g., URI field).

Fig. 5 shows an example of LLMGRAPH's abstraction for a smart home network [52]. Each node is a device (Switch/Sensor/Water-sensor/Hub/Siren) or a external service (e.g., *api.dlink.com*). Each edge is a group of communication flows (the edge context is omitted for simplicity). For benign communication, LLMGRAPH's abstraction clearly shows that there is no direct communication between the devices. The devices only communicate with the external services, which matches the typical edge-to-cloud communication pattern for a wide range of devices.

The LLMGRAPH's abstraction can detect malicious communication patterns. As shown in Fig. 5, the malicious communications implemented a chain of intrusion and ransomware attack against the home network. In this attack, the attacker first uses an Exploit-Toolkit (EK) to exploit the D-Link Siren. Then, the attacker uses the D-Link Siren as a stepping stone and send SMB commands to a Synology NAS to exploit the Samba protocol vulnerability and performs a SambaCry attack [1]. The LLMGRAPH's abstraction clearly shows the malicious pattern that: 1) the D-Link Siren is communicating with an external service from adult website *pornhub.com*; 2) the D-Link Siren is communicating with the Synology NAS. If we can learn the communication patterns, then it can be used to detect the chain of APT attacks.

B. Construct LLMGRAPH Abstraction

Algorithm 1 shows the process of building LLMGRAPH's abstraction $G(V, E)$. The inputs are the host information I_{host} and the network traffic T_{all}. A device list D can be derived from the host information I_{host} gathered at the edge router (e.g., dhcp information). For each device d, the 5-tuples $\{d_{mac}, d_{ip}, d_{type}, d_{vendor}, d_{model}\}$ can be derived from host information I_{host} and the network traffic T_{all}. A service list S can be derived from the DNS part of the network traffic $T_{dns} \in T_{all}$, as connecting to a service involves the DNS resolution of the service's domain. For each service s, the 3-tuples $\{s_{ip}, s_{domain}, s_{url}\}$ can be derived from the network traffic T_{all}, and s_{url} can be used as the service id since the IP address may change over time. The vertices list V is an union of internal devices D and external services S, noted as $D \cup S$. To build the edge list E, LLMGRAPH identifies all the flows L in the network traffic T_{all}. For each flow $l \in L$, we can extract the source l_{src}, destination l_{dst}, protocol type l_{proto} and context $l_{context}$ (such as http uri or http XSS option). The flows can be grouped by their source-destination pair (l_{src}, l_{dst}) and generate an edge $e \in E$. Given V and E, the LLMGRAPH abstraction $G(V, E)$ is constructed.

C. GCN Workflow

Based on GCN's data schema, LLMGRAPH's communication graph inputs can be formalized as feature matrix X, adjacency matrix A and label matrix Y. The feature matrix $X \in \mathbb{R}^{N \times C}$ is composed of C feature values for N nodes. The adjacency matrix $A \in \mathbb{R}^{N \times N}$ contains all the edges of the graph. The label matrix Y includes the class labels of each node, and it allows empty value for nodes without label. To encode the graphic input of both feature matrix X and adjacency matrix A, GCN leverages a technique called *spectral graph convolutions*. The basic idea is to calculate the spectral convolutions of graphs defined as the

[1] Set to 0.5 unless specified otherwise.

[2] The internal device in this paper is composed of these five-tuples. The device definition can be extended to include other device attributes, e.g., device hostname.

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.

multiplication of the feature matrix X with a filter encoding the adjacency matrix A in the Fourier domain [28], [33], [46]. The convolved signal matrix is $Z = f(X, A)$ where f is the filter function with parameter Θ in Fourier domain. More specifically, formula $Z = f(X, A)$ can be expanded as:

$$Z = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta$$

Here, $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph G with added self-connections. I_N is the identity matrix, and $\tilde{D}_{ii} = \Sigma_j \tilde{A}_{ij}$. $\Theta \in \mathbb{R}^{C\times F}$ is a matrix of filter parameters, where C is the dimension of feature vector for every node, and F is the number of fourier filters.

After *spectral graph convolutions*, the input feature matrix X and adjacency matrix A are mapped to a hidden layer. Then, GCN [46] propagates the input (or called activation) through multiple hidden layers, including typical *ReLU* function and *softmax* function to generate the final output. In the training stage, the final output will be compared with the label matrix Y to train the parameters in the convolution Z, *ReLU* function, *softmax* function and each hidden layer. Then, in the real-time detection stage, the finally output will be used to predict the class of the node (e.g., whether a device is benign or malicious). For example, for a two-layer GCN, LLMGRAPH will first calculate $A = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ in a pre-processing step. Then, the forward model is as follows:

$$Z = f(X, A) = softmax\left(\tilde{A}\, ReLU\left(\tilde{A}XW^{(0)}\right)\right)W^{(1)}$$

Here $W^{(0)}$ is an input-to-hidden weight matrix and $W^{(1)}$ is a hidden-to-output weight matrix. Then, when training the weight matrix $W^{(0)}$ and $W^{(1)}$, the loss function is defined as the cross-entropy error as follows:

$$\ell = -\sum_{l\in y_L}\sum_{f=1}^{F} Y_{lf} ln Z_{lf}$$

where y_L is the set of node indices that have labels. With the above loss function, the weight matrix $W^{(0)}$ and $W^{(1)}$ are then trained via gradient decent.

D. Extending GCN Expressiveness

We've outlined the basic workflow of GCN above. However, the standard GCN has three significant limitations that hinder its detection capabilities. First, GCN does not support *multiple edges* between two nodes. The adjacency matrix of GCN only records whether two nodes are connected or not, and as a result, it cannot express multiple network flows between two nodes or identify specific malicious flows. Second, GCN does not support *edge context*. Edge context is vital as it captures the content of communications [29], [36], [42], [45], [63]. For instance, an HTTP connection with an 'upgrade' string in the edge context could indicate a firmware update operation, which often signifies code injection attacks. Lastly, GCN does not support *edge direction*. The direction of an edge is crucial as it captures the direction of communication, especially in the case of multi-stage intrusions.

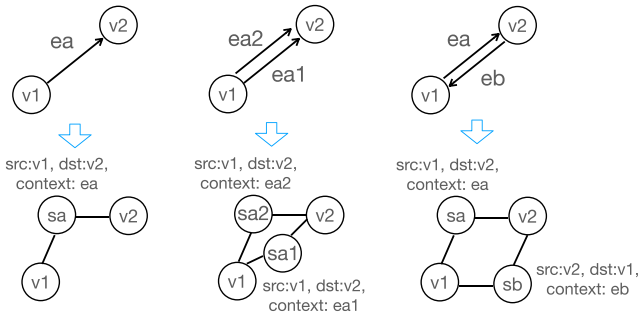

Fig. 6. Extending GCN expressiveness.

To address these three issues, LLMGRAPH extends GCN's expressiveness to be able to capture multiple edges between two nodes, edge context and edge direction. The key idea is to generate *virtual nodes* to represent the edges, and connect them with the edge's original source/destination nodes, as shown in Fig. 6. To support multiple edges between two nodes, LLMGRAPH can create a virtual node for each edge. To support edge context, LLMGRAPH assign the edge context as the virtual nodes's attributes. To support edge direction, LLMGRAPH can convert the direction as part of the virtual nodes's attributes. For example, in Fig. 6, to represent edge ea, virtual node s_a is created. To represent the edge context and direction, s_a is connected with $v1$ and $v2$, and s_a's node attribute will include $context : ea$, $src : v1$ and $dst : v2$. After the extension, LLMGRAPH can capture the network flow characteristics in the training stage, and can identify if one or a group of network flow is benign or malicious in the detection stage. To support multiple edges, for each edge e, we create a virtual node v_s and connect it with the edge's original source e_{src} and destination e_{dst}. To support edge context and direction, the shadow node v_s includes $e_{context}$, e_{src} and e_{dst} to its attributes. We add e_{s1} and e_{s2} to the edge list E, and remove the original edge e.

VI. LLM-BASED AGGREGATOR

The extended GCN discussed in Section V is capable of detecting devices and network flows. However, practically applying this extended GCN to *diverse* edge networks remains challenging. The *diversity* across these networks gives rise to two main issues: 1) *insufficient data* - each network lacks sufficient data to train its own GCN model; 2) *variance between networks* - the differences between edge networks make it difficult to create an effective one-size-fits-all model. In this section, we address the *diversity* challenge by introducing LLMGRAPH's LLM-based aggregator.

A. Handling Insufficient Data

Machine learning models typically require a substantial amount of data for training. However, edge networks face limitations in data availability. For example, suppose the GCN model for a home network is trained using data from a SambaCry attack against a Synology NAS device. A new device such as FreeNAS (different from Synology NAS) lacks sufficient historical data to flag similar SambaCry attack. To address the *insufficient*

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2: LLMGRAPH's Augmentation Mechanism.

Require: LLMGRAPH abstraction $G(V, E)$
Ensure: LLMGRAPH abstraction $G_{Aug}(V_{Aug}, E_{Aug})$
def $sweep(V, E, V_{sub}, layer)$
  for all $v, v \in V_{sub}$ **do**
    $V_{sub} \leftarrow V_{sub} + neighbors(V, E, v)$
  end for
  $layer \leftarrow layer + 1$
  if $layer > MAXHOPS$ **then**
    return V_{sub}
  end if
  return $sweep(V, E, V_{sub}, layer)$

$V^{fuzz} \leftarrow V$
$E^{fuzz} \leftarrow E$
Device attribute $tr^V \leftarrow Attribute(V)$
for all $v_a, v_a \in V$ **do**
  $\{v_a^{fuzz}\} \leftarrow LLMFuzz(tr^V, v_a)$
  for all $e, e \in E$ **do**
    $v_b \leftarrow neighbor(e, v_a)$
    $e^{fuzz} \leftarrow edge(v_a^{fuzz}, v_b, context_e)$
  end for
  $V^{fuzz} \leftarrow V + \{v_a^{fuzz}\}$
  $E^{fuzz} \leftarrow E + \{e^{fuzz}\}$
end for
$V_{sub} \leftarrow sweep(V, E, V_{malicious}, layer)$
$E_{sub} \leftarrow \{e, e_a \in V_{sub}, e_b \in V_{sub}\}$
$V_{Aug} \leftarrow V_{fuzz} + V_{sub}$
$E_{Aug} \leftarrow E_{fuzz} + E_{sub}$
return $G_{Aug}(V_{Aug}, E_{Aug})$

data issue, LLMGRAPH's LLM-based aggregator introduces two mechanisms. The first mechanism is the *domain-specific data augmentation* to manifest more edge network data for creating additional training examples. The second mechanism is *merging edge networks* to combine their GCN models and facilitate knowledge sharing.

LLM-based domain-specific data augmentation: Data augmentation is a common technique used to address the insufficient data issue. However, traditional augmentation methods are primarily designed for image or natural language data. Two common techniques are *rotation* and *cropping*. In image data, rotation involves varying the angle of the original image to capture different perspectives, while cropping focuses on extracting local patterns from sub-images. Obviously, *rotation* and *cropping* are specific to image data, and cannot be directly applied to devices and network flows.

To bridge this gap, LLMGRAPH introduces two LLM-based domain-specific data augmentation mechanisms, as shown in Algorithm 2. The first mechanism is device-type-based fuzzing, which generates various combinations of devices. This concept is analogous to the rotation technique. While image rotation creates images at different angles, LLMGRAPH's device-type-based fuzzing technique generates communication graphs with varying combinations of devices, vendors and models. In the fuzzing process, LLMGRAPH collects device information from multiple edge networks, focusing on device type, vendor, and model. It organizes this data into a tree structure. For each device in the communication graph $G(V, E)$, LLMGRAPH identifies the corresponding branch. It then fixes the device's type and performs the LLM-based fuzzing on the vendors and models. For example, in Table III, given the device type "NAS" and the vendor "Synology", the LLM-based fuzzing is able to generate model data such as "DS224" that manifest the training examples that potentially covers new devices. The underlying intuition of the device-type-based fuzzing is that devices of the same type often exhibit similar functionality and benign communication patterns. In contrast, the APT attack would deviate from these benign patterns for exploitation purpose.

TABLE III
LLM-BASED FUZZING

Input: Given device type "NAS" with vendor "Synology", list out all possible combinations of device, vendor, model.
Ouput: Here are combinations based on Synology NAS.
Device: NAS, Vendor: Synology, Model: DS224+
Device: NAS, Vendor: Synology, Model: DS923+
Device: NAS, Vendor: Synology, Model: DS3622xs+
Device: NAS, Vendor: Synology, Model: DS223j
Device: NAS, Vendor: Synology, Model: DS420j
Device: NAS, Vendor: Synology, Model: DS920+
Device: NAS, Vendor: Synology, Model: DS1621+
Device: NAS, Vendor: Synology, Model: DS1821+
Device: NAS, Vendor: Synology, Model: DS1621xs+

Algorithm 3: LLMGRAPH Network Clustering.

Require: LLMGRAPH abstractions for multiple homes $\{G_m\}$
Ensure: A number of network clusters $\{C_n\}$
def $LLMDistance(f_1, f_2)$
  $f_1 \leftarrow LLMTemplate(f_1)$
  $f_2 \leftarrow LLMTemplate(f_2)$
  $d \leftarrow LLMDistanceScore(f_1, f_2)$
  return d
$\{P_m\} \leftarrow NodeAttribute(\{G_m\})$
Cluster number $m \leftarrow Elbow(\{P_m\}, LLMDistance)$
function $df \leftarrow LLMDistance$
$\{C_n\} \leftarrow$
$HierarchicalClustering(\{G_m\}, \{P_m\}, m, df)$

The second mechanism is a *graph segmentation* mechanism, which is similar to the cropping mechanism. LLMGRAPH's graph segmentation generates sub-graphs that focus "attention" on malicious nodes and their neighbors. Essentially, LLMGRAPH's graph segmentation will generate a sub-graph that includes the malicious nodes and their r-hop neighbors.[3] This helps the GCN capture the boundary pattern between malicious and benign nodes. Algorithm 2 outlines the workflow of the data augmentation mechanism. The fuzzing process utilizes the device-attribute-tree structure, and the graph segmentation mechanism employs an iterative sweeping mechanism to segment the malicious nodes.

[3]In this paper, r are set to 3 unless specified otherwise

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
LLM-BASED DISTANCE SCORE

Input: Calculate a distance score in (0, 1) between home A and home B based on the devices and flows.
Home A - Devices: [NAS, Camera, Laptop, Phone], Flows: [NAS-Camera, Camera-Phone], Attributes: [D-Link Camera]
Home B - Devices: [NAS, Camera, Desktop, Phone], Flows: [NAS-Desktop], Attributes: [D-Link Camera]
Ouput: Let's calculate the distance score.
$DeviceOverlapScore = \frac{CommonDevices}{TotalDevices} = 0.75$
$FlowOverlapScore = \frac{CommonFlows}{TotalFlows} = 0.33$
$DeviceAttributeScore = 1$
Combine the scores:
$OverallDistanceScore = 0.5*0.75+0.3*0.33+0.2*1 = 0.67$

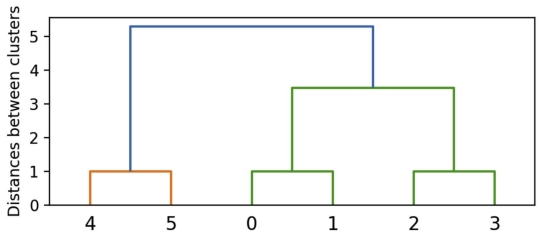

Fig. 7. Dendrogram of the LLM-based clustering distance for six example home networks.

Merging edge networks: Merging edge networks is another strategy to tackle the *insufficient data* problem. It is straight forward to merge LLMGRAPH's communication graph abstractions defined in Section V to enable knowledge sharing. However, due to the diversity of edge networks, creating a one-size-fits-all model is challenging. We will delve into this issue further in the following part.

B. Handling Variance Between Networks

LLMGRAPH clustering various edge networks into distinct groups and developing a dedicated LLMGRAPH model for each cluster, as shown in Algorithm 3. The underlying rationale is that networks with similar devices and services are likely to exhibit comparable communication patterns. To achieve this, we vectorize the node and edge attributes of networks and perform clustering based on these attributes. The choice of clustering method depends on whether the number of clusters is predetermined. K-means is typically used when the number of clusters is fixed, while hierarchical clustering is preferred otherwise. In this case, we employ hierarchical clustering (Line 13) due to the unfixed number of clusters and the structural data of the node/edge attributes. A crucial aspect of this process is defining the distance between different clusters based on the node/edge attributes (Line 1). Since node and edge attributes are various (e.g., numeric, enumerated or string values), LLMGRAPH adopts a LLM-based similarity score as the distance metrics, as shown in Table IV.

Fig. 7 illustrates the clustering result of six example home networks in a dendrogram format. The x-axis represents the network ID, while the y-axis indicates the distance between clusters based on the LLM-based clustering distance. As seen in Fig. 7, networks can be grouped into two or three clusters. To determine the optimal number of clusters, we utilize the Elbow method [62]. This method calculates variance as a function of cluster numbers and selects the elbow point on the curve as the optimal cluster number. Fig. 8 displays the elbow curve for six example home networks. The variance decreases sharply before the cluster number increases to three, indicating that three is the elbow point and thus, the optimal number of clusters.

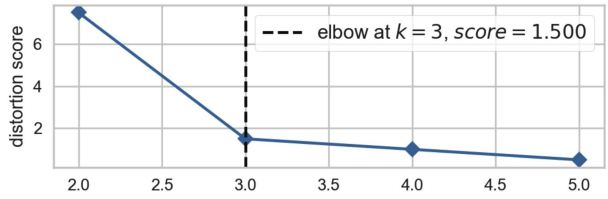

Fig. 8. Distortion score elbow of LLM-based clustering.

VII. IMPLEMENTATION

We implemented LLMGRAPH with 5 K LoC on a desktop server equipped with an `NVIDIA RTX 4090 GPU` (24 GB VRAM), `Intel Platinum 8352 CPU` (36 cores), 32 GB RAM, and 16 TB HDD.

LLM: The LLM annotator and the LLM aggregator are implemented using `Llama-3-8B` [19] and deployed using `Ollama` [20].

RAG: The RAG is implemented using `LangChain` [18] (chunk size = 500 and chunk overlap = 10). The vector store is using `FAISS` [15]. The embedding model is using `Llama-3-8B`.

GCN: The extended GCN module is implemented based on the `GCN` [46].

VIII. EVALUATION

In this section, we evaluate LLMGRAPH and show that:

1) LLMGRAPH is *effective*. Compared to signature-based [11], anomaly-based [53] and graph-based detection (including attack-graph analysis [44], [55], provenance-based detection [40] and GNN-based detection [41]), LLMGRAPH increases the F1-Score by at least 46.9%;

2) LLMGRAPH is *scalable*. LLMGRAPH's extended GCN model's training time for one network with 35 000 devices and services is less than 0.5 s. The training time for 1 million edge networks is under 1000 s;

3) LLMGRAPH can handle *diverse* edge networks. Compared to one-model-per-network and one-model-fit-all approach, LLMGRAPH reduces FNR by 83.3% and FPR by 61.4%.

4) LLMGRAPH's components enhance effectiveness and scalability. The LLM-based annotator improves labeling accuracy by 27.09%. The extended GCN increases the F1-Score by 9.4%. The LLM-based aggregator increases the F1-Score by 69.9% and reduces GCN training time by 78%.

A. Experiment Setup

Testbed: Evaluating edge networks poses challenges due to their massive scale and practical security, privacy, and usability

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.

concerns. To address this, we set up a testbed that emulates different edge networks by combining physical and virtual networks. The testbed includes multiple edge networks and a Security Operations Center (SoC), as depicted in our supplementary materials. In each edge network, we deploy an edge security gateway (a Raspberry Pi 4B running Snort 2.9.8.0). The attack launchpad, physical devices, and virtualized devices are wirelessly connected to the edge gateway. We create virtualized devices and topology using Cisco's GNS3 [7] virtualization testbed. Remote SoCs control the edge gateways, performing analysis and detection (including LLMGRAPH model training). By combining physical and virtual topologies, our testbed accommodates a wide range of edge network settings.

Dataset: We evaluate LLMGRAPH by deploying four real-world datasets - IoT-23 [39], IoT Sentinel [52], VirusTotal [6], and Tranco [56], on our hybrid testbed. The IoT-23 [39] and IoT Sentinel [52] are network trace datasets, and are deployed by replaying the pcap files among the testbed devices. The VirusTotal [6] and Tranco [56] are domain and url datasets, and are deployed by running domain/url accessing scripts on the testbed devices. More specifically, the IoT-23 dataset [39] comprises network packet captures from IoT devices, featuring 20 distinct malware families and three sets of benign IoT device traffic. Originally published in January 2020, its captures span from 2018 to 2019. The IoT Sentinel [52] records benign network packets from 31 types of IoT devices in home networks. VirusTotal [6] dataset is obtained by crawling the VT Intelligence API [21] on Sep-2024. This dataset includes domains and URLs from 30 malware families targeting home/business IoT devices. Tranco [56] is a ranked registry of the top one million popular websites. To ensure legitimacy, we filter Tranco domains using the VT Intelligence API [21] and Google Safe Browsing API [16], excluding malicious or expired entries. Since the domains and urls can expire, all Tranco, VT Intelligence, and Safe Browsing data reflects the Sep-2024 snapshot. The evaluation is also performed on Sep-2024 and the datasets are collected/updated right before the evaluation to ensure it is accurate.

Other approaches: We compare LLMGRAPH with signature-based detection [11], anomaly-based detection [53], and graph-based detection [40], [44], [49], [55]. The graph-based detection includes the attack graph analysis [44], [55], the provenance-based detection [40], and the GNN-based detection [49]. For signature-based detection, we use Snort version 2.9.8.0 [11]. For anomaly-based detection, we setup Kitsune [53]. For attack graph analysis, we identify the known vulnerabilities of every device and setup a symbolic execution to enumerate all the possible paths that violates the security condition based on the attack graph analysis [44], [55]. For provenance-based detection, we compare with R-CAID [40]. For GNN-based detection, we setup E-GraphSAGE [49].

B. LLMGRAPH's Effectiveness

We compare the F1-Score, the FPR (False Positive Rate) and the FNR (False Negative Rate) for signature-based detection [11], anomaly-based detection (Kitsune [53]), attack graph analysis [44], [55], provenance-based detection (R-CAID [40]), GNN-based detection (E-GraphSAGE [49]) and LLMGRAPH. F1-Score is defined as $\frac{TP}{TP+0.5*(FP+FN)}$. FPR is defined as $\frac{FP}{FP+TN}$. FNR is defined as $\frac{FN}{FN+TP}$. As shown in Fig. 9, LLMGRAPH is more *effective* than other approaches, increases F1-Score by at least 46.9%, reduces FPR by at least 9%, and reduces FNR by at least 48.5%.

We conduct troubleshooting on the test cases, as listed in Table V. The signature-based detection are ineffective due to APT attacks adopting evasive techniques, including 0-day exploits. For anomaly-based detection [53], the False Negatives occur due to low-frequency access, such as code injection in the Sambacry attack, while it is also susceptible to FPs because of benign communications with statistical anomalies. For attack graph analysis, the FNs primarily stem from devices with unknown vulnerabilities (e.g., embedded devices in Torri, Okiru, and camera leakage scenarios). For provenance-based detection, the FNs are mainly caused by the lack of host logs, especially on legacy or embedded devices (e.g., Torri, Okiru, camera leakage, smart light blackout, water sensor overflow, and fire alarm). For GNN-based detection, the FNs result from the absence of flow context, such as IoT control context in scenarios like smart light blackout, water sensor overflow, and fire hazard. For LLMGRAPH, the FNs are mainly caused by the encrypted traffic (e.g., Okiru with an encrypted channel and heartbleed). However, LLMGRAPH mitigates this issue by integrating host logs and leveraging access facts.

LLM Sensitivity: In this part, we explore how LLMGRAPH's performance varies when using different LLMs with different parameters - including Llama3-8B (LLMGRAPH's default), Vicuna-13B, Vicuna-7B, RedPajama-7B and RedPajama-3B. These models are selected as they are known to support low-cost edge deployment such as MLC [61]. As shown in Table VI, Llama3-8B achieves the highest F1-score due to its superior inference capabilities. We also observe a consistent trend where larger parameter sizes within the same model architecture yield better performance – for example, Vicuna-13B outperforms Vicuna-7B. These results demonstrate that the LLM's inference capability significantly influences detection accuracy.

C. LLMGRAPH's Scalability

In this part, we assess the scalability of LLMGRAPH. We begin by expanding the size of a single network, adding more devices and services. As shown in Fig. 10 (top), when we increase the network size from 1 to 35 000 (considering both devices and services), the training time scales linearly and is less than 0.5 seconds. Next, we explore the impact of training multiple networks. Fig. 10 (bottom) illustrates the training time as we increase the number of networks from 1 to 1 million (each network has a fixed size of 240 devices and services). Remarkably, LLMGRAPH's training time remains within a reasonable range. For instance, training 1 million networks takes less than 1000 seconds. The scalability of LLMGRAPH is attributed to the LLM-based aggregator. By clustering and merging the communication abstractions of different networks, LLMGRAPH reduces context switching time during the learning stage (e.g., TensorFlow [22]

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.

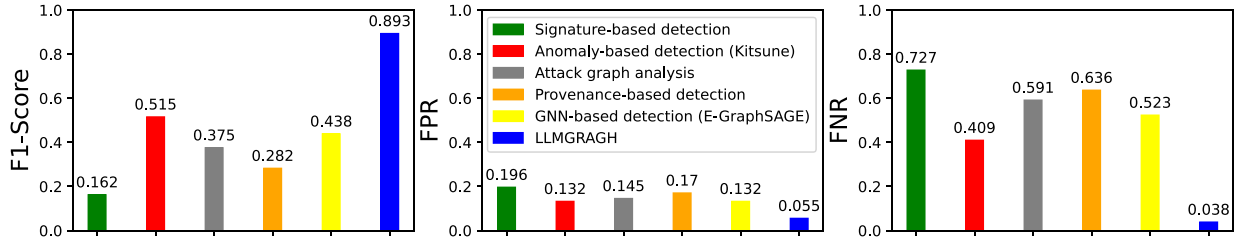

Fig. 9. Comparing single/multi-point detection, attack graph analysis, information flow analysis and LLMGRAPH.

TABLE V
TESTCASE ANALYSIS FOR DIFFERENT APPROACHES

Testcase	Signature	Anomaly	Attack graph	Provenance	GNN-based	LLMGRAPH	Key APT Path
Sambacry	part	✗	✓	✓	✓	✓	Siren - FreeNAS
Mirai	✗	part	part	✓	✓	✓	79.124.8.24/bins/sora.arm - Camera
Torri	✗	part	✗	✗	✗	✓	104.237.218.85 - WeMo
Okiru	✗	part	✗	✗	✗	part	37.48.99.233:5543 - ARC Camera
Heartbleed	✗	part	part	part	part	part	Media Server - Laptop
Camera leakage	✗	✓	✗	✗	✓	✓	gamblingonlinerealmoney - Edimax Camera
Light blackout	✗	✗	✗	✗	✗	✓	Laptop - Philips Hue Bridge
Water overflow	✗	✗	✗	✗	✗	✓	Laptop - D-Link Water Sensor
Fire hazard	✗	✗	✗	✗	✗	✓	Android Phone - iKettle

TABLE VI
LLM SENSITIVITY

LLM Models	F1-Score
Llama3-8B (LLMGRAPH's default)	0.893
Vicuna-13B	0.872
Vicuna-7B	0.841
RedPajama-7B	0.824
RedPajama-3B	0.768

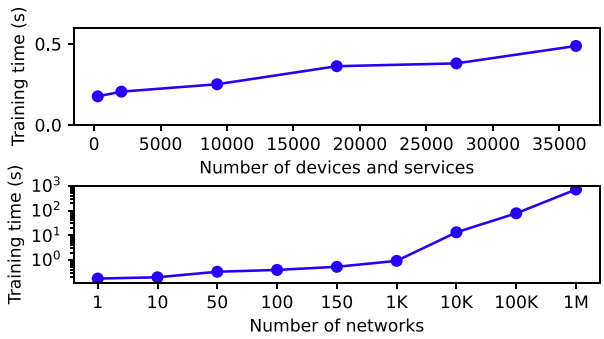

Fig. 10. The training time for LLMGRAPH on different network sizes and different number of networks.

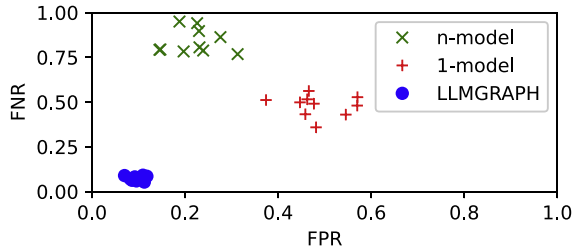

Fig. 11. FPR and FNR for 8 different edge networks.

session initiation time). Additionally, since LLMGRAPH's training time is minimal (less than 1 s for a network with 35 000 devices and services), it can provide real-time model updates when new devices join or APT emerge. This makes LLMGRAPH suitable for real-time detection and prevention.

Given real-world scalability requirements, consider an enterprise with 100 000 remote workers (comparable to a large portion of Google's 183 000-employee workforce). For such an environment spanning 100 000 home networks, our solution achieves training times between 100–1000 seconds (under 17 minutes). This duration is fully acceptable for periodic model updates. Critically, the system supports parallel deployment: Training occurs in development environments while detection runs in production SOC environments. This architecture ensures training processes introduce zero latency to real-time threat detection.

D. Handle Diverse Edge Networks

This section assesses LLMGRAPH's effectiveness across diverse edge network environments. We compare it against two baseline strategies: 1) Per-Network Models (n-model): Trains a separate GCN model for each edge network (e.g., 8 distinct models for 8 networks); 2) Unified Model (1-model): Uses a single GCN model across all networks. Fig. 11 presents the evaluation across 8 heterogeneous edge networks. The n-model strategy suffers from high False Negative Rates (FNR) due to insufficient training data per individual network. The 1-model strategy exhibits elevated False Positive Rates (FPR) caused by its inability to accommodate network diversity. Also, the GCN training time of n-model is 1.78 s, which almost 8 times higher than the 1-model's 0.23 s, due to n-model's repeated framework initialization costs (e.g., TensorFlow startup overhead). LLMGRAPH significantly outperforms both baselines, reducing FNR by 83.3% and FPR by 61.4%, with an low training time of 0.27 s (as the 8 networks are merged by the LLM Aggregator and only needs to be trained once for GCN).

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.

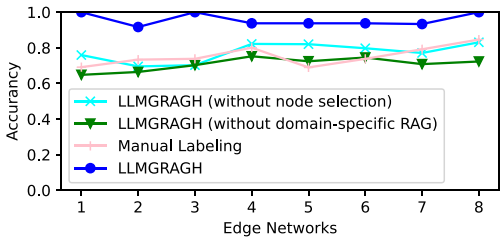

Fig. 12. Benefit of the LLM-based annotator.

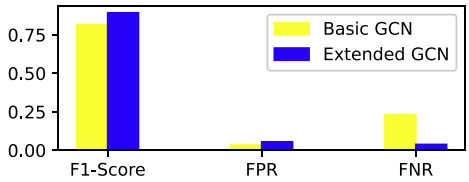

Fig. 13. Benefit of LLMGRAPH's extended GCN.

E. Benefit of LLMGRAPH's Components

LLM-based Annotator: We compare the LLM-based annotator to manual labeling. We conduct an IRB-approved study and invited ten security personal[4] with vulnerability mining competition experiences to manually label devices and network flows across eight edge networks. Fig. 12 illustrates the labeling accuracy for each network. Overall, LLMGRAPH's LLM-based annotator achieves a 27.09% accuracy improvement comparing with manual labeling. We also found that the performance of manual labeling is heavily impacted by the scale of the network. For instances, the accuracy of manual labeling is low on large networks such as network 5 (around 1000 devices and flows) and network 6 (around 500 devices and flows).

We conduct an ablation study to evaluate the contribution of each component in the LLM-based annotator. As illustrated in Fig. 12, we separately remove the domain-specific RAG module and the node selection module. Results demonstrate that both components significantly enhance performance, with the RAG module providing greater accuracy gains. Specifically, integration of the domain-specific RAG module boosts accuracy by 35.2%, and inclusion of the node selection module improves accuracy by 23.6%. This confirms the RAG module's marginally higher contribution to overall accuracy.

Extended GCN: We compare LLMGRAPH's extended GCN with basic GCN in terms of F1-Score, FPR, and FNR, as depicted in Fig. 13. In this experiment, the basic GCN is performing per-device level detection, and LLMGRAPH's extended GCN is performing per-flow level detection. The F1-Score, FPR, and FNR are calculated at per-device level for basic GCN and at per-flow level for LLMGRAPH's extended GCN. Compared with basic GCN, the extended GCN yields a 9.4% increase in F1-Score and an 83.5% reduction in FNR, with a reasonable low FPR below 0.06. LLMGRAPH's extended GCN can effectively reduce the FNR, as the flow context significantly enhances the detection of malicious cases. The more fine-grained per-flow level detection cause the FPR to increase, but is reasonably low below 0.06. Overall, the extended GCN can provide more accurate detection at a finer-granularity than the basic GCN.

[4]Each personal was given one masked ID, and no personal information was gathered in the IRB-approved study.

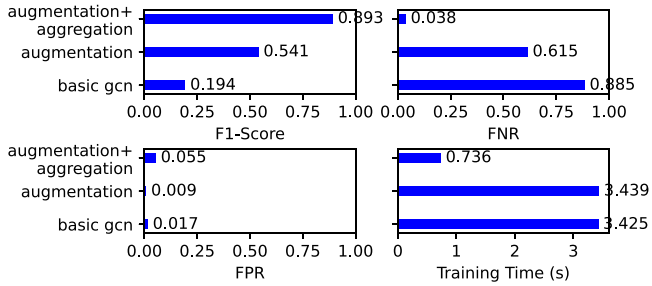

Fig. 14. Benefit of the LLM-based aggregator.

LLM-based Aggregator: In this section, we assess the benefits of LLMGRAPH's LLM-based Aggregator, which includes two enhancements - data augmentation and network aggregation. In the evaluation process, we start with basic GCN, and incrementally adding each enhancement. As shown in Fig. 14, the combined effect of data augmentation and network aggregation leads to a 69.9% improvement in F1-Score, an 84.7% reduction in FNR, and an FPR below 0.055. Additionally, training time decreases significantly—from the basic GCN's 3.4 seconds to LLMGRAPH's 0.74 seconds (for 45 networks with a maximum size of 398 devices and services), resulting in a 78% reduction in training time. From basic GCN to *data augmentation*, the improvements are mainly because the data augmentation balances the biased ratio of malicious cases and benign cases, and enhances the "attention" to the malicious devices and services and their boundary with the benign neighbors. For *network aggregation*, the main improvements are because the edge networks are clustered and merged. The testcases are shared and each cluster has a dedicated model, which helps to address the insufficient data issue and the variance issue across *diverse* edge networks.

IX. Related Works

Anomaly-based detection: Anomaly-based detection identifies security threats through deviations from established statistical baselines in network traffic patterns [24], [25], [35], [38], [48], [53]. This approach encompasses several specialized methodologies: Kitsune [53] employs autoencoders to detect anomalies by measuring reconstruction errors (e.g., RMSE) in traffic feature representations. Complementary sampling-based techniques include Bartos et al.'s random forest classifier [25] and FlowLen's [24] ensemble learning applied to sampled traffic flows. For periodic traffic analysis, Whisper [38] utilizes spectral decomposition in the frequency domain to identify deviations from normal flow periodicity signatures. Beyond network traffic, DeepLog [35] applies recurrent neural networks to model temporal patterns in system logs for anomalous event detection, while Jaqen [48] specializes in DDoS identification through probabilistic flow sketching. However, it's important to note that these anomaly-based detection methods primarily focuses on

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.

statistical anomalies and may still be vulnerable to APTs with evasive techniques such as low-frequency penetration.

Attack graph analysis: Attack graph analysis [44], [55] provides a formal methodology for assessing system security by generating a comprehensive representation of all potential paths an intruder could exploit to achieve their objectives within a system. Constructing such graphs necessitates detailed knowledge of known device vulnerabilities, user privilege assignments, and explicit assumptions regarding the attacker's specific goals. Phillips and Swiler [55] propose the basic concept of attack graphs with an "attack-centric" view of the system, which models the sequences of malicious events initiated by the attacker, representing the system's state transitions solely through adversarial actions. Jha et. al. [44] proposed a more generic language with a wider range of system events (such as failure of a link and user errors) and attacks occur simultaneously. However, it's important to note that the attack graph approach heavily relies on existing knowledge of device vulnerabilities and assumes clear attacker goals, which is hard to achieve in millions of edge networks.

Provenance-based detection: Provenance-based detection leverages data lineage tracing to analyze the origins and history of system events, offering valuable insights for intrusion detection [40], [43], [47], [60]. Several notable implementations advance this approach: R-CAID [40] integrates Root Cause Analysis (RCA) by precomputing node-level root causes during graph construction and embedding these causal relationships directly. TagS [47] provides a unified representation of system behavior through provenance graphs, designed for extensibility to diverse environments including containers and microservices. Meanwhile, P-EDR [34] models event-level data and control dependencies within its provenance framework. Despite their strengths, these methods face significant operational limitations. They typically require manual labeling efforts (e.g., classifier training in R-CAID) and encounter scalability constraints due to the processing demands of voluminous system logs, presenting challenges for real-time deployment.

Series of GNN-based detection: Graph Neural Networks (GNNs) represent a growing frontier in network intrusion detection systems, offering promising approaches to model complex relational data inherent in network infrastructures. This research trajectory includes several notable contributions: Friji et al. [37] established a GNN framework that classifies communication flows through maliciousness scoring, providing a graph-based methodology for threat assessment. Another significant advancement, StrucTemp-GNN [26], specifically addresses the unique challenges of IoT ecosystems by jointly modeling structural topology and temporal dependencies, enabling more accurate real-time intrusion detection in dynamic edge environments. Further advancing this domain, E-GraphSAGE [49] enhances detection capabilities by explicitly leveraging the inherent hierarchical relationships within graph-structured network data, optimizing neighborhood aggregation techniques to improve pattern recognition in security-relevant graph representations. Despite these innovations, conventional GNNs (e.g., GCN [46], GraphSAGE [41]) fundamentally fail to capture essential network flow context and directional relationships due to adjacency matrix constraints [23]. While costly extensions like Hodge Laplacians partially embed edge attributes, they remain impractical for analyzing multiplexed traffic flows on single edges [58], [67]. A critical operational limitation is the dependency on significant labeled node subsets (5% as per GCN [46]) – a requirement logistically infeasible in edge networks where comprehensive ground truth is unavailable, creating real-world deployment barriers.

X. Discussion

Handling encrypted communication: A considerable portion of communications in edge networks are encrypted. LLMGraph mitigates this issue in two ways. The first way is to leverage network flow metadata, providing correlations between devices and network flows in GCN adjacency matrix. Another way is to integrate host logs, incorporating this information into the GCN nodes. We show the effectiveness of such mitigation in Section VIII.

Handling adversarial ML attacks: Recent studies [30], [50], [51], [59], [66] have highlighted the susceptibility of Graph Neural Networks (including GCN) to adversarial ML techniques. The adversarial ML techniques' focus is to use flow perturbations to cause FNs and FPs in the GNN-based detection. LLMGraph provides a practical protective mechanism to disrupt the adversary's flow perturbations capability via setting limits on perturbation budget and introduce randomness in graph augmentation, as shown in our supplementary materials. We will future explore this direction in the future.

Relation with Federated Multi-Task learning: Recent federated multi-task learning mechanisms [57] also aims at solving the insufficient data and diversity issue. However, federated multi-task learning is a distributed learning process that requires an effective local learning model (such as SVM) as the base model. Therefore, LLMGraph may serve as the base model for federated multi-task learning and we will explore the possibility of combination in the future.

XI. Conclusion

Remote working from weakly secured edge networks faces severe threats from APTs. Current security detection systems, including signature-based detection, anomaly-based detection and graph-based detection, are ill-equipped for addressing APTs considering the scale and complexity of edge networks. In this paper, we propose LLMGraph- a learning mechanism that combines LLM and GCN to achieve label-free detection against APTs in edge networks. We show LLMGraph is effective & scalable and is a promising component (e.g., deployed in edge security gateways) to help secure edge networks. We also acknowledge that there is still room for improvement for LLMGraph in handling encrypted communication and handling adversarial ML attacks. We will explore these directions further in future works.

Acknowledgement

The authors would like to thanks Ziyi Zhou and Fudu Xing for their support on this work.

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] 7-year-old samba flaw lets hackers access thousands of linux PCs remotely, 2017. [Online]. Available: https://thehackernews.com/2017/05/samba-rce-exploit.html

[2] Hacking team hack, 2018. [Online]. Available: https://gist.github.com/Sjord/ac8dfff3a3ac3180c065f370f24b30a8

[3] Amazon eero, 2021. [Online]. Available: https://eero.com/

[4] Cisco meraki, 2021. [Online]. Available: https://meraki.cisco.com/

[5] Netgear orbi, 2021. [Online]. Available: https://www.netgear.com/home/wifi/mesh/

[6] Virustotal, 2021. [Online]. Available: https://www.virustotal.com/gui/

[7] Getting started with gns3, 2022. [Online]. Available: https://docs.gns3.com/docs/

[8] ChatGPT, 2023. [Online]. Available: https://openai.com/chatgpt

[9] Microsoft copilot, 2023. [Online]. Available: https://www.microsoft.com/en-us/microsoft-copilot

[10] Redpajama, a project to create leading open-source models, 2023. [Online]. Available: https://www.together.ai/blog/redpajama

[11] Snort, 2023. [Online]. Available: https://www.snort.org/

[12] Vicuna model, 2023. [Online]. Available: https://huggingface.co/lmsys/vicuna-13b-v1.5

[13] What is the log4j vulnerability?, 2023. [Online]. Available: https://www.ibm.com/topics/log4j

[14] Wormgpt: An AI tool for hackers, 2023. [Online]. Available: https://www.popularmechanics.com/technology/security/a45533297/what-is-wormgpt/

[15] Faiss, 2024. [Online]. Available: https://faiss.ai/

[16] Google safe browsing, 2024. [Online]. Available: https://safebrowsing.google.com/

[17] Hackergpt, 2024. [Online]. Available: https://github.com/Hacker-GPT/HackerGPT

[18] Langchain, 2024. [Online]. Available: https://www.langchain.com/

[19] Llama 3, 2024. [Online]. Available: https://ollama.com/library/llama3

[20] Ollama, 2024. [Online]. Available: https://ollama.com/

[21] Protecting the perimeter with vt intelligence - malicious urls, 2024. [Online]. Available: https://blog.virustotal.com/2023/12/protecting-perimeter-with-vt_18.html

[22] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Des. Implementation*, USENIX Association, 2016, pp. 265–283.

[23] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," 2020, *arXiv:2006.05205*.

[24] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, and A. Madeira, "FlowLens: Enabling efficient flow classification for ML-based network security applications," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 66–84.

[25] K. Bartos, M. Sofka, and V. Franc, "Optimized invariant representation of network traffic for detecting unseen malware variants," in *Proc. USENIX Secur. Symp.*, 2016, pp. 807–822.

[26] I. E. Boukari, I. A. Derdouha, S. Bouzefrane, L. Hamdad, S. Nait-Bahloul, and T. Huraux, "Structemp-GNN: An intrusion detection framework in IoT networks using dynamic heterogeneous graph neural networks," in *Proc. Int. Conf. Mobile Secure Program. Netw.*, Springer, 2023, pp. 17–39.

[27] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *Proc. 2nd Int. Conf. Learn. Representations*, 2014, pp. 1312–1326.

[28] J. Bruna, W. Zaremba, A. D. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," 2014, *arXiv:1312.6203*.

[29] Z. B. Celik, G. Tan, and P. Mcdaniel, "IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT," in *Proc. 2019 Netw. Distrib. Syst. Secur. Symp.*, 2019, pp. 326–341.

[30] H. Chang et al., "A restricted black-box adversarial framework towards attacking graph embedding models," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 3389–3396.

[31] T. Chen et al., "Dense x retrieval: What retrieval granularity should we use," 2023, *arXiv:2312.06648*.

[32] Z. Chen et al., "Label-free node classification on graphs with large language models (LLMs)," in *Proc. Int. Conf. Learn. Representations*, 2024, pp. 31632–31655.

[33] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 3844–3852.

[34] F. Dong et al., "Are we there yet? An industrial viewpoint on provenance-based endpoint detection and response tools," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2023, pp. 2396–2410.

[35] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. 2017 ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1285–1298.

[36] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "FlowFence: Practical data protection for emerging IoT application frameworks," in *Proc. USENIX Secur. Symp.*, 2016, pp. 531–548.

[37] H. Friji, A. Olivereau, and M. Sarkiss, "Efficient network representation for GNN-based intrusion detection," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.*, Springer, 2023, pp. 532–554.

[38] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime robust malicious traffic detection via frequency domain analysis," in *Proc. 2021 ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 3431–3446.

[39] S. Garcia, A. Parmisano, and M. J. Erquiaga, "IoT-23: A labeled dataset with malicious and benign IoT network traffic," 2020.

[40] A. Goyal, G. Wang, and A. Bates, "R-CAID: Embedding root cause analysis within provenance-based intrusion detection," in *Proc. IEEE Symp. Secur. Privacy*, 2024, pp. 257–257.

[41] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1025–1035.

[42] W. He et al., "Rethinking access control and authentication for the home Internet of Things (IoT)," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 255–272.

[43] M. A. Inam et al., "SoK: History is a vast early warning system: Auditing the provenance of system intrusions," in *Proc. IEEE Symp. Secur. Privacy*, 2023, pp. 2620–2638.

[44] S. Jha, O. Sheyner, and J. Wing, "Two formal analyses of attack graphs," in *Proc. 15th IEEE Comput. Secur. Found. Workshop*, 2002, pp. 49–63.

[45] Y. J. Jia et al., "ContexIoT: Towards providing contextual integrity to appified IoT platforms," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2017.

[46] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks, in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 11305–11312.

[47] Z. Li et al., "TagS: Real-time intrusion detection with tag-propagation-based provenance graph alignment on streaming events," 2024, *arXiv:2403.12541*.

[48] Z. Liu et al., "Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric DDoS attacks with programmable switches," in *Proc. USENIX Secur. Symp.*, 2021, pp. 3829–3846.

[49] W. W. Lo, S. Layeghy, M. Sarhan, M. Gallagher, and M. Portmann, "E-GraphSAGE: A graph neural network based intrusion detection system for IoT," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, 2022, pp. 1–9.

[50] J. Ma, S. Ding, and Q. Mei, "Towards more practical adversarial attacks on graph neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 4756–4766.

[51] Y. Ma, S. Wang, T. Derr, L. Wu, and J. Tang, "Graph adversarial attack via rewiring," in *Proc. 27th ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2021, pp. 1161–1169.

[52] M. Miettinen et al., "IoT SENTINEL: Automated device-type identification for security enforcement in IoT," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 2177–2184.

[53] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," 2018, *arXiv: 1802.09089*.

[54] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Comput. Netw.*, vol. 31, pp. 2435–2463, 1999.

[55] C. Phillips and L. P. Swiler, "A graph-based system for network-vulnerability analysis," in *Proc. 1998 Workshop New Secur. Paradigms*, 1998, pp. 71–79.

[56] V. L. Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," 2018, *arXiv: 1806.01156*.

[57] V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4427–4437.

[58] H. Sun, X. Li, D. Su, J. Han, R.-H. Li, and G. Wang, "Towards data-centric machine learning on directed graphs: A survey," 2024, *arXiv:2412.01849*.

[59] Y. Sun, S. Wang, X. Tang, T.-Y. Hsieh, and V. Honavar, "Adversarial attacks on graph neural networks via node injections: A hierarchical reinforcement learning approach," in *Proc. Web Conf.*, 2020, pp. 673–683.

[60] M. Surbatovich, J. Aljuraidan, L. Bauer, A. Das, and L. Jia, "Some recipes can do more than spoil your appetite: Analyzing the security and privacy risks of IFTTT recipes," in *Proc. 26th Int. Conf. World Wide Web*, 2017, pp. 1501–1510.

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.

[61] MLC team, "MLC LLM: Universal LLM deployment engine with ML compilation," 2023. [Online]. Available: https://github.com/mlc-ai/mlc-llm
[62] R. L. Thorndike, "Who belongs in the family," *Psychometrika*, vol. 18, pp. 267–276, 1953.
[63] Y. Tian et al., "SmartAuth: User-centered authorization for the Internet of Things," in *Proc. USENIX Secur. Symp.*, 2017, pp. 361–378.
[64] H. Touvron et al., "LLaMA: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.
[65] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," 2017, *arXiv: 1710.10903*.
[66] X. Wan, H. Kenlay, R. Ru, A. Blaas, M. A. Osborne, and X. Dong, "Adversarial attacks on graph classifiers via Bayesian optimisation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 6983–6996.
[67] Y. Zhou et al., "Co-embedding of edges and nodes with deep graph convolutional neural networks, *Sci. Rep.*, vol. 13, no. 1, 2023, Art. no. 16966.

Tianlong Yu received the PhD degree in computer science from Carnegie Mellon University. He is an associate professor with the School of Artificial Intelligence, Hubei University. He was awarded the Symantec Fellowship, in 2017. He has published extensively in top-tier journals and conferences, including NDSS and NSDI. After completing his PhD, he joined Palo Alto Networks to develope cybersecurity products for edge networks. He has extensive experience in translating research findings into practical products. His current research interests include network security, distributed machine learning, and large language models.

Gaoyang Liu received the PhD degree from the School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan, China, in 2021. He is currently a postdoctoral researcher with the School of Computing Science, Simon Fraser University, British Columbia, Canada. His research interests include security and machine learning.

Chen Wang (Senior Member, IEEE) received the BS and PhD degrees from the Department of Automation, Wuhan University, China, in 2008 and 2013, respectively. From 2013 to 2017, he was a postdoctoral research fellow with the Networked and Communication Systems Research Lab, Huazhong University of Science and Technology, China. Thereafter, he joined the faculty of the Huazhong University of Science and Technology where he is currently an associate professor. His research interests are in the broad areas of wireless networking, Internet of Things, and mobile computing, with a recent focus on privacy issues in intelligent systems. He is a senior member of ACM.

Yang Yang received the bachelor of engineering and master of science degrees from the Wuhan University of Technology, in 2009 and 2012, respectively, and the PhD degree in computer science from the Huazhong University of Science and Technology, in 2017, under the supervision of Professor Hongbo Jiang. He is an associate professor with the School of Artificial Intelligence, Hubei University. With the support of the China Scholarship Council (CSC), he visited Simon Fraser University in Canada, in 2013, where he worked under the guidance of Professor Jiangchuan Liu. His research interests are primarily in networking and machine learning.

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on December 09,2025 at 04:14:58 UTC from IEEE Xplore. Restrictions apply.