Always be Pre-Training: Representation Learning for Network Intrusion Detection with GNNs

Hidden for blind-review

***Abstract*—Graph Neural Network (GNN)-based Network Intrusion Detection Systems (NIDS) have recently demonstrated state-of-the-art performance on benchmark datasets. Nevertheless, these methods suffer from a reliance on target encoding for data pre-processing, limiting widespread adoption due to the associated need for annotated labels—a cost-prohibitive requirement. In this work, we first summarize related work on GNN-based NIDS, discussing their limitations. Moreover, we propose a solution involving in-context pre-training and the utilization of dense representations for categorical features to jointly overcome the label-dependency limitation. Our approach exhibits remarkable data efficiency, achieving over 98% of the performance of the supervised state-of-the-art with less than 4% labeled data on the `NF-UQ-NIDS-V2` dataset. Furthermore, we also shed light on the avenues for future research in this direction.**

***Index Terms*—Computer networks, intrusion detection, machine learning, graph neural networks, NIDS, few-shot learning, self-supervised learning, IoT**

I. Introduction

A Network Intrusion Detection System (NIDS) [1] monitors the traffic on a computer network to detect anomalous or malicious activities. Undetected intrusions pose threats to the confidentiality, integrity, and availability of computer systems [2]. Unlike Host Intrusion Detection Systems (HIDS) [3], which focus on monitoring system telemetry on individual hosts, NIDS takes a distinct approach. It observes and analyzes the traffic passing through a dedicated point in the network as illustrated in Fig. 1, triggering alerts to downstream threat management when abnormal or suspicious traffic is detected.

With the rise of the Internet-of-Things (IoT), comprised of compute-constrained and low-power devices that are typically unable to run dedicated HIDS, there is a growing demand for centralized intrusion detection systems, such as NIDS [4].

A traditional signature-based NIDS compares network forensics to a set of predefined rules and patterns to identify traces that might indicate an attack or intrusion. The rules need to be hand-crafted for each deployment environment and are fundamentally unable to detect novel or zero-day attacks [2].

More recently, research has focused on intrusion detection as a classification problem in Machine Learning (ML) [4]–[6]. ML-based detectors can be trained to recognize baseline patterns in network traffic, enabling them to identify threats beyond known examples. Synthetic datasets [7]–[11] have been utilized to evaluate ML solutions for NIDS, as real traffic data is difficult to make public for security concerns [12]. Additionally, real traffic data contains a minuscule proportion of malicious examples, which can still be cost-prohibitive to label. Such imbalance between benign and malicious activities can also pose challenges to model training [13].

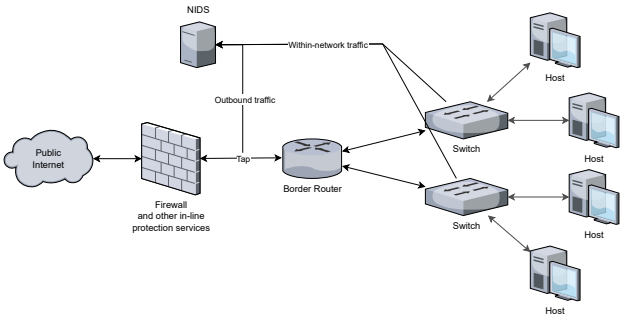

Fig. 1. A monitoring deployment of NIDS, centralizing intrusion detection by monitoring the network of activities [2].

Recently, intrusion detectors based on Graph Neural Networks (GNNs) [14]–[17] have achieved state-of-the-art performance on synthetic benchmarks. GNN-based NIDS construct graph representations of the unit of activity they monitor. Such graphs can represent low-level activities like packet behavior [18] or high-level network flow connections such as Hypertext Transfer Protocol (HTTP) requests [8]. The latter consider topologically related activities (e.g., other network flows sharing a common source or destination node) while detecting intrusion on a single record of network activity (e.g., network flow). This enables the classifiers to better detect coordinated attacks or those involving a chain of actions, such as Distributed Denial of Service (DDoS) [19] and Man-In-The-Middle (MITM) [20] attacks.

Supervised [14], [18], [21], [22] and unsupervised [15], [23] GNN-based NIDS have been proposed. In supervised methods, the classifier predicts the labeled class for each training example. In unsupervised methods, the model learns to encode data distribution without specific label guidance, subsequently using these encodings to establish decision boundaries.

A. State-of-the-Art and their Limitations

While GNN-based NIDS have achieved state-of-the-art performance, they suffer from the following limitations.

1) Challenges in Categorical Feature Handling: Many GNN-based NIDS [14]–[16] boast state-of-the-art performance on heterogeneous synthetic datasets [8], [10], [11]. These dataset are dubbed heterogeneous for their inclusion of features associated with different network protocols such as HTTP, secure sockets layer, and the internet protocol suite (TCP/IP). These features are usually categorical and contain rich semantics for traffic behavior [8], [24]. However, current state-of-the-art GNN-based methods either drop or reduce these categorical features to scalar-valued target encodings,

which computes the empirical correlation between categories and labels during pre-processing. Such practice potentially loses vital information for intrusion detection, which could lead to poor generalization under distribution shift.

2) Label Dependency: Obtaining and maintaining an accurately labeled dataset representative of the target domain can be cost-prohibitive, posing a central challenge in NIDS research [12]. As an example, recreating a dataset such as `ToN-IoT` [8] would require an organization to identify and label $216,043$ malicious flows within their own environment.

Unsupervised methods offer a solution to the challenge of requiring labeled data for ML model training. However, existing GNN-based unsupervised methods, such as *Anomal-E* [15], preprocess categorical features using target-encoding. This method computes the empirical correlation between categories and label classes, thus still relying on labels.

B. Our Contributions

In this study, we investigate approaches for representing categorical features in GNN-based NIDS to mitigate label-dependency limitations. Initially, we examine the representation of these features using *dense vector embeddings*, as they are more expressive than scalar target encoding. However, we find that directly learning such embeddings through supervised learning leads to poor performance in some settings, evidenced by a performance gap in F1-score of approximately 7%. Subsequently, we introduce ***in-context pre-training and fine-tuning***, a training procedure that initially trains a GNN encoder on unlabeled data from a network before fine-tuning on a small sample of labeled data from the same network, by theorizing that the initial Self-Supervised Learning (SSL) would better initialize the weights before supervised training. We show that this technique not only improves performance compared to target encoded representations but also achieves remarkable data-efficiency.

Key-results: Considering the `ToN-IoT` [8] and `NF-UQ-NIDS-V2` [11] datasets (discussed in Section III-A), we observe that the pre-trained model retains over 95% of its performance on the full dataset when fine-tuned on less than 4% of the labeled data. In-context pre-training thus offers an alternate solution to the label dependency issue through *few-shot learning*. Additionally, pre-trained GNNs can be easily adapted for multi-way classification with a tractable amount of labeled examples, whereas unsupervised methods have only been shown suitable for binary classification.

The paper's structure is as follows: Section II summarizes related works in NIDS, and Section III provides an overview of benchmarks, GNNs, and SSL. Section IV introduces in-context pre-training and fine-tuning methods with dense representation, Section V details experimental configurations and results, and Section VI concludes and discusses future works.

II. Related Works in GNN-based NIDS

This section surveys GNN-based NIDS approaches, including supervised and unsupervised methods, and briefly touches on cross-domain NIDS related to label dependence.

A. Supervised GNN-based NIDS

E-GraphSAGE [14] is the first general-purpose GNN-based NIDS. The method proposed a simple GraphSAGE-like [25] model that propagates edge feature information across the neighborhood. *E-ResGAT* [16] proposed a graph attention network with residual connections operated on line graphs (a graph transformation that convert nodes to edges and vice versa) and showed that the residual connection significantly improved the performance on minority classes. State-of-the-art performance is attained by [21], which proposed a three-layer GNN consisting of two spectral layers as the first and last layer and a spatial aggregation in the middle. *NE-GConv* [22] is a GNN-based NIDS that incorporates both node and edge features. In [26], the authors considered the temporal evolution of network traffic and proposed a NIDS system based on spatio-temporal GNNs. In [17], classification performance is improved by training a supervised model alongside unsupervised objectives that consider the traffic quantity at each node.

In [18], the authors proposed a *Graph2Vec+* random forest NIDS, performing flow-level classification by constructing graphs for packet burst behavior [27]. By focusing on low-level packet behavior rather than extracting statistical features, the system achieves high data efficiency, retaining over 95% performance with only 10% of the training data.

B. Unsupervervised GNN-based NIDS

Anomal-E [15] is the first unsupervised GNN-based NIDS. It learns meaningful representations for edges by applying Deep Graph Infomax (DGI) [28], a contrastive self-supervised learning algorithm that maximizes the mutual information between local and global representation of the communication graph. The learned edge embeddings are then fed to traditional unsupervised classification methods such as isolation forest and Cluster-Based Local Outlier Factor (CBLOF) [29] for intrusion detection. In [23], the authors proposed *ARGANIDS*, where an Adversarially Regularized Graph Autoencoder (ARGA) [30] and its variants are first pre-trained on unlabeled data. The learned embeddings from the encoder are then used to train a random forest to classify network flows.

Both approaches [15], [23] train unsupervised models without subsequent fine-tuning, neglecting the optimization of pre-trained model weights through supervised learning.

C. Cross-Domain NIDS

The widespread adoption of supervised ML-based NIDS has been hindered by the overall lack of available data. This is in part due to obstacles such as organizations being discouraged from sharing their expert-labeled data due to security concerns [12]. One strategy to overcome this limitation is to develop ML models that can generalize across various deployment domains. In [31], the authors demonstrated that current ML-based NIDS perform poorly across different domains of network traffic. Subsequently, [32] proposed training domain-generalizable ML-based NIDS by utilizing only domain-invariant features.

TABLE I
COMMONLY USED NOTATIONS

Symbol	Definition
$G = (\mathcal{V}, \mathcal{E})$	Graph G with nodes $\mathcal{V}$ and edges $\mathcal{E}$
$x_u^{(k)}$	Vector representation of node u at layer k
h_u, h_{uv}	Final node/edge representation (embedding)
d_x, d_e	GNN node/edge hidden dimension
a	Aggregated message
K	Total # of GNN layers
x_i	One feature of input x
$\mathbf{e}_i$	One-hot vector
e_{uv}	Features of edge $(u, v) \in \mathcal{E}$
$\mathcal{N}(u)$	The set of all nodes v such that $(v, u) \in \mathcal{E}$
$\gamma^{(k)}, \phi^{(k)}, \bigoplus$	Update, message function, aggregation operator
f_θ, q_ξ	Encoder & decoder in SSL
$\mathcal{D}$	Data sample
$\mathcal{S}$	Similarity function in DGI
$\overline{s}$	Graph summary
$\mathcal{C}(G)$	Corrupted graph
$\tilde{h}_{uv}$	finaledge representation from the corrupted graph
ω, φ	Embedding/interaction function
W, W_1, W_2, W_s	Weight matrices

III. BACKGROUND

This section provides essential background information on NIDS dataset benchmarks, GNNs, and SSL. The commonly used notations across the paper are defined in Table I.

A. Heterogeneous NIDS Benchmarks

By utilizing various sources of network activities, heterogeneous NIDS datasets allow intrusion detectors to construct a more comprehensive and multi-dimensional understanding of network dynamics, which could allow the NIDS to adapt to new and emerging threats by considering a broad spectrum of indicators and behaviors [33].

ToN-IoT [8] was the first to propose a heterogeneous NIDS dataset. ToN-IoT's heterogeneity is reflected by the inclusion of 40 features across various network protocols. Specifically, the dataset contains connection features from Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) at the transport layer, statistical summaries of TCP/IP activities at internet and transport layers, secure sockets layer states also at the transport layer, and Domain Name System (DNS) and HTTP activities (application layer). ToN-IoT consists of 9 attack types, including backdoor, cross-site scripting (XSS), password cracking, MITM, Denial of Service (DoS), DDoS, scanning, ransomware, and injection attacks. There are $461,043$ total records in the dataset.

NF-UQ-NIDS-V2 [11] designs 43 standardized NetFlow [34] features across four datasets: UNSW-NB15 [7], ToN-IoT [8], CSE-CIC-IDS2018 [9], and BoT-IoT [35]. The authors compile NF-UQ-NIDS-V2 by extracting the newly designed features from the raw packet capture files from the four aforementioned datasets. Like ToN-IoT, the new feature set contains various network protocols such as TCP, IP, UDP, the Internet Control Message Protocol (ICMP), and secure sockets layer. In addition, the dataset includes other internet and application layer protocols such as DNS and the File Transfer Protocol (FTP). NF-UQ-NIDS-V2 contains $75,987,976$ records in total with 20 attack categories.

Earlier datasets, like UNSW-NB15, CSE-CIC-IDS2018, and BoT-IoT, primarily contain statistical features summarizing generic network connections. UNSW-NB15 has 40 out of 49 numeric features, while CIC-IDS-2018 and BoT-IoT exclusively consist of numeric features. In contrast, ToN-IoT and NF-UQ-NIDS-V2 are predominantly composed of categorical features describing protocol activities. ToN-IoT has 31 out of 40 categorical features, and NF-UQ-NIDS-V2 has 15 out of 43 categorical features.

B. Graph Neural Networks (GNNs)

A key feature of GNNs is their capability to learn rich representations of graphs through *message passing* across a graph's topological structure. More specifically, this means combining the representation of each node with an aggregation of neighboring node features to generate a new representation for the node at each layer. This process is usually done multiple times through the different layers of GNN. Formally, this message passing framework is defined by three abstract functions at each layer: *message function* $\phi^{(k)}$, *aggregation operator* $\bigoplus$, and *update funciton* $\gamma^{(k)}$. One can compute the next layer representation by the following formula.

$$x_u^{(k)} = \gamma^{(k)} \left(x_u^{(k-1)}, \bigoplus_{v \in \mathcal{N}(u)} \phi^{(k-1)} \left(x_u^{(k-1)}, x_v^{(k-1)}, e_{vu} \right) \right) \tag{1}$$

Here, $x_u^{(k)} \in \mathbb{R}^{d_x}$ is the hidden representation of node u at the k-th layer. Usually, $x_u^{(0)}$ is the node's feature vector associated with node u. The message function takes edge features $e_{vu} \in \mathbb{R}^{d_e}$ and the hidden representations of any neighbor v of u as input. The *aggregation* operator combines all messages from the neighboring nodes $\mathcal{N}(u)$. $\gamma^{(k)}$ is the *update* function that outputs the next layer hidden node representation by taking the aggregated message and the node representation from the previous layer. Different GNNs mainly differ based on the selection of the message, update, and aggregation functions.

1) E-GraphSAGE: E-GraphSAGE [14] encodes edge information in node representations by propagating both node and edge information in its message function. Formally, the message function of E-GraphSAGE is defined as

$$\phi\left(x_u, x_v, e_{vu}\right) = W_1[x_v; e_{vu}], \tag{2}$$

where $[x_v; e_{vu}]$ is the concatenation of vector x_v and e_{vu} and W_1 is a learnable matrix. Note that the message function output in E-GraphSAGE is invariant to the input central node x_u. The aggregation operation simply averages the messages:

$$a := \bigoplus_{v \in \mathcal{N}(u)} \phi_v = \sum_{v \in \mathcal{N}(u)} \phi_v \tag{3}$$

Finally, the aggregated message is combined with center node x_u to compute the next layer of representation,

$$\gamma(x_u, a) = \sigma(W_2[x_u; a]), \tag{4}$$

where σ is a non-linear activation function. Equation 2 and 4 together ensure that the updated node representation includes information from the center node representation x_u, edge features e_{uv}, and the neighboring node representations x_v.

In E-GraphSAGE with K layers, the final layer node embeddings $h_u := x_u^{(K)}$ of neighboring node representations are concatenated to form edge embeddings $h_{uv} = [h_u; h_v]$.

C. Graph Self-Supervised Learning (SSL)

SSL is an ML paradigm that trains models by fitting the inherent structure or characteristics within the data through a self-supervised objective $\mathcal{L}_{SSL}$ without any annotated labels. In the context of network intrusion detection, SSL allows the model to learn meaningful representations from the ubiquitous unlabeled traffic data without explicit labels for different types of intrusions. Graph SSL methods employ an encoder f_θ to transform input data into a latent representation and a decoder q_ξ to convert the latent representation into a training signal on data sample $\mathcal{D}$ with the help of the objective function. Graph SSL can be formulated as the following optimization problem;

$$\theta^*, \xi^* = \arg\min_{\theta,\xi} \mathcal{L}_{SSL}(f_\theta, q_\xi, \mathcal{D}). \tag{5}$$

The output embedding from the optimized encoder f_{θ^*} can be used as input features for downstream classifiers [15], [23] or the encoder can be further trained for the same purpose.

Existing GNN-based NIDS research has considered two types of graph SSL techniques: contrastive and generative. Both methods learn to encode the input graph into a lower-dimensional embedding with the encoder f_θ. However, contrastive methods supervise the encoder training with a discriminative decoder q_ξ that maximizes the difference between the embeddings of positive and negative examples, while generative methods do so by using the decoder to reconstruct the input from the embedding. At the time of writing, two SSL-based NIDS has been proposed: Anomal-E [15] and ARGANIDS [23]. The former leverages DGI [28], a constrastive SSL technique, while the latter employs ARGA (and its variants) [30], a generative SSL technique. Our experiments do not consider ARGANIDS as it is not suitable for reconstructing one-hot input features. Consequently, we only describe Anomal-E in details in this section.

1) Anomal-E: DGI learns meaningful representation of graphs by maximizing mutual information between the local and global representations of a graph. In practice, this is achieved by maximizing the similarity $\mathcal{S}$ between the graph summary $\overline{s}$ and edge embeddings h_{uv} on the original graph G and minimizing the similarity between the same vectors on a corrupted graph $\mathcal{C}(G)$.

$$\begin{aligned}\mathcal{L}_{\text{Anomal-E}} = -\frac{1}{2n}\sum_{i=1}^{n}\sum_{(u,v)\in\mathcal{E}} &\log\left(\mathcal{S}(\overline{s}, h_{uv})\right) \\ &+ \log\left(1 - \mathcal{S}(\overline{s}, \tilde{h}_{uv})\right),\end{aligned} \tag{6}$$

The edge embeddings h_{uv} and $\tilde{h}_{uv}$ are outputs from an E-GraphSAGE encoder f_θ on the original graph G and corrupted graph $\mathcal{C}(G)$, respectively. The graph summary $\overline{s}$ is defined as the sum of edge embeddings:

$$\overline{s} = \sum_{(u,v)\in G} h_{uv}. \tag{7}$$

The corruption function $\mathcal{C}$ in Anomal-E randomly permutes the edge features between edges of the input graph. The corrupted graph serves as an negative example that does not belong to the input distribution. By optimizing the objective in Eq. (6), Anomal-E learns to separate the embeddings of original graphs and corrupted graphs in the Euclidean space. Such method uses the *contrast* between self-generated examples as training signal, thus dubbed contrastive methods.

Finally, the similarity function in Anomal-E is defined as

$$\mathcal{S}(x, y) = x^T W_s y \tag{8}$$

where W_s is a learnable matrix.

IV. Proposed Method

Here, we initially define dense vector representation within the context of intrusion detection and provide examples of implementing such representations for both numeric and categorical features. Subsequently, we discuss the necessity of in-context pre-training for dense representation models.

A. Dense Vector Representation

Recent heterogeneous NIDS datasets include high-level protocol features, which contain rich semantics that describe network dynamics. Consider the following DNS features in a ToN-IoT example [8], describing a DNS request for the IPv4 address of Google home page that is not rejected and successfully returned:

```
dns_query("google.com"),dns_qtype("A"),
dns_rcode("NOERROR"),dns_rejected(FALSE),
```

The features contain individual semantic meanings, while construct a more contextual meaning together. To capture both layers of meaning, we propose learning an *embedding* function ω, which maps each feature to a high-dimensional vector representation, and an *interaction* function φ, which models the dynamics between features.

An embedding function of feature i is a function $\omega_i : \mathcal{X}_i \hookrightarrow \mathbb{R}^{d_i}$ from the feature space $\mathcal{X}_i$ to the Euclidean space. d_i is the size of individual feature embeddings. Define $\omega(x_1, x_2, \ldots, x_r) = (\omega_1(x_1), \omega_2(x_2), \ldots, \omega_k(x_r))$. An interaction function $\varphi : \mathbb{R}^{r\times d_0} \to \mathbb{R}^{d_e}$ takes all embedded features $\{\omega_i(x_i)\}_{i\geq 1}$ as input and combines them to output a single vector of size d_e. We use d_e denote the size of the embedding because we mainly consider GNNs that operate on edge features in this paper. In ML literature, the composite function $\varphi \circ \omega$ is often parameterized jointly and referred to as a feature map. We parameterize them separately as they serve different purposes. Next, we introduce two common

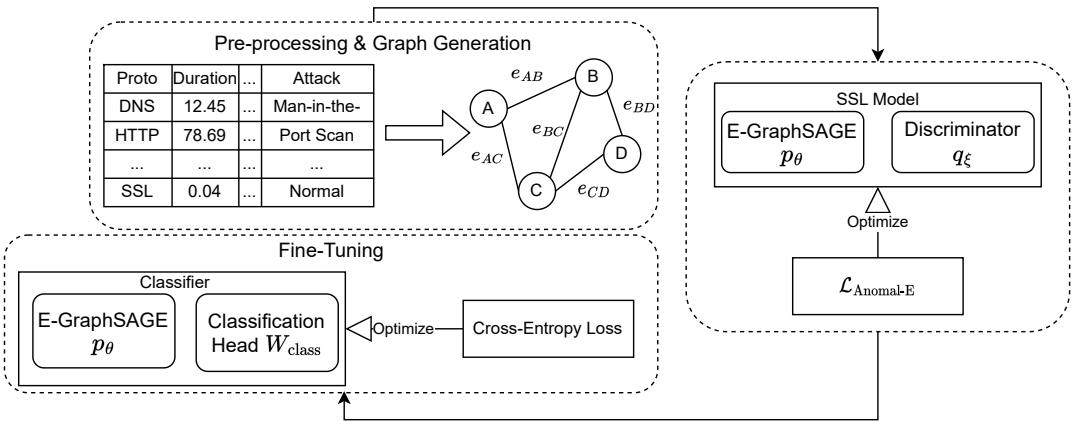

Fig. 2. Proposed training pipeline with in-context pre-training, demonstrated using Anomal-E [15] as the SSL technique. During the pre-training phase, we train an encoder p_θ and a decoder q_ξ. Subsequently, p_θ is connected to a classification head and trained using labeled data.

practices in representation learning for embedding numeric and categorical feature respectively.

1) Linear Projection on Numeric Features.: Let $x \in \mathbb{R}^{r_{num}}$ be a vector with r numeric features and $W \in \mathbb{R}^{d_e \times r_{num}}$. The linear projection $\mathbf{x} = Wx$ defines an embedding function and an interaction function. Let

$$[\omega_j(x_j)]_i = W_{ij}x_j, \tag{9}$$

we have

$$\mathbf{x}_{num} = Wx = \sum_j W_{ij}x_j = \sum_{j=1}^{r_{num}} \omega_j(x) = \varphi(\omega(x)) \tag{10}$$

We see that linear projection embeds each feature x_j as the j-th column $W_{:,j}$, scaled by x_j, and combines each embedded feature through summation. Next we show that projecting one-hot encoding with a linear transformation yields similar embedding and interaction for categorical features.

2) One-Hot Encoding for Categorical Features: Suppose we aim to encode a vector of categorical features $x = (x_1, x_2, \ldots, x_{r_{cat}}) \in \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_{r_{cat}}$, where $|\mathcal{X}_i| = c_i$ is the number of possible categories for feature i. A common method for encoding these features involves converting each feature into a one-hot vector and then transforming the vector using a linear projection:

$$v_{cat} = W\mathbf{e}, \tag{11}$$

where $e = [\mathbf{e}_1^T; \mathbf{e}_2^T; \ldots; \mathbf{e}_{r_{cat}}^T]^T$ is a concatenation of one-hot vectors $\mathbf{e}_i$ of length c_i and $W \in \mathbb{R}^{d_e \times c}$. $c = \sum_{i=1}^{r_{cat}} c_i$. We claim that the transformation defines an embedding function and an interaction function.

Write the weight matrix W as the concatenation of a column of smaller matrices $[W_1; W_2; \ldots; W_{r_{cat}}]$ where $W_i \in \mathbb{R}^{d_e \times c_i}$. By linear algebra, we have

$$\mathbf{x}_{cat} = \sum_{i=1}^{r_{cat}} W_i\mathbf{e}_i \tag{12}$$

Let $j(i)$ indicate the index of 1 in the one-hot vector $\mathbf{e}_i$.

$$\mathbf{x}_{cat} = \sum_{i=1}^{r_{cat}} W_i\mathbf{e}_i = \sum_{i=1}^{r_{cat}} (W_i)_{:,j(i)}, \tag{13}$$

where $(W_i)_{:,j(i)}$ is the $j(i)$-th column of matrix W_i. Thus, each feature i is embedded as a vector, $\omega_i(x_i) = (W_i)_{:,j(i)}$, and combined through addition.

3) Mixed Features: From the two examples above, we observe that both numeric and categorical features can be embedded as vectors through a linear projection. When the input features comprise both numeric and categorical types, we can define an interaction function that integrates both. In our approach, we generate the final vector representation of the features by concatenating the numeric and categorical embeddings obtained previously, and then pass them through a linear projection.

$$z = W[\mathbf{x}_{num}; \mathbf{x}_{cat}]. \tag{14}$$

B. In-Context Pre-training

Self-supervised pre-training has been shown effective in few-shot learning [36] and improving model generalization [37]. SSL can leverage the abundant unlabeled network traffic data for pre-training. This allows the model to learn from the inherent structure and patterns in network dynamics without the need for explicit labels. During pre-training, SSL objectives encourage the model to understand relationships and context within the raw network traffic, as opposed to the its correlation with labels. This leads to the learning of generic representations that capture essential features and information, making the model more capable of generalizing to unseen instances. We show later in experiments that this ability to learn generalizable weights is essential to applying dense representation to GNN-based NIDS.

Conventional pre-training typically involves obtaining data from a broader domain than that of the fine-tuning task. However, defining a broader domain for intrusion detection

on a specific network is challenging. Instead, we propose **in-context pre-training**, where the pre-training data is sourced from the same network as the labeled data. Fig. 2 illustrates the proposed process of training an intrusion detector with in-context pre-training. This approach results in an ML-based NIDS methodology that is less reliant on labeled data. To train an intrusion detector for a specific network from scratch, one first compiles a representative dataset consisting of raw, unlabeled traffic from the network. After labeling a manageable subset of the traffic (see Section V for details on the effort involved), one obtains a ready-to-use intrusion detector through fine-tuning.

V. Experiments

In this section, we outline our experimental configurations and discuss the experimental results. We begin by introducing the dataset on which we evaluate our methods. Then we detail the process by which we pre-process our data. Finally, we motivate our experiments by enumerating three research questions, which are answered in conjunction with an analysis of the experimental results.

A. Datasets

We choose the `ToN-IoT` [8] and `NF-UQ-NIDS-V2` [11] datasets due to their rich categorical features. `ToN-IoT` comprises 65% normal flows and 35% malicious traffic, while `NF-UQ-NIDS-V2` contains 33.12% and 66.88% normal and malicious flows, respectively. `NF-UQ-NIDS-V2`, compiled from four simulated attack scenarios, presents a more challenging scenario than `ToN-IoT` in our experiments. Due to memory constraints, we sub-sample both datasets to approximately 130,000 training and 50,000 testing flows each. We under-sample the majority classes to achieve a more balanced label distribution.

B. Hardware Configuration

We run all experiments on an RTX 3070 GPU with 8GB of memory. We use 16GB RAM and an Intel I5-12600K CPU.

C. Pre-processing & Graph Construction.

We utilize the standard features provided by [8] and [11]. Source port numbers are omitted from the feature set. Consistent with [14], we employ uniformly randomly generated IPs in the IPv4 range to represent the source of the flows. This encompasses any address from `0.0.0.0` to `255.255.255.255`. We identify the flow destinations using a combination of IP addresses and port numbers.

We pre-process the dataset leveraging various Python packages. Numeric features are standardized with `StandardScaler` in `scikit-learn` directly unless the standard deviation of a feature is twice larger than the mean, in which case we take the binary logarithm of the feature first. All unbounded categorical features are dropped, except for "DNS query" in `ToN-IoT`, which we convert to a binary feature that indicates whether the query belongs to Majestic Million top one million most visited domains[1] at the time of research. Subsequently, the remaining categorical features, along with the added binary feature, are converted to one-hot vectors as described in Section IV.

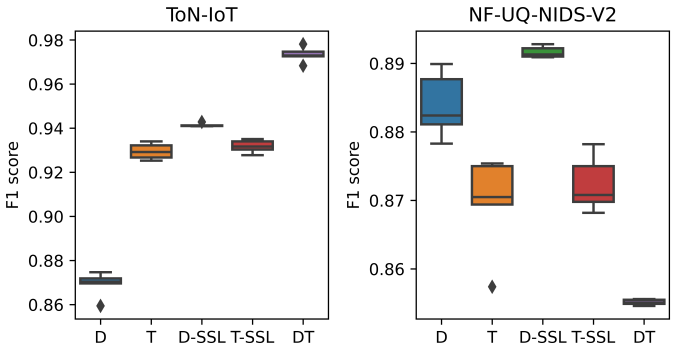

Fig. 3. Full-data setting: F1-score of dense representation models (D) and target encoding models (T) trained on all labeled data. For each model type, we experiment with a pre-trained variant (-SSL) and one without. Decision Tree (DT) is included here as a baseline.

We construct the graph by using nodes to represent hosts in the network and edges the flow between them. In line with [15], both training and testing graphs are constructed as undirected multi-graphs. The numeric features and categorical features are concatenated and stored as edge features. The node features of the graph are initialized as vectors of 1's with a length of 64.

D. Evaluation Metric

We evaluate intrusion detectors with F1-score, defined as:

$$\mathrm{F1} = 2 \times \frac{\mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \tag{15}$$

Precision represents the rate of correctly identifying true instances among alerts, while Recall indicates the percentage of true instances correctly alerted among all true instances. F1-score, as the harmonic mean of Precision and Recall, provides a summary of both abilities. We report the micro-F1-score, which aggregates F1-scores across classes using a weighted average based on the number of instances in each class, summarizing the overall performance on a specific dataset.

E. Research Questions and Experimental Settings

Our experiments aim to answer the following questions;
Q1. How does dense representation models perform compared to target encoding models?
Q2. Does in-context pre-training enhance GNN performance?
Q3. Does in-context pre-training result in a high-performing NIDS solution even when labeled training data is scarce?

The research questions are answered in two experimental settings, the *full-data setting* and the *few-shot setting*. In the full-data setting, models are trained on all the available labeled data. As we find the single F1-scores in this setting can be

[1] https://majestic.com/reports/majestic-million

close across different models, we report the F1-scores of 5 trials for each model variant. We present the trial results in a box plot. In the few-shot setting, the models are provided with at most 5,000 labeled examples, and further benchmarked with a decreasing amount of labeled data. We sub-sample the original training data with an under-sampler, which resulted in a more balanced label distribution. This is to simulate the fact that, when a small dataset is crafted manually, class balance is usually attained.

F. GNN Architecture and Hyperparameters

Across our experiments, we use E-GraphSAGE as the backbone of the GNN models with two variation factors: 1) whether the GNN embeds the input features with dense representation or target encoding, and 2) whether E-GraphSAGE is pre-trained with Anomal-E as an encoder. We use acronyms D and T to represent the GNNs without pre-training that utilizes either dense representation or target encoding embeddings respectively, and D-SSL and T-SSL to refer to their pre-trained counterparts. In addition, we include Decision Tree (DT) as a baseline. In the the full-data setting, we benchmark the five aforementioned models on both datasets. In the few-shot setting, we only include the best-performing unpre-trained GNN from the full-data setting as the representative of unpre-trained GNNs.

We train all the GNNs using the Adam optimizer with a fixed learning rate of 0.001. Each GNN architecture comprises two layers with a hidden dimension of 128, resulting in an output edge representation size of 256. We train the `sklearn` implementation of DT using the `gini` criterion and set `max_splitting_size` to 5. This set of hyperparameters are picked to maximize validation performance.

G. Results & Discussion

1) Full-Data Results Fig. 3: D-SSL is the best-performing among GNN models. SSL improves the dense representation model on both datasets. It increases the mean F1-score by 8.32% and 0.87% and decreases the the variance by 82.45% and 29.73% on `ToN-IoT` and `NF-UQ-NIDS-V2`, respectively. Meanwhile, there is no consistent improvement in the mean or variance of the F1-score for the dense representation models across the two datasets.

Dense representation models performed comparatively better on `NF-UQ-NIDS-V2` than the other models, overperforming DT by 4.27% on `NF-UQ-NIDS-V2` in mean F1-score compared to the 3.28% underperformance on `ToN-IoT`. This phenomenon can be explained by the fact that DT and target encoding models, unparameterized and less parameterized than dense representation models, has an advantage on simpler datasets like `ToN-IoT`, but struggle more on more complex datasets like `NF-UQ-NIDS-V2`.

2) Few-Shot Results: The results in Fig. 4 depict the performance in the few-shot learning setting. Across both datasets, pre-trained models demonstrate the greatest resilience to data scarcity. Remarkably, the top-performing models, T-SSL and D-SSL, achieved F1-scores of 93.22% and 84.64%, respectively. These scores correspond to 99.8% and 95.11% of the highest F1-scores attained in the full-data setting, using only 3.7% of the original training data.

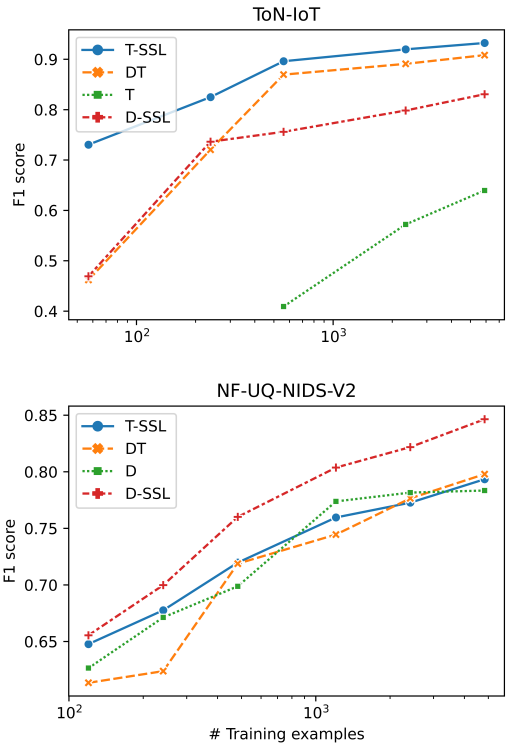

Fig. 4. Few-shot setting: Comparing pre-trained (target encoding, dense representation) and directly learned models (E-GraphSAGE, DT) on `ToN-IoT` (top) and `NF-UQ-NIDS-V2` (bottom) with limited data. Note that we only present the best-performing GNN model without pre-training in the plots.

DT ranks second in performance on both datasets when there are more than 500 labeled examples, while the second best pre-trained model ranks second on both datasets when there are less than 500 labeled examples. This indicates that pre-trained models' strength lies in learning representations that is robust against the data scarcity. Furthermore, the fact that DT outperforms certain pre-trained GNN models cast doubts on whether current synthetic datasets challenge ML-based NIDS enough.

3) Summary of Findings: Combining the results from the two experiments, we can answer our research questions;

A1. While the dense representation GNNs outperform target encoding GNNs on `NF-UQ-NIDS-V2`, target encoding proves superior on the simpler dataset without pre-training. However, with in-context pre-training, dense representation GNNs consistently outperform target encoding models.

A2. While in-context pre-training is typically avoided in other deep learning tasks due to concerns about overfitting, it consistently enhances the performance of dense representation GNNs on our considered NIDS datasets.

A3. In our experiments, pre-trained models exhibit significantly greater data efficiency compared to other methods, demonstrating their superiority by a large margin.

VI. Conclusion & Future Work

We have shown that, when pre-trained, dense representation is the superior method to embed features for GNN-based NIDS. In addition, we demonstrate that in-context pre-training yields data-efficient models that attain comparable performance with just a fraction of training data compared to previous state-of-the-art models. By proposing a few-shot learning framework, we add to the body of work that addresses the label-dependence of ML-based NIDS.

Future Work: This work only considers the most basic GNN configurations and graph SSL techniques. We leave it for future work to explore more advanced combination of GNN architectures and graph SSL techniques. Moving forward, we plan to evaluate our framework using real network traffic to gather informative insights into model performance on more realistic and complex distributions. It is also vital for future work to incorporate the directionality of network flows in future GNN-based NIDS.

References

[1] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, p. 20, Jul. 2019.

[2] H.-J. Liao *et al.*, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.

[3] R. Gassais, N. Ezzati-Jivan, J. M. Fernandez, D. Aloise, and M. R. Dagenais, "Multi-level host-based intrusion detection system for internet of things," *Journal of Cloud Computing*, vol. 9, no. 1, p. 62, Nov. 2020.

[4] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[5] Y. Mirsky *et al.*, "Kitsune: An ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.

[6] E. Bertino and N. Islam, "Botnets and internet of things security," *Computer*, vol. 50, no. 2, pp. 76–79, 2017.

[7] N. Moustafa *et al.*, "An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things," *IEEE IoT-J*, vol. 6, no. 3, pp. 4815–4830, 2019.

[8] N. Moustafa, "A new distributed architecture for evaluating ai-based security systems at the edge: Network ton-iot datasets," *Sustainable Cities and Society*, vol. 72, p. 102994, 2021.

[9] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *International Conference on Information Systems Security and Privacy*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:4707749

[10] M. Sarhan, S. Layeghy, N. Moustafa, and M. Portmann, "Netflow datasets for machine learning-based network intrusion detection systems," in *Big Data Technologies and Applications*, Z. Deze, H. Huang, R. Hou, S. Rho, and N. Chilamkurti, Eds. Cham: Springer International Publishing, 2021, p. 117–135.

[11] M. Sarhan, S. Layeghy, and M. Portmann, "Towards a standard feature set for network intrusion detection system datasets," *Mobile Networks and Applications*, vol. 27, no. 1, p. 357–370, Feb. 2022.

[12] J. L. Guerra *et al.*, "Datasets are not enough: Challenges in labeling network traffic," *Computers & Security*, vol. 120, p. 102810, 2022.

[13] T. Hasanin, T. M. Khoshgoftaar, J. L. Leevy, and R. A. Bauder, "Severely imbalanced big data challenges: investigating data sampling approaches," *Journal of Big Data*, vol. 6, no. 1, p. 107, Nov. 2019.

[14] W. W. Lo *et al.*, "E-graphsage: A graph neural network based intrusion detection system for iot," in *IEEE NOMS*, 2022, pp. 1–9.

[15] E. Caville *et al.*, "Anomal-E: A self-supervised network intrusion detection system based on graph neural networks," *Knowledge-Based Systems*, vol. 258, p. 110030, 2022.

[16] L. Chang and P. Branco, "Graph-based solutions with residuals for intrusion detection: The modified e-graphsage and e-resgat algorithms," *arXiv preprint arXiv:2111.13597*, 2021.

[17] H. Nguyen and R. Kashef, "Ts-ids: Traffic-aware self-supervised learning for iot network intrusion detection," *Knowledge-Based Systems*, vol. 279, p. 110966, 2023.

[18] X. Hu *et al.*, "Toward early and accurate network intrusion detection using graph embedding," *IEEE TIFS*, vol. 18, pp. 5817–5831, 2023.

[19] M. K. Hasan, A. K. M. A. Habib, S. Islam, N. Safie, S. N. H. S. Abdullah, and B. Pandey, "Ddos: Distributed denial of service attack in communication standard vulnerabilities in smart grid applications and cyber security with recent developments," *Energy Reports*, vol. 9, p. 1318–1326, 2023.

[20] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 2027–2051, 2016.

[21] T. Altaf, X. Wang, W. Ni, G. Yu, R. P. Liu, and R. Braun, "A new concatenated multigraph neural network for iot intrusion detection," *Internet of Things*, vol. 22, p. 100818, 2023.

[22] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning distributed representations of graphs," *CoRR*, vol. abs/1707.05005, 2017. [Online]. Available: http://arxiv.org/abs/1707.05005

[23] A. Venturi *et al.*, "ARGANIDS: A novel network intrusion detection system based on adversarially regularized graph autoencoder," in *SAC*, ser. SAC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1540–1548. [Online]. Available: https://doi.org/10.1145/3555776.3577651

[24] R. Zuech, T. M. Khoshgoftaar, and R. Wald, "Intrusion detection and big heterogeneous data: a survey," *Journal of Big Data*, vol. 2, no. 1, p. 3, Feb. 2015.

[25] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017.

[26] G. Duan *et al.*, "Application of a dynamic line graph neural network for intrusion detection with semisupervised learning," *IEEE TIFS*, vol. 18, pp. 699–714, 2023.

[27] K. Al-Naami *et al.*, "Adaptive encrypted traffic fingerprinting with bi-directional dependence," in *ACSAC*. New York, NY, USA: Association for Computing Machinery, 2016, p. 177–188. [Online]. Available: https://doi.org/10.1145/2991079.2991123

[28] P. Veličković *et al.*, "Deep graph infomax," *arXiv preprint arXiv:1809.10341*, 2018.

[29] Z. He, X. Xu, and S. Deng, "Discovering cluster-based local outliers," *Pattern Recognition Letters*, vol. 24, no. 9, p. 1641–1650, 2003.

[30] S. Pan *et al.*, "Adversarially regularized graph autoencoder for graph embedding," *arXiv preprint arXiv:1802.04407*, 2018.

[31] S. Layeghy and M. Portmann, "Explainable cross-domain evaluation of ml-based network intrusion detection systems," *Computers and Electrical Engineering*, vol. 108, p. 108692, May 2023. [Online]. Available: http://dx.doi.org/10.1016/j.compeleceng.2023.108692

[32] S. Layeghy, M. Baktashmotlagh, and M. Portmann, "DI-NIDS: Domain invariant network intrusion detection system," *Knowledge-Based Systems*, vol. 273, p. 110626, Aug. 2023. [Online]. Available: http://dx.doi.org/10.1016/j.knosys.2023.110626

[33] T. M. Booij *et al.*, "ToN-IoT: The role of heterogeneity and the need for standardization of features and attack types in iot network intrusion data sets," *IEEE IoT-J*, vol. 9, no. 1, pp. 485–496, 2022.

[34] B. Claise *et al.*, "Cisco systems netflow services export version 9," 2004.

[35] N. Koroniotis *et al.*, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-IoT dataset," *CoRR*, vol. abs/1811.00701, 2018. [Online]. Available: http://arxiv.org/abs/1811.00701

[36] Y. Chen *et al.*, "A new meta-baseline for few-shot learning," *CoRR*, vol. abs/2003.04390, 2020. [Online]. Available: https://arxiv.org/abs/2003.04390

[37] W. Wang *et al.*, "Memorization in self-supervised learning improves downstream generalization," *arXiv preprint arXiv:2401.12233*, 2024.