

Practical Evaluation of Graph Neural Networks in Network Intrusion Detection

Andrea Venturi^{1,*}, Daniele Pellegrini¹, Mauro Andreolini², Luca Ferretti²,
Mirco Marchetti¹ and Michele Colajanni³

¹Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, Modena, Italy

²Department of Physics, Computer Science and Mathematics, University of Modena and Reggio Emilia, Modena, Italy

³Department of Informatics, Science and Engineering, University of Bologna, Bologna, Italy

Abstract

The most recent proposals of Machine and Deep Learning algorithms for Network Intrusion Detection Systems (NIDS) leverage Graph Neural Networks (GNN). These techniques create a graph representation of network traffic and analyze both network topology and netflow features to produce more accurate predictions. Although prior research shows promising results, they are biased by evaluation methodologies that are incompatible with real-world online intrusion detection. We are the first to identify these issues and to evaluate the performance of a state-of-the-art GNN-NIDS under real-world constraints. The experiments demonstrate that the literature overestimates the detection performance of GNN-based NIDS. Our results analyze and discuss the trade-off between detection delay and detection performance for different types of attacks, thus paving the way for the practical deployment of GNN-based NIDS.

Keywords

ML-based NIDS, Graph Neural Network, Cybersecurity, Network Intrusion Detection

1. Introduction

As the number and complexity of cyberattacks continue to increase [1], network intrusion detection has become crucial for modern IT systems. Network Intrusion Detection Systems (NIDS) based on Machine Learning (ML) algorithms have garnered attention due to their capability of automatically learning to detect intrusions from network traffic samples [2].

This paper focuses on a novel research direction that utilizes Graph Neural Networks (GNN). In contrast to traditional Machine Learning (ML) methods that rely solely on independent network traffic samples, GNNs also take into account discernible topological patterns exhibited by modern cyberattacks [3, 4]. Previous papers proposing GNN-based NIDS show promising results and high performance. However, there is still a significant gap between scientific evaluation and practical deployment of these tools in real-world settings.

ITASEC 2023: The Italian Conference on CyberSecurity, May 03–05, 2023, Bari, Italy


*Corresponding author.

✉ andrea.venturi@unimore.it (A. Venturi); daniele.pellegrini@unimore.it (D. Pellegrini);
mauro.andreolini@unimore.it (M. Andreolini); luca.ferretti@unimore.it (L. Ferretti); mirco.marchetti@unimore.it
(M. Marchetti); michele.colajanni@unibo.it (M. Colajanni)

🆔 0000-0003-3822-968X (A. Venturi); 0009-0000-6582-7699 (D. Pellegrini); 0000-0003-3671-6927 (M. Andreolini);
0000-0001-5824-2297 (L. Ferretti); 0000-0002-7408-6906 (M. Marchetti); 0000-0002-9499-1559 (M. Colajanni)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

In this paper, we identify and discuss two common pitfalls in the evaluation methodology of related works that lead to unreliable results. Firstly, they consider randomly sampled subsets of the original traffic capture to form the train and test sets, respectively. However, such a random division may conceal some contiguous structural patterns exhibited by malicious traffic that are important for correct detection. Secondly, they only consider complete evaluation graphs built on large test sets. There are two issues with this assumption: it is inapplicable in real-world settings in which it is unknown when it is the right time to construct the graph and perform the evaluation; it considers huge graphs built over time spans of several hours, thus introducing unacceptable detection delays.

We propose a novel evaluation methodology that overcomes the identified issues by avoiding a random division between train and test sets and evaluating the GNN-based detectors by varying the time-window on which to build the evaluation graphs. In this way, we can assess whether and how much the completeness of the graph is important for the classification performed by these tools, and we can evaluate the trade-off between detection delay and detection accuracy.

We apply our methodology to a recent GNN-based NIDS at the state-of-the-art and consider two publicly available datasets largely employed in related literature. The results show different performance patterns. Most classifiers exhibit an unstable behavior when the time-windows are too short, suggesting a lower bound for the detection delay. On the other hand, a subset of them is robust to these perturbations, suggesting that, in certain situations, the GNN does not rely on structural patterns for the predictions. These findings provide useful insights for the deployment of GNN-based NIDS and pave the way for additional investigations aimed at devising more efficient solutions.

The remainder of this paper is structured as follows. Section 2 provides background on GNN and discusses related work. Section 3 focuses on the pitfalls of previous approaches and presents our novel evaluation methodology. Section 4.3 introduces the considered case study, while Section 5 shows the experimental results. Finally, Section 6 draws the conclusions and provides insights for future research directions.

2. Background and Related Work

Modern NIDS increasingly involve ML techniques for their tasks. Traditional approaches are based on network flows (or *netflows*), which consist of tabular structures in which each entry summarizes, with a predefined set of statistics and metrics (also referred to as *features*), the communication between two endpoints in a network over a certain period of time [5]. In the traditional supervised learning paradigm, each netflow is associated with a label (that is, benign or malicious), and is passed to the ML algorithm which learns to distinguish among the classes.

Traditional ML models treat netflows one at a time and independently from each other [2, 6, 7]. Although this assumption allows near real-time predictions and a responsive production of security alerts, a single netflow might be unable to express the dependencies between multiple malicious operations typical of modern multi-step attacks [8]. This limitation allows motivated threat actors to evade detection through adversarial attacks, in which minor changes in the patterns of a single malicious communication flow can fool the ML detectors [9, 10].

An alternative and recent research trend aims to address this issue by proposing the usage of

Graph Neural Networks (GNNs) for network intrusion detection [11]. These models belong to a modern family of Deep Learning (DL) techniques that do not operate over the classical Euclidean space but accept a graph as an input to the model. The idea behind this approach is that cyber-attacks present some topological patterns that can be easily detected by looking at the overall network graph structure. In other words, a graph representation can express the complex dependencies among different malicious operations so that a GNN-based NIDS can lead to robust detectors that do not rely on single netflows.

Related literature for GNN-based NIDS has demonstrated the value of these tools both in traditional and adversarial environments [3, 4, 12]. Previous proposals share the same operational workflow. First, they need to transform netflow data into a graph representation. This is a crucial factor that can affect the whole GNN model architecture. For example, the authors of [3] propose a GNN that operates directly on a *flow graph* in which each endpoint of the network represents a node of the graph and the edges are associated with the netflows. On the other hand, other papers use a *line-graph* representation in which the netflows are seen as nodes of the graph (e.g., [4, 13]), or even ad-hoc graph formats [12]. After having obtained the graph representation, it can be submitted to the GNN for the learning and evaluation phases. Multiple model choices apply at this point. The majority of proposals follow a *transductive* learning setup in which the complete graph structure is known by the model at the training time [14, 15]. Few works can also be applied to *inductive* settings in which the model does not need to retrain if the topology of the network changes [3, 13].

Despite the good performance of GNN-based NIDS, there is still a significant gap between research and practice. As we will discuss in Section 3, GNN-based NIDS are evaluated on a graph built from a random testing subset of the considered dataset, which may conceal some crucial topology information. Most importantly, this evaluation approach does not consider the temporal constraints required to build the graph itself in real-world settings. In this paper, our aim is to bridge this gap by proposing a more realistic evaluation methodology that should be followed to better assess the detection capabilities of GNN-based NIDS in real-world scenarios.

3. Evaluation methodology

GNN-based NIDS follow a two-step workflow in which the graph is usually built only before training and evaluating the network. Related works employ an evaluation methodology that mimics that used for standard ML algorithms, in which a given dataset is randomly separated into train and test sets, which are then used to create the training and test graphs, respectively. We identify two main pitfalls with this approach if applied to network intrusion detection. Firstly, the random division between training and test samples is valid only when the data samples are evaluated independently of each other, as in classical ML algorithms [16]. Instead, GNNs link correlated nodes, and hence, a random division can break the underlying topology of the network structure and lead the models to learn an incorrect graph structure. Secondly, a similar evaluation methodology builds a test graph using all the data in the test set. As dataset captures span several hours or even days [17, 18], building a graph on such a large test set would require waiting for a long time to collect all the netflows transmitted during that period. This is unfeasible in a realistic scenario as it would introduce excessive detection delays [19].

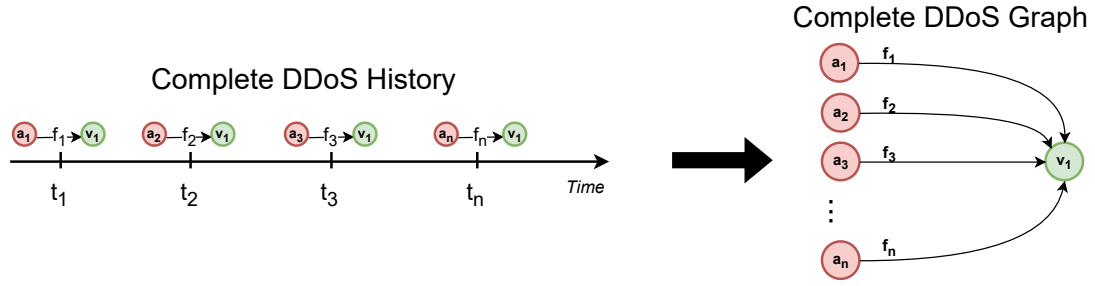


Figure 1: DDoS complete graph.

Let us suppose that we build a GNN-based NIDS trained to detect DDoS attacks. These attacks typically involve multiple attacking hosts a sending traffic to a single victim node v , resulting in a graph structure as shown in Figure 1. Here, the n attacker's nodes a_i overflow the victim node v_1 with multiple netflows f_i , forming the graph on the right. Without loss of generality, we suppose that the i nodes send their traffic in order at times t_i . If considered singularly, the netflows f_i of a DDoS attack can show characteristics resembling normal traffic, and can be misclassified by detectors analyzing them independently (e.g., standard ML-based NIDS) [20]. The strength of applying a GNN for network intrusion detection lies in the fact that these models do not rely exclusively on netflow features (even if they still play an important role) and can easily learn to identify such a peculiar structure shown by the DDoS graph, raising alarms for the involved flows. However, the standard evaluation scenario considered by previous works uses the entire history of events to build the testing graph employed to assess the performance of the GNN.

Let us now consider Figure 2. Here, we suppose that the construction of the testing graph occurs periodically, considering only the netflows coming in a time-window of length τ . This reflects a more realistic evaluation scenario in which it is unknown when it is the right time to build the evaluation graph and perform the prediction. For instance, if we consider the case of Figure 2, where the test set capture starts at time t_0 , the first evaluation graph will comprise the netflows received between t_0 and $t_e = t_0 + \tau < t_2$, when the second netflow of the DDoS attack is transmitted. The result would not contain the complete DDoS attack scenario and could complicate the work of the GNN as it has been trained on standard DDoS attacks. Similar conclusions can be drawn for the other time periods shown in the figure.

We can conclude that previous works employing complete graphs limit their evaluation to the best-case settings, which are biased by the presence of the whole history of the attack. To overcome this pitfall, we propose a more reliable evaluation scenario that can be used to assess GNN-based NIDS that need to be deployed in practice. We avoid a random split between training and testing sets to let the model learn the actual graph as reported in the dataset capture. Then, to reflect a more practical deployment scenario, we temporally divide the test set into different time-constant periods of length τ . Hence, we build one evaluation graph for each of the obtained subsets using only the netflows for the corresponding time-period, as shown in Figure 2. In this way, we simulate a GNN-based NIDS that is invoked periodically at a constant time-rate of τ , and is provided with a more realistic evaluation graph that reflects the practical

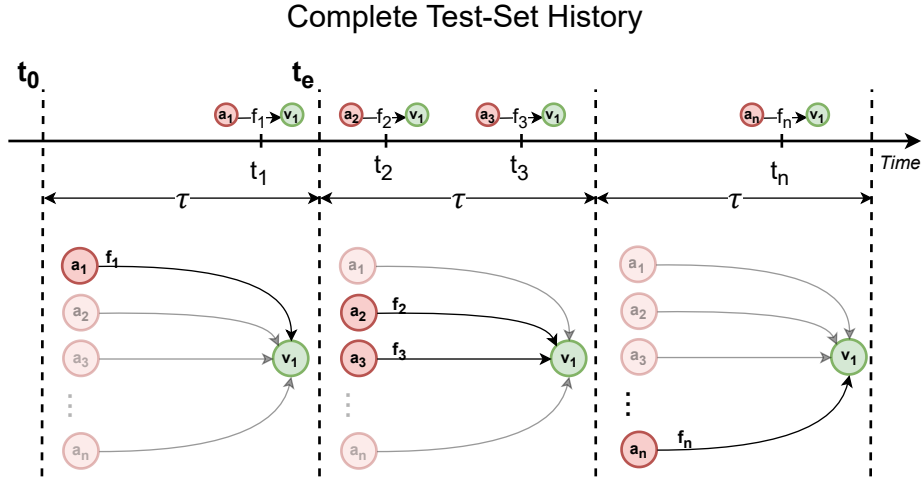


Figure 2: Examples of evaluation graphs generated from time windows of τ dimensions.

scenarios.

4. Case Study

We now present a case study in which we apply our evaluation methodology to assess the performance of E-GraphSAGE [3], a recently proposed GNN-based NIDS achieving state-of-the-art performance. We introduce the model, describe the considered datasets and detail the experimental scenarios.

4.1. E-GraphSAGE

E-GraphSAGE [3] is a variation of the widely used GraphSAGE model [21] that has been adapted to perform netflow classification. It is considered one of the pioneering GNN models that operate in an inductive manner in this field and has gained popularity in related literature, with several studies incorporating it as a key component [22, 23, 4]. E-GraphSAGE operates on flow-graphs in which each node corresponds to an endpoint of the network (that is, IP address-port pair) as extracted from the netflows. Through a series of graph convolutional layers and aggregator functions, it learns to produce low-dimensional embedding representations for each netflow, taking into account not only the features of the analyzed netflow but also information from its neighborhood. For additional details, we refer the reader to the original work [3].

For the scope of this paper, it is worth noting that the experimental campaign presented by the authors of E-GraphSAGE is subject to the pitfalls described in the previous section: the train and test sets are randomly separated, and it is only evaluated by considering the complete test graph. Our case study extends the experiments performed in the original work to assess the capabilities of E-GraphSAGE in more realistic evaluation scenarios. We base our tests on the implementation of E-GraphSAGE made available by the same authors in [24]. However, we avoid the standardization phase of the IP addresses and Port numbers intended by the authors,

as it would completely disrupt the network topology [13]. Moreover, unlike the provided multi-class version, as the best practices for network intrusion detection suggest [25, 26], we build multiple binary detectors that are specifically tailored to detect each attack in our datasets.

4.2. Datasets

Our experimental campaign is based on two widely used datasets for network intrusion detection: CTU-13 [17] and ToN-IoT [18].

CTU-13 includes thirteen real-world network traffic captures that mix normal traffic with traffic from different botnet variants. The data is already labeled in a netflow format, which is suitable for our study. The botnets in the dataset exhibit unique structures, such as DDoS and Port Scans, with each capture lasting from a few hours to entire days, making it a good benchmark for GNN-based NIDS. We preprocess the CTU-13 dataset by removing non-TCP traffic and outliers, as well as explicit IP addresses and port numbers to avoid learning shortcut patterns [16]. Instead, we use categorical features, such as "IP Address Type" and "Port Type," which indicate the network type of the original IP address and the IANA group of the original port number. The final set of features includes duration, source/destination/total packets, source/destination/total bytes, bytes/s, packets/s, ratio of src/dst bytes, protocol, port type, IP address type, flow direction, source/destination state, and type of service. The malicious samples from each botnet variant are separated into dedicated collections, and those with insufficient samples or too short of a capture time are excluded from the experiments as they would prevent a thorough analysis. The final set of botnet variants considered in our experiments is reported in the first column of Table 1a.

ToN-IoT [18] represents another valid benchmark, considered by the same authors of E-GraphSAGE in the original paper [3]. Our focus in this work is on the network component of the dataset, which comprises traces from various cyberattacks and normal traffic from a medium-sized IoT network. We follow the same preprocessing steps as with the CTU-13 dataset and divide each attack into separate collections. The final set of features includes duration, source/destination/total packets, source/destination/total bytes, missed bytes, protocol, DNS info, SSL info, and HTTP info. In this case, we exclude the Man-In-The-Middle class from the evaluation due to the limited number of netflows provided in the dataset. The final set of attacks considered in our experiments is reported in the first column of Table 1b.

4.3. Evaluation methodology implementation

In this section, we describe the implementation of our evaluation methodology outlined in Section 3 and used in our case study. As mentioned before, we create a separate binary E-GraphSAGE model for each attack in our datasets. To do this, we extract a train-test set pair for each attack. However, unlike previous works, we do not randomly divide the samples into training and test sets.

For the CTU-13 dataset, we divide the malicious samples obtained after the preprocessing phase into groups based on their capturing date. We then attach the benign flows belonging to the same time-period to each group. We select the group from the second day of each capture to form the test set, while the remaining groups are combined to form the training set. To mimic

the typical data imbalance in real-world environments, we enforce a 20 : 1 benign:malicious ratio in both the training and test sets [16]. The details are shown in Table 1a.

For the ToN-IoT dataset, each malicious capture is divided into two groups, namely training and testing groups, based on the capture date, as most attacks last exactly two days. For attacks that last less than a day (Ransomware, Injection, and XSS), the training group includes the first 75% of the samples, and the last 25% forms the testing group. We then merge the benign flows into the groups. However, the benign samples in this dataset are far fewer than the malicious ones and are spread across several days. Thus, we form the final training set by merging the training group with the benign samples from the day with the highest number of benign flows (175 300 benign flows). Similarly, for the test set, we select the benign samples from the day with the second highest benign traffic (65 280 flows). In both sets, we maintain the 20 : 1 benign:malicious ratio, as for CTU-13. In this way, we maximize the number of malicious samples in both training and test sets, while simulating real-world captures for the evaluation of our classifiers. The details are shown in Table 1a.

We then train the E-GraphSAGE instances on each attack. For this phase, we use the entire training sets obtained before, as we realistically assume that they involve traffic samples that have already been captured and labeled in the considered network environment. We repeat the learning phase for 200 epochs for all the attacks, as done in the original E-GraphSAGE [3] paper. As discussed in Section 3, we avoid considering the whole test sets of the attacks to assess the performance of the classifiers. Instead, we build smaller evaluation graphs by splitting the test sets into time-constant periods of length τ . This approach simulates the periodical invocation of the classifiers, and by varying the τ value we can study the performance evolution of the model in different time-periods. We select a range of τ values that can represent realistic evaluation periods: 1s, 10s, 30s, 1m, 10m, 30m, 1h, 3h. Given a τ , we randomly select a starting time t in the test set, and include in the evaluation graph just the netflows that present a Timestamp between t and $t + \tau$. We consider valid only those time-periods with at least one malicious and one benign flow.

5. Results

In this Section, we present our experimental campaign. We evaluate the E-GraphSAGE classifiers in two scenarios. The first scenario, which we refer to as the *baseline* evaluation, follows the procedure used in related literature and discussed in Section 3 (see Figure 1). Here, we construct a complete graph representation from each entire test set and proceed evaluating each classifier on it. This approach allows us to validate the E-GraphSAGE algorithm in the common evaluation setting. On the other hand, the second scenario, which we refer to as the *Reduced Time Window* scenario, considers the proposed novel evaluation methodology. Here, we divide the test sets into shorter time periods of varying lengths, as described in the previous section. For each time period length τ , we extract 100 different graphs by varying the starting time t . We then average the results across the 100 graphs for each τ .

We evaluate the classifiers considering the standard metrics used in network intrusion detection, namely *F1-Score*, *Recall (Detection Rate)*, and *Precision*. Considering an attack sample

Table 1

Distribution of CTU-13 and ToN-IoT datasets. **Time** refers to the total capture time in days (or hours); **Training** and **Testing Sets** report the groups (i.e., days (hours) of capture) used to form the training and testing sets, respectively (between parenthesis we report the number of malicious samples in each set).

Botnet	Time	Training Set (#Mal)	Testing Set (#Mal)
Neris	3 days	Days 1, 3 (61 638)	Day 2 (10 300)
Rbot	5 days	Days 1, 3, 4, 5 (22 960)	Day 2 (5 003)
Virut	2 days	Day 1 (15 959)	Day 2 (16 686)
Murlo	2 days	Day 1 (2 809)	Day 2 (2 999)

Attack	Time	Training Set (#Mal train)	Testing Set (#Mal testing)
DDoS	2 days	Day 1 (8765)	Day 2 (3264)
DoS	2 days	Day 1 (8765)	Day 2 (3264)
Bkdr	2 days	Day 1 (8765)	Day 2 (3264)
Scan	2 days	Day 1 (8765)	Day 2 (3264)
Rans	1 day	Hours 1-18 (8765)	Hours 18-24 (3264)
Psw	2 days	Day 1 (8765)	Day 2 (3264)
Inj	1 day	Hours 1-18 (8765)	Hours 18-24 (3264)
XSS	1 day	Hours 1-18 (8765)	Hours 18-24 (3264)

as positive, the three metrics can be computed as follows:

$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN} \quad F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (1)$$

in which TP, FP, TN and FN refer to the True Positive, False Positive, True Negative and False Negative predictions returned by the classifier. These metrics range between 0 and 1, with higher values indicating better performance.

5.1. Baseline performance

We now present the results of the E-GraphSAGE classifiers' performance in the baseline scenario for the CTU-13 and ToN-IoT datasets in Tables 2a and 2b, respectively. All metrics show high scores, indicating the quality of the detectors. In the CTU-13 dataset (Table 2a), all classifiers achieve F1-scores above 0.95 for all botnets, with an average of 0.972. The precision scores are almost perfect, with an average of 0.991, evidencing a low number of false positives, while the recall lies at 0.955, which is a solid indicator of good detection rates. The best performance is provided by the Murlo classifier, with an average F1-Score of 0.997. Similarly, for the ToN-IoT dataset, the classifiers achieved an average F1-score of 0.956, which is consistent with the state-of-the-art results [3, 4]. We remark that for both the DoS and Backdoor attacks, the classifiers achieve a perfect score for all the metrics. Instead, the E-GraphSAGE algorithm struggles to identify Ransomware attacks, as the F1-Score does not surpass 0.844. However, this behavior is expected since Ransomware attacks usually involve few hosts and do not exhibit unique topological structures that can be extracted by the GNN.

Overall, these results demonstrate the efficacy of the E-GraphSAGE classifiers in the standard evaluation scenario where the complete graph is considered during the testing phase. However, as explained in Section 3, this approach may overestimate performance in real-world settings where complete graphs are not available.

Table 2

Baseline performance of the E-GraphSAGE classifiers over the two considered datasets

(a) Baseline performance on the CTU-13 dataset.				(b) Baseline performance on the ToN-IoT dataset.			
Botnet	F1-score	Recall	Precision	Attack	F1-score	Recall	Precision
Neris	0.985	0.990	0.980	DDoS	0.942	0.992	0.897
Rbot	0.954	0.913	1.000	DoS	1.000	1.000	1.000
Virut	0.954	0.921	0.988	Bkdr	1.000	1.000	1.000
Murlo	0.997	0.999	0.996	Scan	0.953	0.962	0.944
average	0.972	0.955	0.991	Rans	0.844	0.812	0.879
(std. dev.)	(0.021)	(0.045)	(0.008)	Psw	0.968	0.940	0.999
				Inj	0.958	0.993	0.926
				XSS	0.986	1.000	0.972
				average	0.956	0.962	0.952
				(std. dev.)	(0.050)	(0.064)	(0.048)

5.2. Reduced Time Window results

We now consider the *Reduced Time Window* scenario in which we apply the proposed evaluation methodology. The results for the two datasets are provided in Table 3 and 4, respectively. For each attack, we show three average scores in the first three subrows, computed based on 100 evaluation graphs with different τ values indicated in the columns. The cells in each table are colored from red to white, with darker red indicating a larger loss compared to the baseline value. Additionally, we present the average number of malicious and benign samples in the evaluation graphs for each τ in the last subrow.

After examining Tables 3 and 4, we observe three distinct performance patterns. For several classifiers, we notice a correlation between lower performance scores and shorter τ values. As discussed in Section 3, this intuitive trend arises as a result of the GNN struggling to detect topological patterns of malicious communication in evaluation graphs created from shorter τ values. For the CTU-13 dataset (Table 3), this tendency is particularly evident for the Neris detector, in which we register a significant drop of over 30% in the F1-score when τ is set to 1 second (from 0.985 to 0.671), and a monotone raise when τ grows. Another example is the Murlo classifier, which maintains low scores for τ values below 10 minutes. Other meaningful cases for the ToN-IoT dataset (Table 4) are represented by the classifiers for the DDoS, Ransomware, Injection and XSS attacks which show steep declines in F1-scores going from 0.942, 0.844, 0.958 and 0.986 to 0.683, 0.593, 0.500 and 0.844, respectively, when τ is set to 1 second.

We also observe a subset of classifiers that exhibit minimal sensitivity to the reduction of time intervals used to construct the evaluation graphs. For instance, the Virut botnet detector of the CTU-13 dataset passes from an F1-score of 0.954 to 0.943 when τ is set to 1 second (loss of just 1%), and it converges to the baseline within the 10 seconds time-window. A similar trend

is offered by the Password-Cracking classifier (F1-score from 0.968 to 0.924), while the scores for the DoS and Backdoor classifiers are not even impacted. This unexpected behavior might imply that the E-GraphSAGE detectors for these attack categories may not rely on malicious topological patterns for detection, but rather on the netflow features. These results also suggest that, for certain attacks, it might be convenient to employ more traditional ML techniques that offer comparable performance, while being far less complex and better explainable.

While most of the classifiers fall into the first two categories, there are two exceptions worth noting. The classifier for Rbot in the CTU-13 dataset rapidly recovers to scores close to the baseline already from the smallest time-windows. However, it shows a significant performance decrease when considering evaluation graphs built on captures lasting 30 minutes and 1 hour (the baseline F1-score of 0.954 drops to 0.794 and 0.721, respectively). Similarly, the Scan detector for the ToN-IoT dataset resembles the behavior of the second group appearing robust even when τ is set to 1 second, but it exhibits a severe Precision drop at the 3 hours interval (from 0.944 to 0.777). Although these classifiers perform well for small evaluation graphs, they represent meaningful cases in which the topological features can weaken detection performance for large evaluation graphs.

In summary, these results endorse our novel evaluation methodology, confirming that previous approaches using complete test graphs may lead to upper-bound performance and an overestimated sense of security. Furthermore, they assume great significance for the practical deployment of these tools in real-world settings, offering valuable insights into the optimal time-window length combinations for achieving the highest detection rates. Finally, they also suggest that for certain cases, employing conventional ML algorithms that do not take into account topological structures may be more appropriate for network intrusion detection.

Table 3
Performance of E-GraphSAGE on the botnets of the CTU dataset.

Botnet	Metric	Baseline	τ							
			1s	10s	30s	1m	10m	30m	1h	3h
Neris	<i>F1-Score</i>	0.985	0.671	0.725	0.759	0.781	0.808	0.897	0.920	0.979
	<i>Recall</i>	0.990	0.673	0.765	0.748	0.771	0.784	0.861	0.888	0.987
	<i>Precision</i>	0.980	0.713	0.753	0.803	0.812	0.855	0.946	0.960	0.971
	Mal:Ben	-	3 : 12	11 : 136	21 : 407	62 : 814	468 : 8 231	1 471 : 25 703	2 575 : 51 500	10 300 : 206 000
Rbot	<i>F1-Score</i>	0.954	0.885	0.930	0.940	0.935	0.904	0.794	0.721	0.954
	<i>Recall</i>	0.913	0.868	0.885	0.898	0.889	0.859	0.748	0.748	0.913
	<i>Precision</i>	1.000	0.920	1.000	0.999	0.998	0.959	0.849	0.849	0.999
	Mal:Ben	-	1 : 1	5 : 11	15 : 131	31 : 172	301 : 928	722 : 3 542	1 209 : 6 483	5 002 : 23 076
Virut	<i>F1-Score</i>	0.954	0.943	0.955	0.948	0.949	0.957	0.956	0.957	0.952
	<i>Recall</i>	0.921	0.956	0.958	0.937	0.934	0.926	0.927	0.930	0.939
	<i>Precision</i>	0.988	0.960	0.968	0.972	0.977	0.990	0.988	0.987	0.967
	Mal:Ben	-	1 : 3	4 : 27	14 : 76	29 : 153	289 : 1 440	851 : 4 289	1 639 : 8 752	10 300 : 206 000
Murlo	<i>F1-Score</i>	0.997	0.851	0.875	0.714	0.706	0.835	0.907	0.959	0.969
	<i>Recall</i>	0.999	0.890	0.881	0.738	0.749	0.820	0.886	0.942	0.947
	<i>Precision</i>	0.996	0.856	0.872	0.710	0.692	0.879	0.953	0.981	0.997
	Mal:Ben	-	1 : 2	4 : 7	4 : 18	4 : 36	39 : 359	141 : 1 028	279 : 1 976	838 : 5 905

Table 4

Performance of E-GraphSAGE on the attacks of the ToN-IoT dataset.

Attack	Metric	Baseline	τ							
			1s	10s	30s	1m	10m	30m	1h	3h
DDoS	<i>F1-Score</i>	0.942	0.683	0.730	0.734	0.795	0.975	0.969	0.967	0.955
	<i>Recall</i>	0.992	0.680	0.715	0.713	0.876	0.984	0.984	0.987	0.982
	<i>Precision</i>	0.897	0.690	0.756	0.794	0.814	0.969	0.957	0.950	0.935
	Mal:Ben	-	1 : 1	1 : 7	4 : 21	7 : 44	73 : 444	213 : 1342	417 : 2705	1049 : 8198
DoS	<i>F1-Score</i>	1.000	1.000	1.000	0.993	1.000	1.000	1.000	1.000	1.000
	<i>Recall</i>	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	<i>Precision</i>	1.000	1.000	1.000	0.991	1.000	1.000	1.000	1.000	1.000
	Mal:Ben	-	1 : 1	2 : 6	8 : 20	18 : 40	186 : 420	525 : 1276	944 : 2596	944 : 2596
Bkdr	<i>F1-Score</i>	1.000	1.000	1.000	0.998	0.998	0.998	0.988	0.998	0.992
	<i>Recall</i>	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	<i>Precision</i>	1.000	1.000	1.000	0.995	0.997	0.997	0.983	0.996	0.989
	Mal:Ben	-	1 : 1	1 : 7	2 : 21	5 : 44	53 : 445	161 : 1351	312 : 2705	824 : 8193
Scan	<i>F1-Score</i>	0.953	0.960	0.989	0.988	0.985	0.986	0.962	0.926	0.869
	<i>Recall</i>	0.962	0.955	0.983	0.984	0.983	0.986	0.981	0.977	0.975
	<i>Precision</i>	0.944	0.970	0.998	0.994	0.988	0.987	0.950	0.895	0.777
	Mal:Ben	-	1 : 1	7 : 7	19 : 22	34 : 45	203 : 463	361 : 1373	544 : 2748	1061 : 8199
Rans	<i>F1-Score</i>	0.844	0.593	0.631	0.726	0.806	0.826	0.878	0.858	0.859
	<i>Recall</i>	0.812	0.586	0.617	0.719	0.801	0.747	0.802	0.797	0.797
	<i>Precision</i>	0.879	0.610	0.660	0.756	0.863	0.999	0.999	0.971	0.958
	Mal:Ben	-	13 : 1	44 : 7	87 : 23	145 : 45	1117 : 446	1895 : 1334	2342 : 2696	2984 : 8243
Psw	<i>F1-Score</i>	0.968	0.924	0.972	0.966	0.980	0.991	0.984	0.980	0.989
	<i>Recall</i>	0.940	0.915	0.953	0.948	0.967	0.990	0.997	0.999	0.999
	<i>Precision</i>	0.999	0.940	1.000	0.995	0.995	0.991	0.972	0.964	0.980
	Mal:Ben	-	1 : 1	3 : 7	11 : 22	23 : 45	215 : 463	570 : 1372	1009 : 2703	2119 : 8132
Inj	<i>F1-Score</i>	0.958	0.500	0.617	0.727	0.643	0.902	0.944	0.947	0.952
	<i>Recall</i>	0.993	0.477	0.563	0.663	0.557	0.864	0.933	0.944	0.960
	<i>Precision</i>	0.926	0.560	0.758	0.908	0.887	0.952	0.956	0.953	0.946
	Mal:Ben	-	1 : 1	7 : 7	19 : 22	36 : 43	340 : 447	904 : 1341	1558 : 2706	2225 : 8196
XSS	<i>F1-Score</i>	0.986	0.844	0.944	0.967	0.984	0.997	0.996	0.997	0.995
	<i>Recall</i>	1.000	0.834	0.934	0.965	0.976	0.999	0.999	1.000	0.999
	<i>Precision</i>	0.972	0.879	0.979	0.979	0.994	0.994	0.994	0.995	0.990
	Mal:Ben	-	1 : 1	5 : 7	16 : 22	32 : 44	284 : 461	703 : 1390	1139 : 2777	2141 : 8261

6. Conclusions

The use of Graph Neural Networks (GNN) in detecting network intrusions has garnered the attention of the research community due to its ability to generate strong detections by leveraging topological structures and netflow features. While previous studies have demonstrated promising outcomes, there remains a significant disparity between research and practical applications. In this paper, we identify and discuss two common pitfalls within the evaluation methodology considered by related work that make it unsuitable for real-world settings: the random division between train and test sets may hide crucial topological patterns, while the use of complete evaluation graphs hinders online detection. We propose a novel evaluation method that avoids these issues and apply it to a recent state-of-the-art GNN-based NIDS. The results of our experimental campaign offer essential aspects to consider for the deployment of these tools in practice. First, it highlights how most of the detectors work effectively only when the time-windows considered to build the evaluation graphs are sufficiently large, leading to

delayed detection. Second, it suggests that in some cases, the detectors ignore the topological patterns for classification, and their inclusion might even result in decreased performance. Therefore, in such scenarios, traditional ML algorithms relying exclusively on netflow features may be a more suitable and less expensive option. Our results show which attack categories can actually benefit from a GNN-based approach and analyze the trade-off between detection delay and detection performance. Future work will extend our case study to other state-of-the-art GNN-based NIDS proposed in the literature and aim to validate our findings using explainability methods and tools.

Acknowledgments

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

References

- [1] J. Jang-Jaccard, S. Nepal, A survey of emerging threats in cybersecurity, *Journal of Computer and System Sciences* 80 (2014) 973–993.
- [2] R. Sommer, V. Paxson, Outside the closed world: On using machine learning for network intrusion detection, in: *2010 IEEE symposium on security and privacy*, IEEE, 2010, pp. 305–316.
- [3] W. W. Lo, S. Layeghy, M. Sarhan, M. Gallagher, M. Portmann, E-graphsage: A graph neural network based intrusion detection system for iot, in: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2022, pp. 1–9.
- [4] L. Chang, P. Branco, Graph-based solutions with residuals for intrusion detection: The modified e-graphsage and e-resgat algorithms, *arXiv preprint arXiv:2111.13597* (2021).
- [5] M. F. Umer, M. Sher, Y. Bi, Flow-based intrusion detection: Techniques and challenges, *Computers & Security* 70 (2017) 238–254.
- [6] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, F. Ahmad, Network intrusion detection system: A systematic study of machine learning and deep learning approaches, *Transactions on Emerging Telecommunications Technologies* 32 (2021) e4150.
- [7] G. Apruzzese, M. Colajanni, M. Marchetti, Evaluating the effectiveness of adversarial attacks against botnet detectors, in: *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, IEEE, 2019, pp. 1–8.
- [8] F. Manganiello, M. Marchetti, M. Colajanni, Multistep attack detection and alert correlation in intrusion detection systems, in: *Information Security and Assurance: International Conference, ISA 2011, Brno, Czech Republic, August 15-17, 2011. Proceedings*, Springer, 2011, pp. 101–110.
- [9] B. Biggio, F. Roli, Wild patterns: Ten years after the rise of adversarial machine learning, in: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2154–2156.
- [10] G. Apruzzese, M. Andreolini, M. Marchetti, A. Venturi, M. Colajanni, Deep reinforcement

- adversarial learning against botnet evasion attacks, *IEEE Transactions on Network and Service Management* 17 (2020) 1975–1987.
- [11] W. Jiang, Graph-based deep learning for communication networks: A survey, *Computer Communications* (2021).
 - [12] D. Pujol-Perich, J. Suarez-Varela, A. Cabellos-Aparicio, P. Barlet-Ros, Unveiling the potential of graph neural networks for robust intrusion detection, *ACM SIGMETRICS Performance Evaluation Review* 49 (2022) 111–117.
 - [13] H. Zhu, J. Lu, Graph-based intrusion detection system using general behavior learning, in: *GLOBECOM 2022-2022 IEEE Global Communications Conference*, IEEE, 2022, pp. 2621–2626.
 - [14] J. Zhou, Z. Xu, A. M. Rush, M. Yu, Automating botnet detection with graph neural networks, *arXiv preprint arXiv:2003.06344* (2020).
 - [15] Q. Xiao, J. Liu, Q. Wang, Z. Jiang, X. Wang, Y. Yao, Towards network anomaly detection using graph embedding, in: *Computational Science–ICCS 2020: 20th International Conference*, Amsterdam, The Netherlands, June 3–5, 2020, *Proceedings, Part IV* 20, Springer, 2020, pp. 156–169.
 - [16] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, K. Rieck, Dos and don'ts of machine learning in computer security, in: *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 3971–3988.
 - [17] S. Garcia, M. Grill, J. Stiborek, A. Zunino, An empirical comparison of botnet detection methods, *Computers & Security* 45 (2014) 100–123.
 - [18] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, A. Anwar, Ton_iot telemetry dataset: A new generation dataset of iot and iiot for data-driven intrusion detection systems, *IEEE Access* 8 (2020) 165130–165150.
 - [19] A. Venturi, C. Zanasi, On the feasibility of adversarial machine learning in malware and network intrusion detection, in: *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*, IEEE, 2021, pp. 1–8.
 - [20] F. De Gaspari, D. Hitaj, G. Pagnotta, L. De Carli, L. V. Mancini, The naked sun: Malicious cooperation between benign-looking processes, in: *Applied Cryptography and Network Security: 18th International Conference, ACNS 2020, Rome, Italy, October 19–22, 2020, Proceedings, Part II* 18, Springer, 2020, pp. 254–274.
 - [21] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, *Advances in neural information processing systems* 30 (2017).
 - [22] J. Lan, J. Z. Lu, G. G. Wan, Y. Y. Wang, C. Y. Huang, S. B. Zhang, Y. Y. Huang, J. N. Ma, E-minbatch graphsage: An industrial internet attack detection model, *Security and Communication Networks* 2022 (2022).
 - [23] E. Caville, W. W. Lo, S. Layeghy, M. Portmann, Anomal-e: A self-supervised network intrusion detection system based on graph neural networks, *arXiv preprint arXiv:2207.06819* (2022).
 - [24] waimorris, E-graphsage, <https://github.com/waimorris/E-GraphSAGE>, 2022.
 - [25] D. Wu, B. Fang, J. Wang, Q. Liu, X. Cui, Evading machine learning botnet detection models via deep reinforcement learning, in: *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, IEEE, 2019, pp. 1–6.
 - [26] B. Biggio, I. Corona, Z.-M. He, P. P. Chan, G. Giacinto, D. S. Yeung, F. Roli, One-and-a-half-

class multiple classifier systems for secure learning against evasion attacks at test time, in: Multiple Classifier Systems: 12th International Workshop, MCS 2015, Günzburg, Germany, June 29-July 1, 2015, Proceedings 12, Springer, 2015, pp. 168–180.