Interaction Context-Aware Network Behavior Anomaly Detection for Discovering Unknown Attacks

Zhi-Quan Qin, Hong-Zuo Xu, Xing-Kong Ma, and Yong-Jun Wang

College of Computer, National University of Defense Technology, Changsha 410073, China

Correspondence should be addressed to Yong-Jun Wang; wangyongjun@nudt.edu.cn

Received 18 June 2021; Revised 30 December 2021; Accepted 27 January 2022; Published 11 April 2022

Academic Editor: Azeez M

Copyright © 2022 Zhi-Quan Qin et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Network behavior anomaly detection is an effective approach to discover unknown attacks, where generating high-efficacy network behavior representation is one of the most crucial parts. Nowadays, complicated network environments and advancing attack techniques make it more challenging. Existing methods cannot yield satisfied representations that express the semantics of network behaviors comprehensively. To tackle this problem, we propose XNBAD, a novel unsupervised network behavior anomaly detection framework, in this work. It integrates the timely high-order host states under the dynamic interaction context with the conversation patterns between hosts for behavior representation. High-order states can better summarize latent interaction patterns, but they are hard to be obtained directly. Therefore, XNBAD utilizes a graph neural network (GNN) to automatically generate high-order features from series of extracted base ones. We evaluated the detection performance of XNBAD in a publicly available benchmark dataset ISCX-2012. To report detailed and precise experimental results, we carefully refined the dataset before evaluation. The results show that XNBAD discovered various attack behaviors more effectively, and it significantly outperformed the existing representative methods by at least 3.8% relative improvement in terms of the overall weighted AUC.

1. Introduction

Network security of companies and organizations has long been threatened by various cyberattacks. Attacks targeting these institutions aim at stealing core secrets, leaking sensitive information, and tampering important data, which cause great damage and pose potential serious threats. Cyberattacks became more severe during COVID-19 pandemic because many companies and organizations use telecommuting, and their networks are more open than ever. According to a report from Zscaler, 1,200 of the COVID-19 related attacks in phishing, malicious websites, and malware targeting remote users were observed and blocked in January 2020, and the number grew to 380,000 in April, showing an increase of about 30,000% [1]. Another report released by World Health Organization (WHO) in April 2020 indicated the number of cyberattacks directed at the organization is five times more than that in the same period of 2019, and 450 active WHO e-mail addresses and passwords were leaked along with thousands of researcher information in a week [2].

Intrusion detection is one of the key steps to protect network security, which has been continuously studied. The anomaly detection-based technique builds a normal profile from large amounts of data and then detects anomalies (attacks) based on their degrees of deviation from the normal profile. Thus, the detection system is empowered to discover unknown attacks. With evolution and innovation of techniques, novel attacks are constantly emerging. Thus, anomaly detection attracts increasing attention from both the computer security community and the machine learning community.

In recent years, more and more researchers have utilized machine learning to develop anomaly detection methods by harnessing its powerful ability of learning complex data patterns and distributions. Nevertheless, it is still a non-trivial task of simply applying machine learning to detect malicious network traffic due to the following challenges. (1) Labeling Difficulty. Most machine learning approaches are in supervised manner which requires abundant labeled training data to guide their model learning process. However,
labeling network data is laborious and expensive. Huge volume of network traffic can be collected in a day but attack traffic rarely appears, let alone to be found. Besides, the manual labeling process can only annotate known attacks from overwhelming network traffic, which means supervised models may fail to retrieve unknown attacks. (2) Encryption Barrier. The payload content is helpful for discriminating attacks because attackers usually exploit the vulnerabilities of network applications through the payloads. With the popularization of encryption protocols such as Transport Layer Security (TLS) protocol, attackers can leverage encryption protocols to deliver payloads, which makes it very hard, if not impossible, to obtain this information. (3) Attack Diversification and Sophistication. Cyberattacks are becoming more and more diversified and sophisticated. Companies and organizations are facing various attacks such as phishing, scanning, brute force, denial of service (DoS), and so on. These attacks may have completely different abnormal characteristics on different aspects, so it is very challenging to perform effective detection on a single aspect. Beyond that, skilled attackers tend to customize attacks for their targets, such as advanced persistent threats (APTs), which make the attacks subtle, stealthy, and hard to discover.

There are two kinds of detection mechanism, i.e., content anomaly detection and behavior anomaly detection. Content detection methods rely on the payloads of application layers to discover anomalies. However, it is difficult to apply these methods directly on the encrypted traffic. In contrast, behavior detection methods do not have this trouble because they are not dependent on payload contents to discover anomalies. An effective data representation is the foundation of a successful behavior detection method. It must be able to describe network activities from a sufficiently high-level perspective, consider various aspects related to attacks as many as possible, and reflect network activities in time.

Existing behavior detection methods can be categorized into four types, i.e., packet-based methods, flow-based methods, extended flow-based methods, and static embedding-based methods, from the perspective of data representation. Packet-based methods [3, 4] utilize L2–L4 header information (e.g., IP addresses and packet sizes) to represent network behaviors and infer anomaly degrees of packets. Though the header information is easy to retrieve, it is too simple to describe high-level attack activities. The information acquired from packet headers is highly untrustworthy in the dynamic and high-speed contemporary network environments [5]. Packet-level detection in massive network traffic generally also induces huge computation costs. Flow-based methods [6, 7] extract features from the packets in flows to represent the network behaviors. Flow features are able to capture the conversation patterns between hosts, which are relatively powerful to discriminate abnormal behaviors and normal ones. However, considering conversation patterns alone is not sufficient. Skilled attackers could mimic normal conversation patterns to evade the detection mechanism. Recently, researchers also explored extracting features from other aspects. Extended flow-based methods not only utilize flow features but also consider extra information such as payload contents [5, 8] and the relationships of multiple flows [9, 10] to improve detection performance. Nevertheless, these existing works fail to handle encrypted traffic or require labeled data for training. Inspired by the field of natural language processing, several methods leverage embedding learning in feature extraction [11, 12]. These methods also target at flows but regard network data (e.g., IP addresses) as words. They obtain numerical representations of words by embedding models like Word2Vec [13], which automatically learns relationships among categorical data in flows. These methods mainly suffer from two problems. First, these methods learn static embeddings. Therefore, in highly dynamic environments, neither can they deal with unseen words nor can they describe changing characteristics of words over time [14]. Second, these methods either ignore numerical features or treat them as categorical features, which means important information in numerical features is likely to be omitted. In a word, existing methods fail to obtain satisfied behavior representations for anomaly detection.

To address the above challenges and overcome the problem of the existing methods, an interaction context-aware network behavior anomaly detection framework XNBAD is proposed in this paper. The key idea of XNBAD is to integrate the conversation patterns between hosts and the timely interaction states of hosts to comprehensively characterize network behaviors for anomaly detection. The host interaction state, summarizing how a host interacts with the remaining hosts in network, refers to one-to-many behavioral characteristics while the conversation pattern refers to one-on-one behavioral characteristics. By taking host interaction states into consideration, network activities are viewed from a higher level, the semantics of network behaviors are expressed more comprehensively, and at the same time, the difficulty of evasion is increased since attackers need to imitate a wider behavioral pattern. Concretely, XNBAD extracts flow features to represent the conversation patterns and generates host features from a short-term interaction context to represent the timely interaction states. The interaction context is modeled as a graph structure, and series of base host features are first extracted from the graph, and then a graph neural network (GNN) is utilized to generate enhanced host features, where latent interaction patterns are mined through the high-order structural information of the graph. To get rid of labeling difficulty, XNBAD trains the GNN and builds the normal profile on an attack-free training set. To evaluate detection performance of XNBAD, we conducted extensive experiments on the publicly available benchmark dataset ISCX-2012 which contains multiple attack scenarios and various malicious behaviors. We summarize the contributions of this work as follows:

(i) A novel unsupervised network behavior anomaly detection framework XNBAD is proposed for detecting unknown attacks. It estimates anomaly scores of network behaviors under the dynamic host interaction context. Besides, to express the semantics of network behaviors comprehensively, it considers both of the conversation patterns between hosts and the interaction states of hosts.

(ii) A novel method that generates timely high-order host interaction features from network traffic is
introduced. It allows to extract series of base features from various aspects (e.g., domain knowledge and graph analysis). Further, a GNN-based feature enhancement is proposed to generate high-order features from the base ones.

(iii) A further exploration on the benchmark dataset ISCX-2012 has been made, which is helpful to obtain more accurate results and more detailed analysis in experiments. In the exploration, mislabeled data are corrected and fine-grained malicious behavior labels are added according to the detailed materials provided with the dataset.

(iv) Extensive experiments on ISCX-2012 are conducted to evaluate the detection performance of XNBAD. The results show that XNBAD effectively discovered more types and numbers of malicious behaviors and significantly outperformed the best competitor with the relative improvement of 3.8% in terms of the overall weighted AUC (a more rigorous metric than the general AUC) at 1% level. The effectiveness of the proposed GNN-based feature enhancement was validated. Besides, the effect of different hyper-parameter configurations was investigated.

To our best knowledge, XNBAD is the first unsupervised learning framework utilizing GNN to detect malicious network behaviors in traffic, and this is the first work to carefully study the ISCX-2012 dataset and analyze the detection ability on different malicious behavior types.

The rest of the paper is organized as follows. Section 2 reviews the related works. Section 3 describes the overview and details of our proposed framework XNBAD. Section 4 introduces the basic information of the ISCX-2012 dataset and our refinement on it. Then, Section 5 explains the experiments conducted on ISCX-2012, and Section 6 presents and analyzes the experiment results. Finally, Section 7 draws the conclusions and discusses future work.

2. Related Work

Several previous studies related to our work are reviewed in this section. As mentioned in Section 1, the network behavior anomaly detection methods can be divided into four categories according to their data representations, i.e., packet-based methods, flow-based methods, extended flow-based methods, and static embedding-based methods. They are briefly summarized in Table 1 and discussed in detail as follows.

2.1. Packet-Based Methods. This category of methods extracts features from L2–L4 packet headers and estimates the anomaly degree for each packet. PHAD [3] is one of the earliest packet-based methods, which uses 33 features from packet headers, such as IP addresses and IP packet size, for detection. It learns the normal value ranges on the features, estimates the scores for anomalous features in a packet, and sums the feature scores up as the packet’s anomaly score for detection. Recently, several deep learning empowered methods are proposed. Kitsune [4] is a packet-based method using an ensemble of autoencoders (AEs) [15] as anomaly detector. It applies damped incremental statistics on packet sizes and counts to represent the behaviors of the source host, channel (source and destination host pair), and socket of a given packet. There are totally 115 packet features extracted including 1D features like the weight, mean, and standard deviation of the source host and 2D features like the approximate covariance between the source and destination hosts. Then, it groups these features and feeds them forward to the unsupervised learning detector for anomaly detection. HELAD [16] is a novel packet-based framework integrating multiple deep learning models. Similar to Kitsune, it extracts packet features by improved damped incremental statistics and then uses a combination of AE and long short-term memory (LSTM) [17] to obtain anomaly scores. Adding LSTM is aimed at considering the relationship between consecutive attacks, but it requires labeled data for training. Though using deep learning techniques can improve the learning capability for complex behavior patterns, the packet-based methods might perform poorly on detecting sophisticated attacks since packet features show a microview of network traffic which cannot reflect the high-level network behaviors. Besides, the header information could become untrustworthy in contemporary dynamic network environments [5]. Moreover, this category of methods will suffer from heavy computation when dealing with massive network traffic.

2.2. Flow-Based Methods. This category of methods first groups packets into flows by the same flow key, e.g., the common used network five-tuple (IP source, Port source, IP destination, Port destination, Protocol), and then extracts flow features based on the header information of packets in flows to estimate the flows’ anomaly degrees. In decades of research, various flow features have been proposed, such as statistical features, sequential features, and neural features generated by deep learning models. CICFlowMeter [18, 19] is a popular open-source flow collection and statistical feature extraction tool. It extracts totally 83 statistical features for bidirectional flows, such as the statistics of the packet sizes, the packet interarrival times, and the TCP flags. ZED-IDS [20] trains a deep AE with the flow features produced by CICFlowMeter to discover potential zero-day attacks, DoS attacks in particular. Similarly, AE-D3F [6] trains an AE with 27 of the statistical flow features proposed in [21] for DDoS attack detection. In [22], the sequential features that array the payload sizes of the first 50 packets of a TLS session are used to classify whether a TLS encrypted session is malicious or not. In [23], similar sequential features are used to infer HTTP protocol semantics and improve malware detection on encrypted traffic. As deep learning gradually gains attention in the field of cyber security, researchers start training end-to-end neural network models, where the automatically learned neural features are immediately generated from sequential features, for relevant classification and detection tasks. FS-Net [24] trains an encoder-decoder recurrent neural model to learn
representative neural features from the sequences of packet sizes within flows for encrypted traffic classification. LUCID [25] organizes the sequence of packet header information as an image and trains a convolutional neural network (CNN) on the images to learn the flow representations for DDoS attack detection. STIDM [26] combines a 1D-CNN and a length-sensitive LSTM to generate discriminative flow representations from the sequences of packet sizes and time intervals for intrusion detection. Flow-based methods focus on the conversations between hosts, which establish a moderate view on network traffic. Although flow features do not involve the contents of conversations, they can provide discriminative behavioral information of flows for detection. However, this category of methods ignores the information outside the conversation which might be helpful for discriminating malicious network behaviors, and furthermore, attackers could imitate normal conversation patterns to bypass these detectors. Besides, the end-to-end deep learning models need to be trained with labeled data, which does not meet the challenge of labeling difficulty.

### 2.3. Extended Flow-Based Methods

This category of methods extends flow-based detection methods by leveraging extra information. Some methods combine the features extracted from the contents of the application payloads with flow features for detection. TR-IDS [8] concatenates the payload features generated by deep learning models and a series of statistical flow features as the input of a supervised random forest anomaly detector. In [5], a set of novel features is proposed to detect intrusion in IoT networks, where the contents of the application layer protocols utilized in IoT environments are extracted as the service-based features, such as the DNS query and the HTTP method. Besides, a bunch of statistical features in a window of 100 records are also used to enhance the flow features.

| Table 1: Summarization of network behavior anomaly detection methods from the aspect of data representation (the existing methods are divided into four categories; their representative references are, respectively, listed in column category). |
|---------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|
| Category                        | Outline of data representation                  | Pros                                           | Cons                                           |
|---------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|
| Packet-based methods [3, 4, 16]  | (1) Directly extract feature for each packet from its L2–L4 header information. | (1) Header information is easy to retrieve.    | (1) Poor performance on sophisticated attacks since only a microview of network activities is provided. |
|                                 |                                                  | (2) Free from encrypted payload.               | (2) Header information could be untrustworthy in modern network environments. |
|                                 |                                                  |                                                 | (3) Heavy computational overhead on massive traffic. |
| Flow-based methods [6, 18, 20, 25, 26] | (1) Group packets into flows first.               | (1) Relatively good performance since a moderate view on network activities is provided. | (1) Have false negatives since it only considers the conversation patterns inside a flow. |
|                                 | (2) Extract features for each flow based on series of header information of its packets. | (2) Free from encrypted payload.               | (2) Possibility of evasion by mimic attacks targeting conversation patterns. |
|                                 |                                                  |                                                 | (3) Labeled malicious samples are required for training (only for those supervised learning methods). |
| Extended flow-based methods [8–10] | (1) On the basis of flow features, incorporate extra information from different aspects such as payload content and multi-flow relationship. | (1) Good performance on various attacks since different aspects of network activities are considered. | (1) Existing methods fail to utilize extra information without supervised learning. |
|                                 |                                                  | (2) Free from encrypted payload (only for those not using payload content). | (2) Poor performance on encrypted traffic (only for those using payload content). |
|                                 |                                                  | (3) Efficient since there is no need to discriminate every packet. | (3) Possible inevitable delay when processing extra information. |
| Static embedding-based methods [11, 12, 14] | (1) Treat network data like words in natural language. | (1) Exploit the co-occurrence relationship of discrete values in network data. | (1) Poor generalization ability in open and dynamic network environments. |
|                                 | (2) Use word embedding methods like Word2Vec to generate embeddings for the “words.” | (2) Free from encrypted payload.               | (2) Fail to treat numerical feature values properly. |
|                                 | (3) Embedding representations or their combinations are used as features. | (3) Very efficient since the embeddings are stored and used immediately once they are learned. | |
Note that those content-related features could help for detection, but they may become useless or be hard (if not impossible) to obtain in encrypted traffic. Other methods attempt to mine the relations of multiple flows to improve detection performance. AMF-LSTM [9] inputs the statistical feature vectors of a flow and its previous flows into an attention-based LSTM model to find their correlations and generate the final representation of the current flow for anomaly detection. Similar methods are used in [27, 28]. STDeepGraph [10] explores spatial (structural) similarities and temporal dependencies of flows. It first associates the flows in a time interval by a graph structure called temporal communication graph and then applies the shortest-path graph kernel, graph signal processing, and 1D-CNN to generate the spatial feature vectors for each flow. Finally, similar to AMF-LSTM, it adopts a modified LSTM model to learn the temporal dependencies and final representations of flows for detection, where the spatial feature vectors and the statistical flow feature vectors are combined as input. Unfortunately, all the above methods of this category are trained in a supervised manner, which does not meet the labeling difficulty challenge. Among them, the most related and inspiring work to ours is STDeepGraph since it extracts features from graphs as well. However, the differences between STDeepGraph and our proposed XNBAD are significant. First of all, XNBAD is designed to train without labeled data, free from the labeling difficulty. Second, XNBAD improves detection performance by supplying the host states under the current network context, in which expert knowledge can be utilized, rather than the structural similarities of flows the supervised neural models learn. Finally, the form of the graphs XNBAD builds is quite different from that of STDeepGraph.

2.4. Static Embedding-Based Methods. This category of methods, inspired by the field of natural language processing, regards the data in network (e.g., IP addresses, port numbers, and so on) as the words (tokens) in natural language and leverages the embedding methods like Word2Vec to generate the numerical representations of network data for detection. The general idea of the embedding methods is to learn the numerical vectors (embeddings) by modeling the co-occurrence likelihood of a target word and its context words. Therefore, the learned embeddings of the frequently co-occurring words are more similar than the embeddings of the infrequently ones. In [14], IP2Vec, a Word2Vec variant adapted to flow records, is proposed to learn the embeddings of IP addresses as well as port numbers and protocols. In [11], SkipGram, one of the Word2Vec models, is used to generate the embeddings of both the categorical and numerical values in flow records, which means numbers are also viewed as categorical values. Then, for each record, the concatenation of the embeddings, replacing the original flow feature vector, is input to supervised detection models. In [12], a network connection is represented by two words, system and connection type where system is the IP source of the connection and connection type is the combination of IP destination, Port destination, and Protocol. Then, the word embeddings are provided by a variant of SkipGram, and the cosine similarity between the embeddings is calculated as the normal degree of the connection. It should be noted that the numerical flow features are not well considered in the above methods, either ignored or treated as categorical. RL-IDS [29] takes care of this issue. It only learns embeddings on categorical features with an unsupervised method FVRL and then integrates the embeddings and numerical features with a supervised method NNRL. These embedding methods mine co-occurrence relationships inside network traffic for data representation, which much differs from the other three kinds of methods. However, they perform poor in terms of generalization in highly dynamic network environments because the embedding learning methods are static, which are twofold [14]. First, they cannot learn the embeddings for words outside training sets, such as new addresses or port numbers. Second, the learned embeddings are fixed, and they only represent the historical states of words but cannot vary with the present situation of network.

3. Methodology

In this section, we present our proposed framework XNBAD in detail. The objective of XNBAD is generating an effective data representation of network behaviors to detect anomalous (malicious) flows. To this end, it describes the network behavior of a flow under the current host interaction context, which includes the timely host interaction states and the conversation between the hosts. Compared to the mainstream methods which only consider the conversations, XNBAD considers both inside and outside of a flow and therefore is able to express the semantics of network behaviors more comprehensively, thereby improving the performance of network behavior anomaly detection.

The overview of XNBAD framework is shown in Figure 1. The host interaction of the entire network is changing over time, so the interaction states of the same host could be very different on two periods. Therefore, XNBAD uses a short-term window of flows to model the current interaction context. The context is modeled as a graph G (called host interaction graph, HIG) where nodes stand for hosts and edges stand for flows. For a flow record r initiated from host s to host d, the interaction states of the hosts s and d under the current context are represented by two host feature vectors $z_s$ and $z_d$, respectively, which are generated based on G, and the conversation is represented by a flow feature vector $q$ which can be directly extracted from the network traffic. Then, the network behavior under the context is represented by a vector $v_r$ as

$$v_r = \psi(z_s, z_d, q),$$

where $\psi$ denotes a function that integrates $z_s$, $z_d$, and $q$. XNBAD finally feeds the behavior vector $v_r$ to an anomaly detector to estimate the anomaly degree of the flow record r, i.e., the anomaly score $\mathcal{A}(r)$.

Accordingly, the detection phase of XNBAD can be divided into three stages, flow collection and feature extraction, host interaction feature generation, and network detection.
behavior anomaly detection. Inside the second stage, three steps are included, i.e., graph construction, base feature extraction, and GNN-based feature enhancement. In the following sections, we will detail the above three stages of XNBAD as well as the training procedures involved.

3.1. Flow Collection and Feature Extraction. A flow is generally defined as a series of packets with the same network five-tuple (IP$_{source}$, Port$_{source}$, IP$_{destination}$, Port$_{destination}$, Protocol). Upon that, flows are categorized into unidirectional flows and bidirectional flows. In a unidirectional flow, all the packets move from the source to the destination while in a bidirectional flow, the packets move between these two ends. XNBAD collects records on bidirectional flows since they contain more sufficient information of the conversations between the hosts than unidirectional flows. A flow record consists of metadata such as the five-tuple, the start and stop timestamps of the flow, and a flow feature vector representing the host conversation within the flow. Formally, a flow record $r$ is represented as (2) in this work.

$$r = (s, d, \mu, q), \quad (2)$$

where $s$ and $d$, respectively, stand for the source and destination IP addresses, $\mu$ stands for the other metadata of the flow, and $q \in \mathbb{R}^{D_{flow}}$ is the $D_{flow}$-dimensional flow feature vector. The general process of this stage is shown in Algorithm 1. During this stage, a dictionary $\mathcal{D}$ is used to buffer living (unfinished) flows, and a queue $\mathcal{Q}$ is used to collect issued (finished) flow records that wait for the next stage. For each incoming packet, a bidirectional flow key is calculated first to find the corresponding living flow, and then the packet information is used to initiate/update the metadata and flow feature vector of the flow. Besides, to avoid waiting for the records of the long-inactive and long-living flows, the timeout mechanism is applied, in which the inactive timeout is five seconds and the flow-life timeout is two minutes. Therefore, once a flow is timed out or a TCP flow is connection-closed, the corresponding flow record is issued and collected.

There are various flow features proposed in the existing works, such as the statistical features [5, 18, 21], the sequential features [22, 23], and the neural features that are automatically learned by deep learning models [24, 25]. Though not accessing the conversation contents, the flow features are able to capture the conversation patterns from the aspects of the number, size, time interval, and direction of the packets within flows. In this work, we choose CICFlowMeter [18, 19] to extract flow features since it is a flexible open-source tool and it can provide abundant and discriminative features for the tasks like anomaly detection and attack classification. Concretely, it extracts over 80 statistical flow features. These features can be split into three groups, i.e., the forward, backward, and bidirectional features. In each directional group, several statistics of the packets are calculated as features such as the minimum, maximum, mean, and standard deviation of the sizes, the interarrival times, the active times, and so on. More details about CICFlowMeter can be found in [18, 19]. It should be noted that not only the above flow features but also other flow features that represent the conversations can be used in XNBAD because how to choose the proper flow features is really dependent on the exact network environment.

Even though the flow features are able to provide discriminative information for anomaly detection, they cannot fully express the semantics of network behaviors of the flows because they only describe a series of actions between the subjects (i.e., source hosts) who initiate the actions and the objects (i.e., destination hosts) who react but neglect the states of them. Therefore, the extra information related to the source and destination hosts is needed to compensate the shortage of the flow features on representing network behaviors.
Global:
Source of incoming packets
Dict buffering living flows
(mutex) Queue collecting issued flows Q
(1) while packet p is coming from:
(2) k← flowkey of p; t← current time
(3) if k not in mutex: # flow a first packet
(4) initialize a flow record r with p
(5) mutex[k]→r
(6) elsif mutex[k] is timeout at t:
(7) r←mutex.pop(k); K.append d (r)
(8) if r is flow-life timeout # break the flow
(9) initialize a flow record r’ with the same source and destination addresses as r
(10) mutex[k]→r’
(11) else: # update flow features incrementally
(12) update K [k] with p and t
(13) if mutex [k] is closed: #append d (mutex.pop(k));

Algorithm 1: Flow collection and feature extraction.

3.2. Host Interaction Feature Generation. A host’s interaction with the other hosts in network reflects its network state in some aspect and is as important as the conversation between the hosts for representing network behaviors in anomaly detection. For example, if a host suddenly sends short messages to many other hosts, then it could be scanning the network so the behaviors of the related flows should be regarded as anomalous. Another example is that if an unauthorized host, which has never interacted with the database server before, downloads files from the server now, then this download behavior probably indicates a data exfiltration by a compromised host and it should be regarded as anomalous. Therefore, XNBAD extracts features of the source and destination hosts from the interaction context of the entire network and then integrates them with the flow features to represent the network behaviors.

XNBAD captures a series of snapshots on the changing interaction context. For each snapshot, it builds a HIG and then generates host feature vectors to catch the timely interaction states of hosts. Since effective host features are hard to design and extract directly, XNBAD extracts several types of base features first and then applies a multi-layer graph neural network (GNN) for feature enhancement. The general process on a snapshot is shown in Algorithm 2. A snapshot corresponds to a two-minute tumbling window. XNBAD first constructs the HIG G and the derived HIG G_d based on a window of flow records in Q. Then, it produces several types of base feature vectors based on the graphs for all the hosts in G. Finally, it feeds the concatenated base host feature vectors forward to the GNN layer by layer to generate the final host feature vectors. The above three steps of this stage are detailed as follows.

3.2.1. Graph Construction. To preserve the interaction information as much as possible, XNBAD constructs a HIG by directly joining the flow records that share the same hosts (IP addresses). A flow’s initial direction is the direction of its first packet, i.e., from the source host to the destination host. Since a pair of hosts can produce multiple flows in different initial directions, the HIG is viewed as a directed edge-attributed multi-graph where parallel edges could exist between a pair of nodes and edges have their own attributes. Formally, given a list of flow records collected in the t-th window R\( (t) \) = \{r\( i \)\_\( t \)\_\( i \) = 1, 2, …, |R\( t \)|\}. Then, for each record r\( i \)\_\( t \), we create its corresponding directed edge from node s\( i \)\_\( t \) to node d\( i \)\_\( t \) as

\[ E\( i \) = (s\( i \)\_\( t \), d\( i \)\_\( t \); μ\( i \)\_\( t \), q\( i \)\_\( t \))\_{i=1}^{n} \]

where the elements after the semicolon are viewed as the edge attributes.

We also give each IP a node index by the indexing mapping \( Φ\_\( t \): V\( (t) \)→\{1, 2, …, |V\( t \)|\}. Then, for each record r\( i \)\_\( t \), we create its corresponding directed edge from node s\( i \)\_\( t \) to node d\( i \)\_\( t \) as

\[ E\( i \) = (s\( i \)\_\( t \), d\( i \)\_\( t \); μ\( i \)\_\( t \), q\( i \)\_\( t \))\_{i=1}^{n} \]

where the elements after the semicolon are viewed as the edge attributes.

Furthermore, we derive a weighted directed simple graph G\_\( t \)\_\( d \) from the multi-graph G\( (t) \) as

\[ G\_d = (V\_d, E\_d, W\_d) \]

\[ E\_d = \{(s, d) | (s, d; μ, q) \in E\_d\} \]

\[ W\_d = (w\_d)_{i,j}^{V\_d} \]

where V\_d = V\( (t) \), E\_d = E\( (t) \) is the derived edge set, and W\_d is the weighted adjacency matrix that represents the interaction strengths between nodes. We generate E\_d by removing the redundant parallel edges from G\( (t) \) which ensures G\_d is a simple graph. We measure the interaction strength between
Input:
A window of flow records \( R \) in the collecting queue \( Q \);
A bunch of different types of base host feature extracting functions \( f_{\text{base}} = (f_{b_1}, f_{b_2}, \ldots, f_{b_m}) \);
\( L \)-layer graph neural network \( g_0 = (g_0^1, g_0^2, \ldots, g_0^L) \) for host feature enhancement

Output:
Enhanced host feature vectors \( Z \) corresponding to \( R \);

# construct graphs (Section 3.2.1)
(1) construct \( G \) and \( G_d \) from \( R \);  
# equations (3) (6)
(2) foreach \( f_b \) in \( f_{\text{base}} \):
(3) produce type-\( b \)-th feature vectors for all nodes in \( G \), i.e., \( X_b \leftarrow f_b(G) \);
# feature concatenation
(4) \( X \leftarrow [X_1 | X_2 | \ldots | X_m] \);
# enhance base features (Section 3.2.3)
(5) \( Z^0 \leftarrow X \);
(6) foreach \( \ell = 1 \) to \( L \):
(7) \( Z^\ell \leftarrow \Theta_\ell (Z^{\ell-1}, G_d) \);
# GNN forward
(8) \( Z \leftarrow Z^L \);
(9) return \( Z \)

Algorithm 2: Host interaction feature generation.

the \( i \)-th and \( j \)-th nodes by the number of flow records between them as \( u_{ij}^{(t)} = \left\lfloor (s, d, \mu, q) \in R^{(t)} \mid q^{(t)}(s) = i \right. \right. \}
\left. \left. q^{(t)}(d) = j \right\rfloor \right\rfloor .

For convenience and clarity, we omit the time notation \( (t) \) sometimes hereafter since we mainly discuss the processes in a single time window.

3.2.2. Base Feature Extraction. In this step, XNBAD extracts series of base features for the hosts in the current window. These features could be obtained from different aspects such as the domain knowledge and expert experiences about this network, the knowledge and experiences transferred from other fields, and so on. Thus, this step can be summarized as a bunch of different types of functions \( f_{\text{base}} = (f_{b_1}, f_{b_2}, \ldots, f_{b_m}) \) that generate the base host interaction feature vectors from the current host interaction graph \( G = (V, E) \) (or its derived graph) as

\[
X = f_{\text{base}}(G) = [f_{b_1}(G), f_{b_2}(G), \ldots, f_{b_m}(G)],
\]

where \( f_{b}(G) \in \mathbb{R}^{[V] \times D_{\text{base}}} \) is the type-\( b \)-th base feature matrix (for \( i = 1, 2, \ldots, m \)) and the output \( X \in \mathbb{R}^{[V] \times D_{\text{base}}} \), with \( D_{\text{base}} = \sum_{i=1}^m D_{b_i} \), is the final base feature matrix. All the matrices store the features of host \( v \in V \) at the \( \phi(v) \)-th row so \( v \)'s final base feature vector \( x_v \) is \( (X \phi(v), \vdots) \). During this step, different types of base feature matrices can be generated independently, and then they are column-concatenated as the final base feature matrix.

Specifically, we extract three major types of base interaction features in this work. They are the domain-based features directly extracted from \( G \), the traditional graph-based features extracted from the derived graph \( G_d \), and the static embedding-based features generated and induced by using a node embedding learning method on the derived graph. It also should be noted that the base host interaction features are not limited to these three types and any other useful feature reflecting the host interaction states can be utilized in XNBAD.

(1) Domain-Based Features. Domain knowledge and experiences from experts like network administrators, security analysts, and so on are very helpful to depict the host interaction states. The following describes series of domain-based features extracted in this work and they cover several aspects of host interaction: host classes, involved services, and used ports.

Host Class. The hosts are classified into four classes U, S, O, and X. U, S, and O stand for the user hosts, servers, and the other hosts in the internal network, respectively, while X stands for the external hosts. Then a 4-dimensional one-hot feature vector is generated for each host according to its host class.

Interactive Distributions on Different Host Classes and Directions. This series of features reflects the state of a host interacting with different host classes. Figure 2 gives an example. For each target node that we extract features for, we first count the numbers of its forward and backward edges connected with the four different host classes. These counting features can be organized by a matrix \( M_{D_H} \in \mathbb{N}^{2 \times 4} \) where the first row’s four elements record the numbers of the edges going to the hosts of the class \( U, S, O, \) and \( X \), respectively, while the second row records those coming from the four classes. Thus, the first row of \( M_{D_H} \) is viewed as the interactive host class distribution on the forward direction while the second row is that on the backward direction. Then, the marginal interactive host class distribution is calculated by applying the elementwise sum on the two row.
Similarly, the interactive direction distributions on each host class and the corresponding marginal distribution can be obtained along with different columns. Besides, for each distribution, the entropy is calculated to measure its diversity. Formally, given a distribution of \( m \) values \( \mathcal{D} = [N_1, N_2, \ldots, N_m] \), we first calculate its normalized distribution \( \hat{\mathcal{D}} = [p_1, p_2, \ldots, p_m] \), where \( p_i = \frac{N_i}{\sum_{i=1}^{m} N_i} \) and then calculate its entropy \( H(\hat{\mathcal{D}}) = -\sum_{i=1}^{m} p_i \ln(p_i) \). Note that the above distributions and entropies are based on the edges (flows). Furthermore, we also generate the similar distributions and entropies based on the bytes and packets for the target node.

**Interactive Distributions on Different Service Classes and Directions.** This series of features reflects the state of a host accessing and offering different services. Similar to the above distributions, for each target node, we count the numbers of its forward and backward edges with different services. We roughly classify the flows into 11 service classes such as HTTP, DNS, Mail, and so on, according to their protocols and ports. Similarly, these features are organized as a matrix \( M_{DS} \in \mathbb{N}^{2 \times 11} \) and then the distributions and entropies can be obtained.

**Interactive Distributions on Different Port Classes and Directions.** This series of features reflects the port usage state of a host. At first, the ports are simply classified into two classes, privileged ports (< 1024) and unprivileged ports (\( \geq 1024 \)), and then for each host, its flows are classified into a 2-by-2 matrix \( M_{DP} \) based on the initial directions and port classes, which is similar to the above procedures. However, instead of counting the flow numbers directly, we record a port distribution which contains the distinct port number with their counts in each matrix element. Then, we calculate its sum, its entropy, and the number of unique ports for each port distribution. Therefore, the resulting features can be organized as a 3-order tensor \( M_{DP} \in \mathbb{R}^{2 \times 2 \times 3} \) where the first order is indexed by the direction, the second is indexed by the port class, and the third is indexed by the result type (the sum, the entropy, or the unique number).

**Graph-Based Features.** In this work, we extract several node features based on traditional graph analysis since they have been shown effective for node anomaly detection in many applications [30–33]. These features are extracted from the derived graph \( G_d \), as shown in Figure 3, and they are described as follows.

**Degree Centralities.** Centralities are a kind of global features measuring the importance of a node in a graph. Here, we compute the in-degree centrality, out-degree centrality, and degree centrality.

**Egonet Features.** An egonet is an induced subgraph of neighbors centered at a given node within a specific radius, and egonet features describe the local structure of a node. Here, several 1-hop egonet features are extracted such as the number of nodes, the number of edges, the number of the 2-step-away node pairs, the sum of the weights, the egonet density, and so on.

**Embedding-Based Features.** This type of features includes the node embeddings learned from the historical normal traffic (i.e., the training set) and the features induced from the node embeddings under the current context.

**Node Embeddings.** The low-dimensional node embeddings learned from a graph are useful for various graph tasks such as node classification and link prediction [34]. Here, we employ the famous embedding model Node2Vec [35] to learn the embeddings for hosts. The learned embeddings reflect the historical interaction relationships between hosts. Generally, if two hosts are used to interacting frequently, they have similar embeddings. Since Node2Vec is a static learning method which cannot learn the embeddings of nodes outside its training graph, we make a closed-world workaround (detailed in Section 3.4) where only the users and servers in the internal network have their own unique embeddings while the other internal hosts and the external hosts share two different embeddings respectively, as illustrated in Figure 4.

**Induced Features from Node Embeddings.** For each coming window, several features are induced from the learned embeddings on the derived graph \( G_d \) to reflect the current relationships between a target host and the hosts it is interacting with. Figure 4 gives an example. This is inspired by the works...
In which the embeddings similarities are used to detect lateral movements. For a target node, we first fetch its 1-hop egonet and the embeddings of its egonet nodes and then we calculate the four features like [36]. Let $\delta_{\text{in}}$ be the minimum and $\delta_{\text{out}}$ be the mean of the similarities between the target node and its egonet nodes, the mean of the similarities for all node pair in the egonet, and the minimum of the similarities between the egonet nodes and their embedding centroid. It is assumed that if a host interacts with unfamiliar hosts, such as lateral movements occurring, these features could be influenced.

Once a target host’s three types of features are obtained, we concatenate them together as its base interaction feature vector. Generally, the above base features have limited representation ability. They only capture low-order interaction patterns since they mainly focus on the interactions within the target node’s 1-hop neighborhood. On the other hand, directly extracting features that capture high-order interaction patterns is difficult and expensive, especially by feature engineering, since it usually encounters intractable problems such as data sparsity and curse of dimensionality.

### 3.2.3. GNN-Based Feature Enhancement

In order to effectively capture high-order interaction patterns, XNBA further utilizes a GNN to enhance the host features. GNNs are a kind of powerful neural networks for representation learning on graphs. They are able to consider both of the structural information of graphs and the attributes of nodes to automatically learn the low-dimensional node representations, and recently they have achieved surprising performances in many graph-related tasks. Formally, this step can be summarized as a parameterized function $g$ like

![Figure 3: Example of extracting graph-based features for a target node. The left shows a derived graph $G_d$ with $V_d = \{a, \ldots, j\}$. The target node is $a$ and its 1-hop egonet $\mathcal{E}(a) = (V_{\epsilon a}, E_{\epsilon a}, W_{\epsilon a})$ is marked by the rounded rectangle. The edge weights of $\mathcal{E}(a)$ are annotated near their corresponding edges. The right is the calculation of the degree centralities and egonet features.](image)

![Figure 4: Example of extracting embedding-based features for a target node. The left shows a derived graph $G_d$ with $V_d = \{a, d, e, i, j, k\}$. The nodes as well as their embeddings $e_i$ obtained from Node2Vec are colored according to their host classes. Note that the nodes of class X and class O share two different embeddings, i.e., $e_i = e_j = e_X$ and $e_k = e_O$. The target node is $a$ and its 1-hop egonet $\mathcal{E}(a) = (V_{\epsilon a}, E_{\epsilon a}, W_{\epsilon a})$ is marked by the rounded rectangle. The right shows the calculation of the four induced features $f_1, f_2, f_3, f_4$ for the target node $a$.](image)

[36, 37] in which the embedding similarities are used to detect lateral movements. For a target node, we first fetch its 1-hop egonet and the embeddings of its egonet nodes and then we calculate the four features like [36]. They are the minimum and mean of the similarities between the target node and its egonet nodes, the mean of the similarities for all node pair in the egonet, and the minimum of the similarities between the egonet nodes and their embedding centroid. It is assumed that if a host interacts with unfamiliar hosts, such as lateral movements occurring, these features could be influenced.
\[
Z = g_\Theta (X, G_d),
\]

where \( \Theta \) denotes all the learnable parameters of the GNN. It takes the base host feature vectors \( X \) and the derived graph \( G_d = (V_d, E_d, W_d) \) as input and outputs the enhanced host feature vectors \( Z \in \mathbb{R}^{\ell \times |D_{gw}|} \). Similar to \( \Theta \), the enhanced vector of host \( v \in V \) is the \( \phi(v) \)-th row of \( Z \), i.e., \( z_v = (Z[\phi(v), :])^T \). In general, a GNN stacks \( L \) layers and outputs node representations by forwarding the input layer by layer recursively like

\[
Z^\ell = g_\Theta (Z^{\ell-1}, G_d), \quad \text{for} \quad i = 1, 2, \ldots, L,
\]

where \( g_\Theta \) is the \( \ell \)-th layer of the GNN, \( Z^\ell \) is the output of the \( \ell \)-th layer, \( Z^0 = X \), and \( Z = Z^L \). At each layer, the input representations of nodes and their first-order neighbors are integrated to generate their output representations by two main operations \textit{neighborhood aggregation} and \textit{representation updating}. By forwarding through the layers, the high-order host interaction information, including the base host feature vectors and the structural information within \( L \)-hop neighborhood, is embedded into the enhanced host feature vectors.

Specifically, considering that the host interaction context is changing over different time windows, we choose GraphSAGE [38] in this work because it is an inductive learning framework which is able to generalize across graphs. Our GraphSAGE has two layers. The aggregation operation at the \( \ell \)-th layer can be formalized as (10) and (11).

\[
\bar{X} (v, \ell) = \mathcal{S} (v, \mathcal{N}(v), G_d, \mathcal{S}^\ell),
\]

\[
h^\ell_{\mathcal{F}(v)} = \frac{1}{S^\ell} \sum_{u \in \mathcal{N}(v, \ell)} z^{\ell-1}_u.
\]

It first draws a subset \( \bar{X} (v, \ell) \) of size \( S^\ell \) from the target node \( v \)’s first-order neighbors \( \mathcal{N}(v) \) by the sampling function \( \mathcal{S} \). \( \mathcal{S} \) samples a node \( u \in \mathcal{N}(v) \) by the proportion to the bidirectional interaction strength between \( u \) and \( v \) which can be obtained from the weight matrix \( W_d \) of \( G_d \), i.e., \( W_d [\phi(v), \phi(u)] + W^T_d [\phi(u), \phi(v)] \). Then, it aggregates the sampled nodes’ representations output at the \( (\ell - 1) \)-th layer as (11). The update operation at the \( \ell \)-th layer is performed as follows.

\[
h^\ell_v = \eta (W^\ell \cdot [z^{\ell-1}_v h^\ell_{\mathcal{F}(v)}] + b^\ell),
\]

\[
z^\ell_v = h^\ell_v \| h^\ell_{\mathcal{F}(v)} \|_2,
\]

where \( \eta \) is the activation function and \( W^\ell \) and \( b^\ell \) are the learnable parameters of the \( \ell \)-th layer. It first concatenates \( z^{\ell-1}_v \) and \( h^\ell_{\mathcal{F}(v)} \) as one vector \( [z^{\ell-1}_v h^\ell_{\mathcal{F}(v)}] \) and feeds it into a fully connected layer as (12). Then, it updates the representation of the target node with the normalized output of the fully connected layer as (13).

### 3.3. Network Behavior Anomaly Detection

During this stage, XNBAD integrates the enhanced host interaction feature vectors output from GNN and the flow feature vectors to generate the network behavior representations and then estimates their anomaly degrees by an anomaly detector. In this work, we concatenate the host and flow feature vectors directly as the behavior feature vector and use a clustering-based detector for detection, which is shown in Algorithm 3. Thus, we rewrite (1) as (14) for each flow record \( r = (s, d, \mu, q) \) in the current window.

\[
v_r = [z_s z_d q],
\]

where \( z_s \) and \( z_d \) are the enhanced feature vectors of hosts \( s \) and \( d \) obtained from the GNN and \( q \) is the flow feature vector. The clustering-based detector profiles normal network behaviors by a set of \( K \) cluster centers \( \mathcal{C} = [c_k]_{k=1}^{K} \) where each center \( c_k \) could represent a kind of normal behavior patterns gathering in the feature space. The detector measures the anomaly degree of a network behavior by the distance between the behavior feature vector and the cluster centers as (15) and (16).

\[
\mathcal{A}(r) = \min_{c_k \in \mathcal{C}} \{\text{dis}(v_r, c_k)\},
\]

\[
\text{dis}(v_r, c_k) = \|v_r - c_k\|_2.
\]

The intuition is that if a behavior is normal, it will lie close to one of the cluster centers, showing a short distance between it and its cluster; otherwise, it will stay away from all the cluster centers, having a relatively long distance.

### 3.4. Learning Procedures

In this section, we describe how to learn the Node2Vec embeddings, the graph neural network, and the cluster centers mentioned above. Among them, the embeddings and the centers are generated by the off-the-shelf models while the graph neural network needs to be guided by a specific loss function. Therefore, we introduce the training data organization for these three models and the loss function used for the graph neural network. These models are trained on the historical normal traffic that may span for hours, days, or more. Given a training set of \( T \) windows of flow records \( \mathcal{R} = [R^{(t)}]_{t=1}^{T} \), we first construct the corresponding HIGs \( \mathcal{G} = [G^{(t)}]_{t=1}^{T} \) and their derived simple graphs \( \mathcal{G}^d = [G^{(t)}_d]_{t=1}^{T} \) where \( G^{(t)} = (V^{(t)}, E^{(t)}) \) and \( G^{(t)}_d = (V^{(t)}, E^{(t)}_d, W^{(t)}_d) \) and then train these models one by one as follows.

#### 3.4.1. Node2Vec

We generate a closed-world simple graph which summarizes the historical host interaction for training Node2Vec. This can be implemented by constructing a big HIG and its derived simple graph from all the records in \( \mathcal{R} \) and then merging the derived graph’s nodes, edges, and weights in the closed-world setting. In the closed-world setting, the internal user and server nodes are kept still while the other internal nodes and the external nodes are merged into two global nodes, respectively, and then the edges are merged and the weights are summed up accordingly. Then, we train Node2Vec on the training graph to obtain the node embeddings.
3.4.2. GraphSAGE. The learning objective of the GraphSAGE in this work is to maximize the normal host interaction context likelihood estimated from the enhanced host interaction feature vectors. We estimate the likelihood of a pair of nodes having interaction (edges) based on the inner product of their enhanced feature vectors. In $G^{(i)} = (V^{(i)}, E^{(i)})$, for a target node $v \in V^{(i)}$, the remaining nodes can be split into the positive set $V^{(i)}_+ (v)$ in which the nodes have edges with $v$ (i.e., 1-hop neighbors of $v$) and the negative set $V^{(i)}_- (v)$ in which the nodes have no edge with $v$. Then, the likelihood of node $v$ interacting with the others under the context $G^{(i)}$ can be estimated as

$$p(v|G^{(i)}) = \prod_{u \in V^{(i)}-\{v\}} p_{u,v}$$

$$= \prod_{u \in V^{(i)}_+(v)} \sigma(z_u^{(i)\top} z_v^{(i)}) \cdot \prod_{u \in V^{(i)}_-(v)} 1 - \sigma(z_u^{(i)\top} z_v^{(i)}).$$

(17)

where $z_u^{(i)}$ denotes the enhanced feature vector of node $v$ output by the GraphSAGE and $\sigma(\cdot)$ is the sigmoid function which ensures the estimated probability $p_{u,v} \in [0,1]$. Then, we can estimate the interaction context likelihood over the nodes and graphs of the training set like

$$p(G^{(i)}) = \prod_{v \in V^{(i)}} p(v|G^{(i)}) \cdot p(\mathcal{G}) = \prod_{t=1}^T p(G^{(i)}).$$

(18)

However, calculating the likelihood over nodes will require $|V^{(i)}| \times (|V^{(i)}| - 1)/2$ times the inner product computations, which costs a huge computation if $|V^{(i)}|$ is large. To control the computation, we use the predefined-size subsets instead of $V^{(i)}_+(v)$ and $V^{(i)}_- (v)$ themselves and use a subset of $V^{(i)}$ when calculating the likelihood over nodes. Consequently, the negative log likelihood over graphs is calculated as the total loss function of the GraphSAGE like (19). Note that $1 - \sigma(x) = \sigma(-x)$.

$$\mathcal{L}(\mathcal{G}) = -\sum_{t=1}^T \sum_{v \in V^{(i)}} \sum_{u \in V^{(i)}_+(v)} \log \sigma(z_u^{(i)\top} z_v^{(i)})$$

$$+ \sum_{u \in V^{(i)}_-(v)} \log \sigma(-z_u^{(i)\top} z_v^{(i)}),$$

(19)

where $V^{(i)}_+(v) = \mathcal{S}(v, V^{(i)}_+(v), G^{(i)}_d, S_d)$ is the subset of size $S$, drawn from $V^{(i)}_+(v)$ by the sampling function $\mathcal{S}$ introduced in (10), $V^{(i)}_- (v)$ is the subset of size $S$, uniformly drawn from $V^{(i)}_-(v)$, and $V^{(i)}$ is the subset of $V^{(i)}$ which we describe later. We apply a gradient descent-based optimization method to minimize $\mathcal{L}(\mathcal{G})$ and update the learnable parameters $\Theta$ of the GraphSAGE in a mini-batch fashion.

3.4.3. K-Means. We run K-means algorithm on a number of network behavior feature vectors to obtain the normal behavior cluster centers. To capture representative normal behavior patterns, we use the behavior vectors across different windows as the training set of K-means. However, enormous flow records can be collected in hours, and using all of them for training is unnecessary and impractical. Therefore, we randomly sample a part of them from each window in proportion to train K-means. Suppose the total number of the sampled records is predefined as $N_t$; then, for the $t$-th window, its proportion is $p^{(t)} = |R^{(t)}|/\sum_{t=1}^T |R^{(t)}|$, so we draw a subset $R^{(t)}_d$ of $p^{(t)} N_t$ records from $R^{(t)}$. This sampling procedure is prepared at the very beginning of training all the models, and $R^{(t)}$ in (19) is obtained by collecting the hosts in $R^{(t)}_d$.

3.5. Time Complexity of Detection Phase. This section analyzes the time complexity of XNBA using detection on a window of flow records, in which data scale and feature dimensionality are mainly considered. The time complexities of different stages of XNBA are summarized in Figure 5 and analyzed as follows.
3.5.1. Flow Collection and Feature Extraction. The flow feature extraction is performed incrementally as shown in Algorithm 1. Therefore, when each packet arrives, all the flow features of the corresponding flow record could be initiated or updated. Let |P| be the packet arrival rate, and the time complexity of collecting a window of flow records is \( O(|P|D_{\text{flow}}) \).

3.5.2. Host Interaction Feature Generation. This stage contains three steps: graph construction, base feature extraction, and GNN-based feature enhancement. The time complexities of the first and last steps are relatively determined while the second is highly dependent on what base features are used in the exact environment. Let \( R \) be a window of flow records, and \( G = (V,E) \) and \( G_d = (V,E_d,W_d) \) be the HIG and the derived HIG constructed from \( R \) as shown in (3) and (6), respectively. (1) Graph Construction. Constructing \( G \) and \( G_d \) needs one pass through \( R \). Besides, auxiliary data structures such as adjacent tables and alias sampling tables [39] of nodes need to be created for the subsequent feature extraction and enhancement, and they can be obtained in \( O(|E_d|) \) time. Totally, the time complexity in this step is \( O(|R| + |E_d|) \). (2) Base Feature Extraction. The time complexity of this step is usually linear with the dimensionality of base features and may have a linear or higher-order relationship with the number of nodes or edges. In this work, it is concluded as \( O(|R|D_{do m} + |V|D_{do m} + |V|^2E_d + |V|^2D_{emb} + |E_d|^2) \) where \( D_{do m} \) and \( D_{emb} \) are the dimensionality of the domain-based features and the node embeddings, respectively. The time complexity of this step is \( O(|R|D_{do m} + |V|E_d) \) since getting host classes, filling the counting matrices (e.g., \( M_{D_{do m}} \)), and computing the interactive distribution features can be done in one pass through \( R \) and \( V \). The time complexity of extracting graph-based features is \( O(|V|^2E_d) \). This is because getting the number of 2-step-away node pairs in the egonet needs to search in \( E_d \times V \) space for all nodes in \( V \), while the others can be done in \( O(|V||E_d|) \) at most. The time complexity of extracting embedding-based features is dominated by computing the induced feature \( f_3 \) (the averaged embedding similarity of node pairs in the egonet). Computing \( f_3 \) includes pre-computing the similarities for all node pairs within \( O(|V|^2D_{emb}) \) and accumulating the node pair similarities for each egonet. Suppose \( V_{\text{eg}} \) is node \( a \)’s egonet node set and note that \( \sum_{a \in V}|V_{\text{eg}}| \leq 2|E_d| \), and it has totally \( \sum_{a \in V}|V_{\text{eg}}|(|V_{\text{eg}}| - 1)/2 \) of node pairs to check and therefore needs \( O(\sum |V_{\text{eg}}|^2) = O(|E_d|^2) \) time in the worst case. (3) GNN-Based Feature Enhancement. Suppose the GNN has \( L \) layers, the sampling size is \( S' \), and the output dimensionality is \( D'_{\text{gnn}} \) at the \( \ell \)-th layer. For each node, neighborhood sampling (equation (10)) can run in \( O(S') \) by the alias method [39]; aggregating (equation (11)) needs \( O(S'D_{\text{emb}}^{\ell-1}) \), full connection layer forward (equation (12)) needs \( O(S'D_{\text{gnn}}^{\ell-1}) \), and L2-normalization (equation (13)) needs \( O(D_{\text{gnn}}^{\ell-1}) \). Therefore, the time complexity of the \( \ell \)-th GNN layer for a node is \( O(S'D_{\text{gnn}}^{\ell-1} + D_{\text{gnn}}^{\ell-1} \cdot D_{\text{emb}}^{\ell-1}) \) with \( D_{\text{gnn}}^0 = D_{\text{base}} \).
In practice, GNN outputs node representations for a batch of nodes once. Therefore, the overall time complexity of the GNN on a window is

$$\Theta\left(\sum_{l=1}^{L} N_{\text{out}}^{l} O_{\text{gmm}}(S^l + D_{\text{gmm}}^l)\right).$$

(20)

where $N_{\text{out}}^{l}$ is the number of unique nodes output at the $l$-th layer and $\beta = |V|/N_{\text{out}}^{l}$ is the number of batches the GNN runs on $G_d$. Note that $N_{\text{out}}^{l-1} \leq \min(|V|, N_{\text{out}}^{l} (1 + S^l))$ due to the sampling operation. In the worst case, $G_d$ is large enough, so $\forall l \leq L$, $N_{\text{out}}^{l-1} = N_{\text{out}}^{l} (S^l + 1)$ which results in $N_{\text{out}}^{l} = N_{\text{out}}^{1} \cdot \Pi_{l=1}^{L} (S^l + 1)$. In order to keep clarity, we reuse $S^l$ as $S^l + 1$ and set $S^l$ and $D_{\text{gmm}}^l$ to $S_{\text{gmm}}$ and $D_{\text{gmm}}$, respectively, for all layers. Then, (20) can be simplified to

$$\Theta\left(|V| \left(\sum_{l=1}^{L} O_{\text{gmm}}(S_{\text{gmm}} + D_{\text{basis}} + D_{\text{gmm}}^l + D_{\text{flow}}^l)\right)\right).$$

(21)

3.5.3. Network Behavior Anomaly Detection. For each flow record $r \in R$, getting its network behavior feature vector $v_r$, calculating the distances between $v_r$ and $K$ cluster centers, and selecting the minimum distance, respectively, need $\Theta(1)$, $\Theta(KD_{\text{bhv}})$, and $\Theta(K)$ time, where $D_{\text{bhv}} = 2D_{\text{gmm}} + D_{\text{flow}}$ is the dimensionality of $v_r$. Therefore, the overall time complexity of this stage is $\Theta(\sum_{r \in R} KD_{\text{bhv}})$.

4. Dataset

We conducted experiments on the ISCX-2012 dataset [40]. In this section, we first introduce the basic information and our refinement about this dataset before detailing the experiment procedures and results.

The ISCX-2012 dataset was collected in a real live testbed simulating an intranet environment with 4 user LANs (denoted as $l_1-l_4$) and 1 server LAN ($l_5$). The user LANs have mainly 21 active user hosts ($u_1-u_{21}$) and the server LAN has 3 servers, i.e., Main Server ($s_{21}$), Secondary Server ($s_{22}$), and NAT Server ($s_{23}$). The LAN hosts can reach the external network through NAT Server. Besides, it has 3 attack hosts ($a_1-a_3$) located in the external network. The traffic of the dataset is generated by profiles. To simulate realistic network activities, the dataset creators analyzed and summarized four weeks worth of network activities in their institution into the $\beta$-profiles to generate the background (normal) traffic for HTTP, SMTP, SSH, IMAP, POP3, and FTP protocols. At the same time, they conducted $\alpha$-profiles to describe multi-stage attack scenarios and generate the malicious traffic. Concretely, four multi-stage attack scenarios were carried out in four different days and they are Infiltrating on Jun13, HTTP-DoS on Jun14, Botnet-DDoS on Jun15, and SSH-BruteForce on Jun17. Furthermore, these four scenarios are designed to have associations so that they are presented together as a more sophisticated attack scenario. Totally, the dataset contains about 91 GB raw network traffic captured for a duration of 7 days, from June 11 to June 17, which is stored in seven PCAP files by days and it provides flow-based ground truth XML files for attack days. All the network interactions (not only between the LAN hosts but also between the LAN hosts and the external hosts) are preserved in the traffic.

There are several reasons for choosing the ISCX-2012 dataset. First, this work aims to detect malicious network behaviors in a relatively local environment such as a campus or enterprise intranet, so the network environment simulated in this dataset meets the requirement of this work. Second, this dataset contains complete attack scenarios and relatively abundant attack behaviors, which makes evaluations as comprehensive as possible. Third, it provides very detailed materials such as the metadata of the network and the step-by-step attack details in paper [40], the complete raw traffic, and the useable label files, so that the dataset users are able to deeply understand the activities in the network. Though there are some newer datasets released like CIC-IDS-2017 and CIC-IDS-2018 [41], these datasets have not provided such detailed information. Therefore, it takes more time to study these datasets and we will leave the evaluations on these datasets in future work.

We studied the ISCX-2012 dataset by carefully cross-checking among the paper, the traffic, and the labels, during which we found two main problems which could have affected the experiment results and analysis. First, the original label files only provide binary labels, i.e., Normal or Attack. It may be sufficient for measuring the performances of detectors but cannot help further analyze their advantages and disadvantages. Note that the attack scenarios are multi-stage, which means there are many different types of malicious behaviors in a scenario. The "Infiltrating" scenario, for example, has the behaviors of scanning, web-based attacking, backdoor reverse connecting, and so on. Therefore, when given the binary label, one can only tell whether the corresponding flow is malicious but hardly distinguish what type of behavior it belongs to. Second, there are several mislabeled and missing records for attack flows in the original label files because the label generation is based on an automatic analysis tool. For example, all the backdoor reverse connecting records since Jun15 are mislabeled as normal while those before Jun15 are labeled correctly as attacks. In Jun15, about six thousands of DDoS attack records from the user host $u_9$ are mislabeled as normal and the phishing record is missing.

To report more accurate results and more detailed analysis, we made a correction, summarized, and added the behavior labels. We extracted the flow records from the traffic and then labeled them with their behavior types. The behavior distribution is shown in Table 2. In general, there are totally 14 different types of malicious network behaviors in the dataset. Four of them are high-volume malicious behaviors, i.e., scan, HTTP DoS, botnet DDoS, and SSH brute force while the rest are subtle malicious behaviors, only taking up a very limited proportion in the total malicious behaviors. We further specify the hosts involved with the behaviors, which can help us understand the behavior semantics and analyze detection results. For example, in Table 2, scan ($u_5$, $l_{13}$) means the behaviors of the user host $u_5$ scanning the first and second LANs, and control ($u_5$, $u_{12}$) means the behaviors of attackers controlling the user host $u_{12}$ through the user host $u_5$. 
Table 2: Behavior distribution of ISCX-2012. The table header specifies the corresponding days and scenarios of the columns. Each cell of the table contains a behavior type and its corresponding number. To express the semantics of the behaviors, the source and destination hosts involved in a malicious behavior are also specified in the bracket where $u_s$, $s_m$, and $a_m$, respectively, stand for the user, server, and attacker hosts while $l_i$ stands for the internal LAN. Especially, if multiple hosts or LANs are involved as one end of a behavior, they are split by commas in the subscript such as $u_{1,6,13,15,17}$ and $l_{1,2}$. The behavior types are sorted in the descending order of the number.

| Jun11 Attack free | Jun13 Infiltrating | Jun14 DoS | Jun15 DDoS | Jun17 Brute force |
|------------------|-------------------|-----------|------------|------------------|
| Normal           | Normal            | Normal    | Normal     | Normal           |
| 401 677          | 153 808           | 194 732   | 546 717    | 382 471          |
| Scan ($u_s$, $l_l$) | HTTP DoS ($u_{1,6,13,15,17}$, $s_{22}$) | 7954      | Botnet DDoS ($u_{5,9,10,13,18,20}$, $s_{22}$) | 43 622 |
| 1564             | Scan ($u_{12}$, $l_3$) | 1530      | Botnet C2 ($u_{12}$, $u_{3,5,9,10,13,18,20}$) | 113 |
| Backdoor connection ($u_s$, $a_l$) | Control ($u_{12}$, $u_{1,6,13,15,17}$) | 144       | Backdoor connection ($u_{12}$, $a_2$) | 126 |
| 125              | Backdoor connection ($u_{12}$, $a_2$) | 65        | HTTP GET download ($u_{3,5,9,10,13,18,20}$, $a_3$) | 14 |
| Control ($u_s$, $u_{12}$) | Slowloris download ($u_{1,6,13,15,17}$, $a_2$) | 106       | Phishing ($a_1$, $s_{22}$) | 1 |
| 39               | Web attacks ($u_{12}$, $s_{22}$) | 4         | Failed FTP download ($a_1$, $s_{22}$) | 7 |
| Backdoor heartbeat ($u_{12}$, $a_2$) | Phishing ($a_1$, $s_{22}$) | 1         |                      |      |
|                  | Phishing ($a_1$, $s_{22}$) | 1         |                      |      |

5. Experiments

We conducted experiments on ISCX-2012 dataset to evaluate the detection performance of XNBAD. We compared it with several representative methods and further analyzed their abilities of discovering different types of malicious behaviors. For each method, we trained a model with the normal traffic on Jun11 and tested it on the four attack days, i.e., Jun13, Jun14, Jun15, and Jun17. Since the performance may vary a lot with models’ hyperparameters, we first ran hyperparameter search and selected the best models for comparison. Meanwhile, we studied the influence of the hyperparameters to the detection performance of XNBAD. Besides, to study the effectiveness of the GNN-based feature enhancement in XNBAD, we conducted the ablation experiments that compared the models with and without the GNN. Finally, we analyzed the runtime and scalability of XNBAD. The experiments were conducted on Ubuntu 16.04 64-bit OS with an Intel Core i7-7700 3.60 GHz CPU and 32 GB RAM, and the models were implemented in Python using DPKT, Scikit-Learn, and PyTorch. In the following sections, we describe the metrics, data preprocessing, and models as well as their hyperparameters considered in the experiments.

5.1. Metrics. In the experiments, we used the weighted area under the curve to measure the detection performance and select the best models.

The area of under the curve (AUC) is a threshold-free classification metric, widely used in unsupervised anomaly detection task, and it is more robust than the threshold-based metric accuracy (ACC) [42]. However, we found that using the general AUC for model selection and comparison will bias the models only good at discovering high-volume attack behaviors. For example, on Jun15 as shown in Table 2, the model which distinguishes most of the DDoS attack behaviors from the normal will achieve a high general AUC value even if it fails to detect the other behaviors. Even if it can detect the others, the gain of the general AUC value is trivial. The reason is that when calculating the general AUC, all the attack samples in the test set are treated equally, weighted as 1.0 implicitly, which makes high-volume attacks count more. In fact, compared to such high-volume attacks, subtle attacks are more valuable and more difficult to discover. Therefore, the performance of discovering subtle attacks is also needed to be reflected in the metric.

The weighted AUC, which is calculated on explicitly weighted samples, is more suitable for the anomaly detection problems having imbalanced multiple types of anomalies. To eliminate the bias of the general AUC, we weighted all anomaly types equally on each attack day. Suppose there are totally $n$ samples on an attack day and they are divided into $m$ types, i.e., $\{T_1, T_2, \ldots, T_m\}$, where $n = \sum_{i=1}^{m} |T_i|$; then, the weight of type $T_i$ is assigned to $\frac{n}{m}$ for $i = 1, 2, \ldots, m$ and the weights of its samples are assigned to $\frac{1}{m} |T_i|$. For a detection method, we randomly sampled 100 thousands of flows on Jun11 for training and then calculated the weighted
AUC on each attack day and averaged the weighted AUCs over days to measure the overall detection performance. We repeated this procedure for 10 times and used the mean values for comparison. Besides, the general AUCs were also reported for reference.

5.2. Data Preprocessing. We performed data preprocessing to make sure the values of the flow and base host feature vectors are in the range \([0, 1]\). There are mainly two principles of the preprocessing. First, for the features having definite lower and upper bounds \([m, M]\), we transformed the feature value \(v\) by min-max normalization, i.e., \(v' = (v - m)/(M - m)\). Second, for those not having bounds (but they show a long-tail distribution), we transformed the feature by logarithmic binning. For example, suppose we need to transform a feature ranging in \([0, +\infty)\) into three bins of width \(q \in (0, 1)\) by logarithmic binning and this feature has a number of observed values in the training set. We first calculate the bin intervals like \([b_1, b_2], [b_2, b_3], \ldots\) where \(b_1 = 0, b_2 = \text{the } q\text{-th quantile of the all observed values, and } b_3 = \text{the } q\text{-th quantile of the rest observed values out of } [b_1, b_2]\). Then, we expand an original feature value \(v\) to three new values corresponding to the three bins. The new values are initiated as zeros. For the bin which \(v\) falls into, its value is set to 1 if it is the last opening bin; otherwise, it is set to the min-max normalized value of \(v\) in the bin. After preprocessing, the flow feature dimensionality \(D_{flow} = 194\) and the base host feature dimensionality \(D_{base} = 400\).

5.3. Models and Hyperparameters. We considered three representative baselines in the experiments. They were the packet-based method Kitsune [4], the static embedding-based method SkipGramIDS [12], and the flow-based baseline named CICFlowKMS that extracts the flow features by CICFlowMeter [18] and utilizes K-means for detection. The experiment settings of XNBAD and its baselines are, respectively, described as follows.

5.3.1. XNBAD. It contains two learning modules here, the K-means clustering-based anomaly detection module and the two-layer graph neural network GraphSAGE. The cluster number \(K\) of K-means was selected from \([25, 50, 75, 100]\). The hyperparameters of layers of the GraphSAGE were kept the same. The neighborhood sampling size was selected from \([5, 10, 15, 20, 25]\), and the output host feature size was selected from \([100, 150, 200, 250, 300]\). Three activation functions, linear, ReLU [43], and sigmoid, were considered. To train the GraphSAGE, we set both the positive and negative sampling sizes to 10 and used the mini-batch of size 50 for loss calculation. Then, we utilized the Adam optimizer [44] for training with the learning rate of 0.1 for one epoch. Besides, we applied \(L_1\)-regularization with weight \(\lambda = 0.1\) to prevent overfitting and applied global clipping over 5 for the possible gradient explosion.

5.3.2. Kitsune. This packet-based method learns from continuous packets and calculates the anomaly score for each packet. Since all the other methods give detection results on flows, we aggregated the packet scores within a flow into the flow score, where Min, Max, and Mean aggregations were considered. For a reasonable comparison, we used first 5.4 million continuous packets on Jun11 for training (amount to the number of training flows of other methods): the first 3 million for the feature extraction learning and the remaining 2.4 million for the autoencoder training. The other hyperparameters of Kitsune were kept as default. Since the original version runs slow even with Cython accelerated, we reimplemented it from the official codes with a little modification for acceleration. The preliminary test showed no detection performance decline of our modified version.

5.3.3. SkipGramIDS. It estimates the abnormal degree of the flows by the similarity of the embeddings of words IP, IP, Port, Protocol. As discussed in Section 2, it cannot run on the unseen IPs and the destination port numbers. Therefore, we did a workaround (closed-world setting) as Ring et al. [14] suggested. Concretely, the internal IPs unseen in the training set, all the external IPs, and the destination port numbers greater than 1024 were mapped to three default values, respectively. In this way, all the words are included in the training set and they can be mapped to the corresponding learned embeddings. SkipGram [13] is used for embedding learning. The embedding size was selected from \([4, 32, 100, 200]\), the context window size was selected from \([2, 5, 10]\), the minibatch size was selected from \([8, 32, 50]\), and the epoch was varied in \([20, 200, 1000, 5000]\).

5.3.4. CICFlowKMS. It builds the normal profile by applying the K-means clustering on flow feature space and calculates the distances from the cluster centers as anomaly scores, which is similar to XNBAD. The clustering number was also considered in \([25, 50, 75, 100]\).

6. Results and Analysis

6.1. Performance Comparison. The detection performances of XNBAD and its three baselines are shown in Table 3. We mainly compared their detection performances by the weighted AUC since it is able to appropriately reflect the ability of detecting the various subtle malicious network behaviors as well as the high-volume ones. Among these four methods, Kitsune was the worst one, only reaching an overall weighted AUC of 0.6266. Its weighted AUCs on the four attack days were around 0.5535 ± 0.6802. Both of SkipGramIDS and CICFlowKMS performed better than Kitsune. The static embedding-based method SkipGramIDS reached the weighted AUCs over 0.8370 on most attack days except Jun17 (0.7625), resulting in an overall weighted AUC of 0.8333. CICFlowKMS achieved a relatively high overall performance (0.9153), becoming the best baseline, and its performance on all attack days was over 0.9400 except that
The relative improvements (Impr.) of XNBAD to each baseline are also listed. Almost all of the improvements are statistically significant at the level of 1% (obtained $p < 1\%$ in $T$-test).

| Day (scenario) | Kitsune | SkipgramIDS | CICFlowKMS | XNBAD |
|---------------|---------|--------------|------------|--------|
| Mean ± std    | Impr. (%)| Mean ± std   | Impr. (%)  | Mean ± std |
| Jun13 (infiltrating) | 0.5535 ± 0.0143 | 74.6 | 0.8819 ± 0.0405 | 9.6 | 0.9454 ± 0.0053 | 2.2 | 0.9663 ± 0.0023 |
| Jun14 (DoS)   | 0.6327 ± 0.0083 | 54.0 | 0.8513 ± 0.0609 | 14.5 | 0.9674 ± 0.0056 | 0.8 | 0.9746 ± 0.0038 |
| Jun15 (DDoS)  | 0.6399 ± 0.0087 | 42.9 | 0.8373 ± 0.0373 | 9.2 | 0.8083 ± 0.0044 | 13.1 | 0.9141 ± 0.0122 |
| Jun17 (brute force) | 0.6802 ± 0.0188 | 38.9 | 0.7625 ± 0.1053 | 23.9 | 0.9400 ± 0.0069 | 0.5 | 0.9446 ± 0.0057 |
| Overall       | 0.6266 ± 0.0078 | 51.6 | 0.8333 ± 0.0344 | 14.0 | 0.9153 ± 0.0036 | 3.8 | 0.9499 ± 0.0029 |

*Not significant ($p > 5\%$).

To gain further insights of the detection performances of the four methods on various malicious behaviors, we deeply studied their detected malicious behaviors at the false positive rate of 0.1 using the refined behavior labels obtained during our dataset study. The average true positive numbers and rates over the ten repetitions are shown in Table 5 and they enlighten us how these methods resulted in the performances shown in Tables 3 and 4. Our analysis is detailed as follows.

6.2.1. Kitsune. It is observed that Kitsune indeed preferred to find the behaviors of the high-volume attacks such as scanning, DoS, and DDoS, but it could barely reveal the subtle attacks. For example, it detected average 3.8 out of 144 of the backdoor connection behaviors initiated from $u_5$ to $q_5$, and it failed to report any of the web attacks. Since Kitsune uses an ensemble of autoencoders which has a quite powerful learning ability, its poor performance is mainly due to its packet header-based features which could not provide enough discriminative information for detection.

6.2.2. SkipGramIDS. Obviously, it performed better than Kitsune on detecting the subtle malicious behaviors such as the backdoor connection and botnet C2, which means the learned embeddings are relatively informative. However, it detected much fewer high-volume attack behaviors than the other three methods, especially on the SSH brute force.
The current state of this shortage. First, too many flows trying SSH login made features. As mentioned in Section 2, there are two reasons for network environments. Since CICFlowKMS did not perform very comprehensively. Compared to the best baseline CICFlowKMS, XNBAD had an absolute advantage in detecting backdoor (TP) and true positive rate (TPR) are reported for each method. The highest #TP on each row is given in bold. It is obvious that XNBAD performed well on the SSH brute force behaviors either, we believe the main reason for its unsuccessful detection was lack of generalization rather than lack of discriminative flow features. As mentioned in Section 2, there are two reasons for this shortage. First, too many flows trying SSH login made the current state of s22 (MainServer) anomalous but the learned embedding could not reflect the anomalous state as XNBA did. On the contrary, they only represented the outdated state of s22 and its corresponding connection type (s22, 22, tcp). Second, SkipGramIDS could not directly provide the embedding of a1 because a1 is unseen in the training set. It had to use the corresponding default embedding for a1, which was also outdated. Therefore, the co-occurrence of (a1, s22, 22, tcp) was improperly estimated as normal. This result also implies the importance of considering timely host interaction states in anomaly detection. 6.2.3. CICFlowKMS. It had a relatively satisfactory detection result on most of malicious behaviors, especially on scan, control, phising, and backdoor heartbeats. This result confirmed that the flow statistical features indeed provide discriminable information for detection. Nevertheless, it still performed poorly on several types of the subtle malicious behaviors. Concretely, it detected only average 40.8% (58.7% of the backdoor connections initiated from u4 to a1, 15.4% (19.3/125) of those from u12 to a1, 30.3% (58.5/193) of the botnet C2 behaviors, and none of the attack tool HTTPGET download behaviors. Besides, it was not

| Day/Behavior | Total | Kitsune | SkipGramIDS | CICFlowKMS | XNBAD |
|--------------|-------|---------|-------------|------------|--------|
|              |       | #TP     | TPR (%)     | #TP        | TPR (%) | #TP     | TPR (%) | #TP     | TPR (%) |
| Jun13        |       |         |             |            |        |         |         |        |         |
| Scan (u25, l12) | 7954  | 7500.4 | 94.3        | 6313.2     | 79.4   | 7575.2  | 95.2    | 7582.1 | 95.3    |
| Scan (u12, l5) | 1564  | 1265.7 | 80.9        | 742.5      | 47.5   | 1349.7  | 86.3    | 1348.5 | 86.2    |
| Backdoor connection (u6, a1) | 144   | 3.8     | 2.6         | 43.2       | 30.0   | 58.8    | 40.8    | 144.0  | 100.0   |
| Backdoor connection (u12, a1) | 125   | 3.3     | 2.6         | 62.5       | 50.0   | 19.3    | 15.4    | 125.0  | 100.0   |
| Control (u6, u12) | 106   | 2.9     | 2.7         | 59.8       | 56.4   | 106.0   | 100.0   | 105.8  | 99.8    |
| Web attacks (u12, s22) | 39    | 0.0     | 0.0         | 23.3       | 39.7   | 80.0    | 97.4    | 39.0   | 100.0   |
| Backdoor heartbeat (u12, a1) | 4     | 0.0     | 0.0         | 2.0        | 50.0   | 4.0     | 100.0   | 4.0    | 100.0   |
| Phishing (a1, s22) | 1     | 0.0     | 0.0         | 0.6        | 60.0   | 1.0     | 100.0   | 1.0    | 100.0   |
| HTTP DoS (u16,8,13,15, s22) | 1970  | 1865.0  | 94.7        | 1305.9     | 66.3   | 1920.6  | 97.5    | 1953.6 | 99.2    |
| Scan (u12, l5) | 1500  | 1192.8  | 78.0        | 1337.8     | 87.4   | 1357.2  | 88.7    | 1338.1 | 87.5    |
| Jun14        |       |         |             |            |        |         |         |        |         |
| Control (u12, u6,13,15,17) | 70    | 13.2    | 18.9        | 57.7       | 82.4   | 69.8    | 99.7    | 68.4   | 97.7    |
| Backdoor connection (u12, a2) | 65    | 1.6     | 2.5         | 32.6       | 50.2   | 57.4    | 88.3    | 64.7   | 99.5    |
| Slowloris download (u16,8,13,15,17, s22) | 62    | 3.3     | 5.3         | 5.7        | 9.2    | 61.0    | 98.4    | 61.0   | 98.4    |
| Botnet DoS (u15,3,5,10,13,18,20, s22) | 43622 | 43620.5 | 100.06      | 30329.2    | 69.5   | 43573.2 | 99.9    | 43608.3 | 100.07 |
| Botnet C2 (u12, u15,3,5,10,13,18,20) | 193   | 90.9    | 47.1        | 145.5      | 75.4   | 58.5    | 30.3    | 182.9  | 94.8    |
| Jun15        |       |         |             |            |        |         |         |        |         |
| Backdoor connection (u12, a2) | 78    | 0.2     | 0.3         | 34.6       | 44.4   | 60.5    | 77.6    | 67.5   | 86.5    |
| HTTPGET download (u3,5,3,10,13,18,20, a3) | 14    | 1.1     | 7.9         | 0.0        | 0.0    | 0.0     | 0.0     | 2.6    | 18.6    |
| Phishing (a1, s22) | 1     | 0.0     | 0.0         | 0.0        | 7.0    | 1.0     | 100.0   | 1.0    | 100.0   |
| Jun17        |       |         |             |            |        |         |         |        |         |
| SSH brute force (a1, s22) | 4943  | 5214.1  | 51.0        | 0.0        | 0.0    | 1298.2  | 26.3    | 3581.2 | 72.4    |
| Login and download (a1, s22) | 37    | 1.2     | 3.2         | 2.0        | 5.4    | 36.6    | 98.9    | 37.0   | 100.0   |
| Backdoor heartbeat (u12, a2) | 23    | 1.5     | 6.5         | 18.4       | 80.0   | 23.0    | 100.0   | 22.6   | 98.3    |
| Failed FTP download (a1, s22) | 7     | 0.0     | 0.0         | 7.0        | 100.0  | 5.7     | 81.4    | 5.2    | 74.3    |

*Not equal but very close to 100%.

Table 4: The mean values (mean) and standard deviations (std) of general AUCs achieved by different methods. The highest mean value on each row is given in bold. The relative improvements (Impr.) of XNBAD to each baseline are also listed. Almost all of the improvements are statistically significant at the level of 1% (obtained p < 1% in T-test).

Table 5: The detection results of different methods on various malicious behaviors at the false positive rate of 0.1. Both true positive number (#TP) and true positive rate (TPR) are reported for each method. The highest #TP on each row is given in bold. It is obvious that XNBAD performed very comprehensively. Compared to the best baseline CICFlowKMS, XNBAD had an absolute advantage in detecting backdoor connection, botnet C2, and SSH brute force behaviors.
6.3. Hyperparameter Influence. In the experiments, the best model of XNBAD was obtained when the neighborhood sampling size was 20, the output host feature size was 250, the linear activation was used for the GraphSAGE, and the cluster number was set to 75 on K-means. The detection performances with the hyperparameters varying around the best are shown in Figure 6.

Figure 6(a) illustrates the trend of the detection performance going with the cluster number. As observed, the weighted AUC grew from the minimum (below 0.94) to the maximum (near 0.95) when the cluster number increased from 25 to 75 and then it dropped a little when the cluster number got to 100, which indicates that the cluster number was a relatively important hyperparameter to the detection performance of XNBAD. The clusters represent the normal network behavior patterns. Therefore, the cluster number controls the level of detail in modeling the behavior patterns. Too few clusters (25 here), an insufficient description, lead to underfitting while too many lead to overfitting. Figure 6(b) shows only a slight oscillation of the detection performance with the neighborhood sampling size. It means that the detection performance was not very sensitive to the neighborhood sampling size.

The influence of the output size to the detection performance was more significant than that of the cluster number as shown in Figure 6(c). The weighted AUCs on the two sides (100 and 300) dropped largely from the maximum, which was partly due to the underfitting and overfitting of the GraphSAGE. In addition, we think it was the negative effect induced by the dimensionality difference between the output host features and the flow features. Since the anomaly score is distance based, the dimensions of the host and flow features decide their contributions to the anomaly score. For instance, the flow feature size is 194 here and if the output host feature size is set to 100, then the flow features will provide 94 more dimensions in calculating the anomaly score than the source or destination host features. Too large dimensionality difference leads to an improper contribution to the anomaly score and results in a poor performance.

Figure 6(d) presents the detection performance on the three different activation functions of the GraphSAGE in XNBAD. The activation function was also one of the hyperparameters that affect the detection performance. Surprisingly, using the linear activation had the best performance. Nevertheless, it is worthy to note that using the linear activation in the GraphSAGE does not mean that the output features are linear to the input features because the last operation of the layers of the GraphSAGE is $L_2$-normalization which introduces the non-linear characteristic into the output. When using the ReLU activation, the weighted AUC was under 0.92 and the error bar was from 0.91 to nearly 0.93, which indicates that the model with the ReLU activation performed poorly and unstably. Using the non-linear activation sigmoid resulted in a better performance with the weighted AUC over 0.93.

Together, the results in this section indicate that the detection performance of XNBAD was mainly affected by the output host feature size, the activation function, and the cluster number but not sensitive to the neighborhood sampling size.

6.4. Enhancement Effectiveness. To understand the effectiveness of the GNN-based feature enhancement, we conducted ablation experiments. In the experiments, we compared XNBAD with its non-GNN variant in which the GNN was removed and then the base host features were...
directly concatenated with the flow features to represent the network behaviors. The results are shown in Figure 7.

As observed, XNBAD outperformed its non-GNN variant by a large margin on all attack days, resulting in a relative improvement of 7.3% in terms of the overall weighted AUC. It means that the GNN is indispensable to the great detection performance XNBAD achieved. There are two benefits from applying the GNN. First, the GNN reduced the high-dimensional base host features to the low-dimensional features, which overcame the aforementioned negative effects introduced by the dimensionality difference between the host features and the flow features. As observed in Figure 6(c), it did not achieve a good performance when the dimensionality of the host features was too large (300) or too small (100), let alone the non-GNN variant using the base host features of size 400. Therefore, with a proper output size, the GNN made the integration of the flow features and the output host features in harmony. Second, the GNN captured the high-order structural information of the interaction graphs, which made the output host features discriminative beyond the base host features. To confirm this benefit alone, we additionally evaluated the performance of another variant that uses the GNN with the output size of 400 (the base feature size) and all the other hyperparameters unchanged. It turned out that even without dimensionality reduction, the GNN variant having the overall weighted AUC of 0.9069 still beat the non-GNN variant, which verifies the second benefit of the GNN.

In summary, the GNN-based feature enhancement applied in XNBAD was effective and it greatly improved the detection performance. The GNN served not only as a high-order feature extractor but also as a dimensionality reducer during the enhancement.

6.5. Runtime Analysis. To evaluate the runtime and scalability of XNBAD, we ran XNBAD sequentially over the four test days of ISCX-2012 dataset, where Algorithm 1 was run first to collect all the flow records and then Algorithms 2 and 3 were run on each window of records. We recorded the durations of each processes and then studied their accumulated runtime over all windows and their distributions of the runtime in a single window.

As shown in Table 6, the overall runtime of XNBAD on the four test days of ISCX-2012 dataset was about 1.3 hours, where the first stage, i.e., flow collection and feature extraction, took up most of the time (about 1.2 hours) while the remaining two stages together ran for about 9 minutes only. This is because the first stage needs to go through numerous packets for flow feature extraction, which leads to the heavy computation overhead, while the latter two stages only need to deal with flows, nodes, and edges that are much less than...
Figure 7: The comparison between the models with the GNN-enhanced host features and the non-GNN host features.

Table 6: The accumulated runtime (in seconds) of different stages and steps of XNBAD over four test days of ISCX-2012.

| Section          | Stage/step                          | Runtime |
|------------------|-------------------------------------|---------|
| §3.1             | Flow collection and feature extraction | 4133.05 |
| §3.2.1           | Graph construction                  | 6.34    |
| §3.2.2           | Base feature extraction             | 289.69  |
| §3.2.3           | GNN-based feature enhancement       | 38.87   |
| §3.3             | Network behavior anomaly detection  | 34.13   |
| §5.2             | Feature normalization               | 162.15  |
| Overall          |                                     | 4667.14 |

Figure 8: The runtime distributions of the window-level processes in XNBAD on the data scale of a single window. (a–f) The distributions of the main and most time-consuming processes. (g) The distribution of the whole window-level process in XNBAD. The runtime is in seconds. \(|V|, |E_d|, \text{ and } |R|\) are the number of nodes and edges of \(G_d\) and the number of flow records in a single window, respectively. Red lines are the fitted curves of the relationships between the runtime and data scale, with their fitting setting listed under the legends. (a) Extracting interactive distribution features (domain-based). (b) Extracting induced feature \(f_3\) (embedding-based). (c) Generating GNN-enhanced host features. (d) Detecting anomalous network behaviors. (e) Normalizing host features. (f) Normalizing flow features. (g) Whole window-level process.
packets. Nevertheless, compared to the four-day duration, the overall runtime of XNBAD was much less, and thus it was acceptable. Since each window has 2 minutes and each day has 720 windows, on average, XNBAD took about 0.19 seconds (≪2 minutes) to output the detection results after a window of data was ready. It indicates that XNBAD is able to finish its pipeline on one window immediately before the data of next window arrive.

In the part of the window-level processes, the base host feature extraction and feature normalization were the two most time-consuming ones, with about 4.8 minutes and 2.7 minutes, respectively. More specifically, computing the interactive distribution features and the embedding-induced features $f_3$ and normalizing flow features, respectively, took about 2.5, 1.5, and 2.0 minutes. They took most of the time. On the other hand, both of the feature enhancement and anomaly detection only used about half a minute overall.

Furthermore, we investigated the relationships between the runtime and data scale in a single window. The results of the main and time-consuming processes are illustrated in Figure 8. As shown, most of the processes’ runtime was linear to the data scale (the number of flows/nodes/edges) while the runtime of computing $f_3$ was quadratic to the number of edges, which consists with the time complexity analysis in Section 3.5. The total runtime on a single window was also quadratic to the number of edges, which was mainly due to computing $f_3$. Nevertheless, the maximum total runtime was only about 2.7 seconds when there were more than 1000 edges, which suggests that XNBAD was still efficient on the large data scale. However, according to the tendency, we can foresee that with the data scale continuously growing up to some extreme extent, the total runtime could exceed the window duration, which will make windows of data overstock. To mitigate this problem, we recommend to use simple base host features instead of the complicated ones as much as possible and let the GNN work since the GNN-based enhancement is effective and much faster. Besides, since windows are independent, we can utilize multi-processing to handle different windows in parallel. Moreover, several engineering optimizations can be applied on this situation, but it is beyond the scope of this paper.

7. Conclusions

In this paper, we proposed a novel unsupervised network behavior anomaly detection framework XNBAD which improves detection by describing network behaviors under the dynamic host interaction context. To comprehensively express the behavior semantics, XNBAD integrates the extracted flow features with the GNN-enhanced timely high-order host interaction features as network behavior representation. We carefully studied and refined the publicly available benchmark dataset ISCX-2012 and then conducted experiments on the refined dataset to evaluate the detection performance of XNBAD. The experimental results show that XNBAD is a promising framework. It was able to effectively detect not only the high-volume attack behaviors but also most of the subtle attack behaviors in the dataset, and it outperformed the other three representative baselines with at least 3.8% relative improvements at the 1% significance level on the overall weighted AUC. The results also demonstrate the effectiveness of the host feature enhancement using the GNN and the efficiency of XNBAD.

Future works might include developing unsupervised end-to-end XNBAD variants which learn on the dynamic interaction context via more advanced GNNs [45, 46] and jointly train models to eliminate possible suboptimization and examining XNBAD on other environments like IoT and more advanced and complex datasets. Besides, since behavior detection cannot discover all types of malicious behaviors (as discussed in Section 6.2), making XNBAD collaborate with other detection mechanisms like content detection and host-based detection is also an interesting research direction.

Data Availability

The ISCX-2012 dataset used to support the results of this work is publicly available at https://www.unb.ca/cic/datasets/ids.html. The refined behavior labels used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was supported by the National Key Research and Development Program of China (no. 2018YFB0204301) and the National Natural Science Foundation of China (no. 61472439).

References

[1] D. Desai, “30,000 percent increase in COVID-19-themed attacks,” 2020, https://www.zscaler.com/blogs/security-research/30000-percent-increase-covid-19-themed-attacks.
[2] WHO Reports Fivefold Increase in Cyber Attacks, Urges Vigilance, World Health Organization, Geneva, Switzerland, 2020, https://www.who.int/news/item/23-04-2020-who-reports-fivefold-increase-in-cyber-attacks-urges-vigilance.
[3] M. V. Mahoney and P. K. Chan, Phad: Packet Header Anomaly Detection for Identifying Hostile Network Traffic, Florida Institute of Technology, Tech. Rep., Melbourne, Florida, 2001.
[4] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, “Kitsune: an ensemble of autoencoders for online network intrusion detection,” in Proceedings of the 2018 Network and Distributed System Security Symposium, Diego, California, February 2018.
[5] N. Moustafa, B. Turnbull, and K.-K. R. Choo, “An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things,” IEEE Internet of Things Journal, vol. 6, no. 3, pp. 4815–4830, 2019.
[6] K. Yang, J. Zhang, Y. Xu, and J. Chao, “Ddos attacks detection with autoencoder,” in Proceedings of the IEEE/IFIP Network Operations and Management Symposium, pp. 1–9, Budapest, Hungary, April 2020.
[7] M. Asad, M. Asim, T. Javed, M. O. Beg, H. Mujtaba, and S. Abbasi, “Deepdetect: detection of distributed denial of service attacks using deep learning,” The Computer Journal, vol. 63, no. 7, pp. 983–994, 2020.
[8] E. Min, J. Long, Q. Liu, J. Cui, and W. Chen, “Tr-ids: anomaly-based intrustion detection through text-convolutional neural network and random forest,” Security and Communication Networks, vol. 2018 pages, 2018.
[9] M. Zhu, K. Ye, Y. Wang, and C.-Z. Xu, “A deep learning approach for network anomaly detection based on amf-lstm,” in Proceedings of the IFIP International Conference on Network and Parallel Computing, pp. 137–141, Hohhot, China, August 2018.
[10] Y. Yao, L. Su, Z. Lu, and B. Liu, “Stideepgraph: spatial-temporal deep learning on communication graphs for long-term network attack detection,” in Proceedings of the 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), pp. 120–127, Rotorua, New Zealand, August 2019.
[11] X. Zhuo, J. Zhang, and S. W. Son, “Network intrusion detection using word embeddings,” in Proceedings of the 2017 IEEE International Conference on Big Data (BigData), pp. 4686–4695, Tokyo, Japan, October 2017.
[12] R. S. M. Carrasco and M.-A. Sicilia, “Unsupervised intrusion detection through skip-gram models of network behavior,” Computers & Security, vol. 78, pp. 187–197, 2018.
[13] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” in Proceedings of the 1st International Conference on Learning Representations, Scottsdale, Arizona, USA, May 2013.
[14] M. Ring, A. Dallmann, D. Landes, and A. Hotho, “Ip2vec: learning similarities between ip addresses,” in Proceedings of the 2017 IEEE International Conference on Data Mining Workshops (ICDMW), pp. 657–666, New Orleans, LA, USA, November 2017.
[15] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” in Proceedings of the 2nd International Conference on Learning Representations, Banff, AB, Canada, April 2014.
[16] Y. Zhong, W. Chen, Z. Wang et al., “Helad: a novel network anomaly detection model based on heterogeneous ensemble learning,” Computer Networks, vol. 169, Article ID 107049, 2020.
[17] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.
[18] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, “Characterization of encrypted and vpn traffic using time-related features,” 2nd Internatinal Conference on Information Systems Security and Privacy, vol. 2, pp. 407–414, 2016.
[19] “Cicflowmeter,” https://github.com/ahlashkari/CICFlowMeter.
[20] M. Catillo, M. Rak, and U. Villano, “Discovery of dos attacks through the zed-ids anomaly detector,” Journal of High Speed Networks, vol. 25, no. 4, pp. 349–365, 2019.
[21] A. Moore, D. Zuev, and M. Crogan, Discriminators for Use in Flow-Based Classification, Queen Mary University of London, Department of Computer Science, Tech. Rep., London, England, 2013.
[22] B. Anderson and D. McGrew, “Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity,” in Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1723–1732, Halifax, NS, Canada, August 2017.
[23] B. Anderson, A. Chi, S. Dunlop, and D. McGrew, “Limitless http in an https world: inferring the semantics of the https protocol without decryption,” in Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, pp. 267–278, Richardson, TX, USA, March 2019.
[24] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, “Fs-net: a flow sequence network for encrypted traffic classification,” in Proceedings of the IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, pp. 1171–1179, Paris, France, April 2019.
[25] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martinez-del-Rincon, and D. Siracusa, “Lucid: a practical, lightweight deep learning solution for ddos attack detection,” IEEE Transactions on Network and Service Management, vol. 17, no. 2, pp. 876–889, 2020.
[26] X. Han, R. Yin, Z. Lu et al., “Stidm: a spatial and temporal aware intrusion detection model,” in Proceedings of the 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 370–377, Guangzhou, China, November 2020.
[27] M. A. Ferrag and L. Maglaras, “Deepcoin: a novel deep learning and blockchain-based energy exchange framework for smart grids,” IEEE Transactions on Engineering Management, vol. 67, no. 4, pp. 1285–1297, 2020.
[28] S. Gamage and J. Samarabandu, “Deep learning methods in network intrusion detection: a survey and an objective comparison,” Journal of Network and Computer Applications, vol. 169, Article ID 102767, 2020.
[29] W. Wang, S. Jian, Y. Tan, Q. Wu, and C. Huang, “Repre-resentation learning-based network intrusion detection system by capturing explicit and implicit feature interactions,” Computers & Security, vol. 112, Article ID 102537, 2022.
[30] T. Zimmermann and N. Nagappan, “Predicting defects using network analysis on dependency graphs,” in Proceedings of the 30th international conference on Software engineering, pp. 531–540, Leipzig Germany, 2008.
[31] L. Akoglu, M. McGlohon, and C. Faloutsos, “Oddball: spotting anomalies in weighted graphs,” Advances in Knowledge Discovery and Data Mining, vol. 6119, pp. 410–421, 2010.
[32] A. D. Kent, L. M. Liebrock, and J. C. Neil, “Authentication graphs: analyzing user behavior within an enterprise network,” Computers & Security, vol. 48, pp. 150–166, 2015.
[33] A. Gamachchi, L. Sun, and S. Boztaş, “Graph based framework for malicious insider threat detection,” in Proceedings of the 50th Hawaii International Conference on System Sciences, pp. 2638–2647, Hilton Waikoloa Village, Hawaii, USA, January 2017.
[34] H. Chen, B. Perozzi, R. Al-Rfou, and S. Skiena, “A tutorial on network embeddings,” 2018, https://arxiv.org/abs/1808.02590.
[35] A. Grover and J. Leskovec, “node2vec: scalable feature learning for networks,” in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 855–864, San Francisco, CA, August 2016.
[36] R. Wei, L. Cai, A. Yu, and D. Meng, “Age: authentication graph embedding for detecting anomalous login activities,” in Proceedings of the International Conference on Information and Communications Security, pp. 341–356, Beijing, China, December 2019.
[37] S. Zhao, R. Wei, L. Cai, A. Yu, and D. Meng, “Ctldm: continuous-temporal lateral movement detection using graph
embedding,” in *Proceedings of the International Conference on Information and Communications Security*, pp. 181–196, Beijing, China, December 2019.

[38] W. L. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” *Advances in Neural Information Processing Systems*, vol. 30, pp. 1024–1034, 2017.

[39] M. D. Vose, “A linear algorithm for generating random numbers with a given distribution,” *IEEE Transactions on Software Engineering*, vol. 17, no. 9, pp. 972–975, 1991.

[40] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, “Toward developing a systematic approach to generate benchmark datasets for intrusion detection,” *Computers & Security*, vol. 31, no. 3, pp. 357–374, 2012.

[41] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization,” in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, pp. 108–116, Madeira, Portugal, January 2018.

[42] J. Jin Huang and C. X. Ling, “Using auc and accuracy in evaluating learning algorithms,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.

[43] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, vol. 15, pp. 315–323, 2011.

[44] D. P. Kingma and J. Ba, “Adam: a method for stochastic optimization,” in *Proceedings of the 3rd International Conference on Learning Representations*, San Diego, CA, USA, May 2015.

[45] Y. Zhao, J. Qi, Q. Liu, and R. Zhang, “Wgcn: graph convolutional networks with weighted structural features,” in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 624–633, Montréal, Canada, July 2021.

[46] Y. Wang, Y.-Y. Chang, Y. Liu, J. Leskovec, and P. Li, “Inductive representation learning in temporal networks via causal anonymous walks,” in *Proceedings of the 9th International Conference on Learning Representations*, Vienna, Austria, May 2021.