The Case for Learned Provenance-based System Behavior Baseline

Yao Zhu [*1] **Zhenyuan Li** [*1 2] **Yangyang Wei** [2] **Shouling Ji** [1]

Abstract

Provenance graphs describe data flows and causal dependencies of host activities, enabling to track the data propagation and manipulation throughout the systems, which provide a foundation for intrusion detection. However, these Provenance-based Intrusion Detection Systems (PIDSes) face significant challenges in storage, representation, and analysis, which impede the efficacy of machine learning models such as Graph Neural Networks (GNNs) in processing and learning from these graphs. This paper presents a novel learning-based anomaly detection method designed to efficiently embed and analyze large-scale provenance graphs. Our approach integrates dynamic graph processing with adaptive encoding, facilitating compact embeddings that effectively address out-of-vocabulary (OOV) elements and adapt to normality shifts in dynamic real-world environments. Subsequently, we incorporate this refined baseline into a tag-propagation framework for real-time detection. Our evaluation demonstrates the method's accuracy and adaptability in anomaly path mining, significantly advancing the state-of-the-art in handling and analyzing provenance graphs for anomaly detection.

1. Introduction

Recently, Provenance-based Intrusion Detection Systems (PIDSes) that utilize provenance graphs for threat modeling gain widespread attention from both industry and academia (Rehman et al., 2024; Cheng et al., 2023). Originating from system audit logs, provenance graphs capture detailed behaviors within operating systems through causally linked relationships (Gehani & Tariq, 2012). Provenance graphs model system activities by representing components as nodes and their interactions as edges, for example, `PROCESS A.exe READ FILE B.txt`. These edges are further enriched with temporal data, enhancing the graph's ability to chronologically trace system actions. Crucially, the inherent causal structure of these graphs ensures that only prior actions can affect subsequent behaviors, providing a robust framework for analyzing and predicting system events.

Intrusion detection requires immediate responses due to the urgent nature of the task. While provenance graphs offer considerable benefits for security analysis, they face several challenges that impair their effectiveness (Alahmadi et al., 2022; Li et al., 2021). The constant growth of system logs generates vast and intricate provenance graphs, featuring many nodes and edges, along with a high average node degree. This complexity renders the graphs computationally intensive to manage and analyze. Additionally, although the rich semantic and contextual information within these graphs aids in understanding system behaviors, it also significantly increases computational overhead (Wang et al., 2022). Moreover, system logs are typically gathered continuously, incorporating new entities and relationships frequently. This dynamic nature often results in an out-of-vocabulary (OOV) problem during event encoding, which affects the accuracy and effectiveness of the graph representations and presents adaptability challenges.

To address these shortcomings, we propose a novel learning-based anomaly detection approach that more effectively captures system behaviors from provenance graphs. Anomaly detection in dynamic graphs is a crucial aspect of network analysis, especially in the intrusion detection scenario (Zeng et al., 2022). Existing frequency databases extract and record the frequency of each system event from logs, allowing a decoupled approach to handling provenance graphs (Hassan et al., 2019). Inspired by these databases, as well as learned storage techniques that optimize the storage and retrieval of large-scale graph data, our method begins by decoupling provenance graphs into individual system events along with their frequencies.

We employ an embedding model to convert these events into vectors, which serve as inputs for a lightweight learning model. This model is trained to recognize patterns based on the frequency of occurrences, addressing the dynamic

[*]Equal contribution [1]College of Computer Science and Technology, Zhejiang University, Hangzhou, China [2]College of Software Technology, Zhejiang University, Ningbo, China. Correspondence to: Zhenyuan Li <lizhenyuan@zju.edu.cn>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

nature of system behaviors. As host behaviors evolve, the challenge of normality shifts, where previously learned behaviors may not accurately classify new benign or malicious events, becomes evident. Our approach includes a mechanism specifically designed to manage the frequent introduction of new entities and relationships in the graphs, effectively tackling the out-of-vocabulary and normality shift challenges. By learning the distribution of vector frequencies that describe system behaviors, we establish a baseline for normal system activity. This baseline is integrated into a tag-propagation framework (Li et al., 2024a;b) for real-time anomaly detection. Our evaluations demonstrate the accuracy and adaptability of our method in anomaly path mining, showcasing significant improvements in handling and analyzing provenance graphs for intrusion detection.

We have developed a prototype of our approach[1] that features an adaptive embedding method to address out-of-vocabulary elements and normality shifts in dynamic environments. This method preserves the essential semantic and contextual information in the provenance graphs and incorporates a lightweight machine learning model designed to efficiently learn behavioral patterns from large-scale provenance graph data in a native streaming format. We utilized this model to establish a baseline for systems' normal behaviors, which we then integrated with a tag-propagation framework to facilitate real-time detection. Our comprehensive evaluation, conducted on large-scale and open-source datasets, confirms the effectiveness and efficiency of our provenance graph embedding method. The results highlight its accuracy and adaptability in real-time anomaly path mining tasks, demonstrating its potential to significantly enhance anomaly detection capabilities.

2. Provenance-based Intrusion Detection

2.1. Provenance Graph Definition

Provenance graphs describe the history of an operating system's execution (Gehani & Tariq, 2012), where nodes represent system's entities, and edges capture the interactions between adjacent nodes. A provenance graph can be defined as $G =< V, R >$, where V represents the set of nodes in G and R is the set of edges representing the relationship between neighbor nodes. An edge $(u, v) \in R$ exists between two entities u and v ($u, v \in V$) when their interactions are logged. The provenance graph is composed of a sequence of events, each representing an interaction between a pair of adjacent nodes and the edge connecting them. Formally, the event streams can be aggregated into a streaming provenance graph, represented as:

$$G = \{(s, r, d, t) : s, d \in V, r \in R, t \in \mathbb{R}^+\}, \quad (1)$$

[1]Available at https://github.com/AddoZhu/behavior_baseline

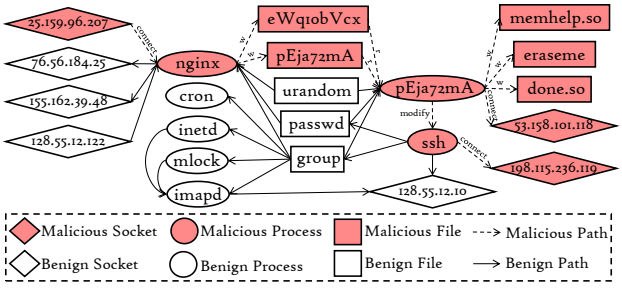


Figure 1. Nginx Backdoor Attack in Provenance Graph

Table 1. Event Definitions

SRC(s)	DEST(d)	OPERATION(o)
Process	File	Write File; Create File
File	Process	Read File; Process Load
Process	Process	Start Process; End Process
Process	Network	Send Message
Network	Process	Receive Message

where (s, r, d, t) denotes an edge r from node s to d at time t, reflecting the causality direction. Table 1 lists detailed event flow mappings. Each event $e = (s, r, d, t) \in \mathcal{E}$ adds to the dynamic representation of the system, capturing the evolving nature of the provenance graph over time.

Figure 1 shows a local provenance graph from DARPA E3-CADETS[2] that describes nginx backdoor attack activity (program, 2020). The attacker compromises a `Nginx` web server by downloading a malicious payload and executing to gain root privileges. The ovals, diamonds, and rectangles represent provesses, sockets, and files, respectively. To emphasize malicious behaviors, we highlight the nodes representing attack behaviors in red and depict the edges associated with these activities as dashed lines, while gray nodes and solid edges represent normal behaviors.

2.2. Frequence-based Anomaly Detection

Some anomaly detection approaches (Hassan et al., 2019; Xie et al., 2018) build event frequency databases (also known as baseline databases) to assess the suspiciousness of individual events and paths for detection and investigation. The frequency database stores the occurrence frequency of individual events in historical logs, where a higher frequency indicates that the event is more likely to be normal. We assign a regular score (Γ_e) to each event, representing the degree of normality of an event, as following:

$$\Gamma_e = \frac{\textit{Frequency of e(s, r, d)}}{\textit{Frequency of e(s, r, *)}} \quad (2)$$

[2]Transparent-Computing.https://github.com/darpa-i2o/Transparent-Computing

where *Frequency of e(s, r, d)* represents the number of the occurrence of event e in the historical records where the triple (s, r, d) is identical, and *Frequency of e(s, r, *)* represents the number of occurences of events that share the same source and relationship. The regular score (Γ_e) indicates the probability of a specific event occurring. If event e has never occurred, its regular score is 0.

We illustrate an example of an Nginx backdoor attack in Figure 1, a behavior that has never been observed in the system logs. Since this attack has not occurred in historical logs, it is absent from the frequency database, leading to attack-related events, such as `Network 25.159.96.207 connect to Process Nginx`, having very low regular scores close to zero. In contrast, benign events similar to this behavior, such as `Process nginx connect to Network 78.205.235.65`, may have appeared in historical logs, resulting in high regular scores. However, if such a benign event is absent from the frequency database, this method becomes ineffective, highlighting its limited adaptability.

As the system logs accumulate, the methods relying on the baseline database struggle to adapt to the normality shift problem, losing adaptability over time. Meanwhile, the accumulation of a large number of historical events in the databases results in memory storage pressure and query overhead.

2.3. Online Detection with Tag-propagation Framework

In order to handle the continuously generated event stream, several works (Hossain et al., 2017; Milajerdi et al., 2019) adopt tag-propagation strategies to enhance the real-time capabilities of detection systems and their proficiency in handling streaming data. These strategies utilize tags to cache computation results and propagate them through the provenance graph based on causal attributes, ensuring that each event is checked only once. Therefore, tag-propagation-based methods can achieve real-time detection with high accuracy, minimizing redundant processing, and significantly reducing computational overhead. However, these methods with relatively simplified detection logic always struggle to adapt to complex and novel attacks.

2.4. Practical Challenges

Despite their advances, existing PIDSes exhibit several limitations that impact their effectiveness in real-world applications:

C1. **Real-time Detection.** Many existing systems (Hassan et al., 2019; Rehman et al., 2024) struggle with real-time detection, leading to significant delays in generating alerts. These delays can be critical for preventing or mitigating ongoing cyber attacks (Li et al., 2021).

C2. **Graph Size and Complexity.** Provenance graphs can become exceedingly large due to the complexity of modern software systems and the sophisticated nature of attacks such as Advanced Persistent Threats (APTs) (Ding et al., 2023). Managing such large graphs requires substantial memory and can force compromises between computational resources and the granularity of detection.

C3. **Temporal Information.** Other methods (Zeng et al., 2022; Wang et al., 2022) fail to fully leverage temporal information within system logs, resulting in suboptimal performance. Capturing and utilizing temporal patterns is essential for accurately identifying and understanding attack progressions.

C4. **Coarse-grained Detection and Interpretability.** Some approaches (Han et al., 2020; Wang et al., 2020; Manzoor et al., 2016) use coarse-grained detection strategies that may efficiently process data, but often produce alerts lacking sufficient detail for effective analysis, compromising their practical utility.

C5. **Adaptation to Unseen Benign Behaviors.** Entities in provenance graphs, such as process names, file paths, and IP addresses, contain crucial semantic information for detection (Ding et al., 2023). The dynamic nature of system environments, where new entities and relationships frequently emerge, calls for adaptive embedding strategies to maintain accurate detection capabilities. Standard anomaly detection methods establish a baseline from historical data, but often fail to recognize new benign behaviors, leading to false positives. While Graph Neural Networks (GNNs) offer a potential solution by analyzing rich semantic contexts and complex relationships, their high computational demands limit the broader deployment.

To sovle C2 and C5, we employ a learned storage method, utilizing a lightweight neural network model to capture the distribution of system behaviors in historical logs, enabling efficient management of large and complex provenance graphs. Moreover, this embedding-based learning model possesses the generalization ability to alleviate the out-of-vocabulary (OOV) issue arising from newly introduced entities and relationships, as well as the normality shift induced by host behavior changes.

After that, we integrate this model with the tag-propagation algorithm to process graph streams and enable real-time detection. Under this model, the causal order and temporal information between events are inherently captured through the tag propagation process, solving C1 and C3. Meanwhile, the anomaly scores of consecutive events are propagated and aggregated by the tag-propagation algorithm to derive a path-level anomaly score, which facilitates real-time alerts

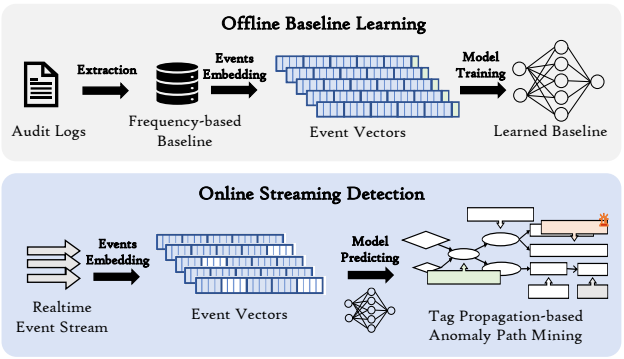


Figure 2. Approach Overview

while producing interpretable attack paths, thus addressing C4.

3. Method

This section outlines the architecture of the provenance graph embedding method, representing the system behavior, and integrating with the tag-propagation algorithm to conduct online streaming detection based on provenance graphs. As shown in the overview of Figure 2, our method consists of two main phases: 1) an offline baseline learning phase, where a lightweight neural network model learns benign behavior patterns based on event frequencies to establish a learned baseline; and 2) an online streaming detection phase, which integrates the learned baseline with the tag-propagation algorithm to mine anomaly paths within real-time streams.

3.1. Events Embedding

In the preprocessing of historical audit logs, we refine attributes such as process names, file paths, network IP addresses, and interactions between entities, as illustrated in Table 1. Each event e is represented as a triple (s, r, d), and its regular score Γ_e is computed using Equation 2. These events and scores are then aggregated to construct a frequency-based baseline. Given an event sequence $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$, the goal is to process each event sequentially and encode it into a fixed-dimensional vector representation. This process enables downstream tasks such as graph representation learning, establishing a learned baseline, and anomaly detection. In addition to encoding models, there are specific challenges and potential optimization methods for event encoding in provenance graphs. We have designed the following three mechanisms to address these issues.

Adaptive Embedding To address the out-of-vocabulary (OOV) problem, we encode unrecognized entities, which are not present in the training corpus of the embedding models, as zero vectors. Formally, to encode an event $e_i = \{w_{i1}, w_{i2}, ..., w_{ij}\}$, each word w_{ik} is represented based on its presence in the system audit logs L. The event e_i's vector $\mathbf{v}_{e_i}$ is as follow shows:

$$\mathbf{v}_{e_i} = \begin{cases} \textbf{Embed\_s}[e_i], & \text{if } e_i \in L, \\ \bigoplus_{w \in e_i} \mathbf{v}_w \begin{cases} \textbf{Embed\_w}[w], & \text{if } w \in L, \\ \text{vector}(0), & \text{otherwise}, \end{cases} & \text{if } e_i \notin L. \end{cases} \tag{3}$$

As Doc2Vec and FastText are capable of handling OOV words through specific encoding mechanisms. Their mathematical formulations and underlying principles are presented in Appendix B.2. In our experiments, we compared these models with the adaptive encoding method.

Doc2Vec can use `infer_vector()` function to encode documents that do not appear in the training set. We treat an event as a document, which is the unit processed by Doc2Vec. The inference process optimizes the event vector $\mathbf{v}_e$ to maximize its compatibility with the model's trained parameters, given the context of the words in e_i. Formally, this can be expressed as:

$$\mathbf{v}_{e_i} = \arg\max_{\mathbf{v}} \prod_{t \in e_i} P(t|\mathbf{v}, \Theta) \tag{4}$$

where t represents tokens in the event e_i and $P(t|\mathbf{v}, \Theta)$ is the probability of token t given the embedding $\mathbf{v}$ and the model parameters Θ. The optimization is typically performed through a few iterations of the Stochastic Gradient Descent (SGD) algorithm, starting from a random vector $\mathbf{v}$.

As a subword-level embedding model, FastText could encode each word into a targeted length vector using the subwords. Each word w is decomposed into a set of subwords $S_w = \{s_1, s_2, ..., s_n\}$, where each subword is a substring of the word w. If $w \notin L$, its embedding vector $\mathbf{v}_w$ is computed as the average embeddings of its constituent subwords. Formally, this process can be expressed as:

$$\mathbf{v}_w = \frac{1}{|S_w|} \sum_{s \in S_w} \mathbf{v}_s \tag{5}$$

where $|S_w|$ is the cardinality of the subword set S_w. After obtaining the vector for each word in an event, we concatenate them sequentially to form the completed vector representation of the event sentence.

Weight Embedding Long file paths may blur the distinction between vector representations of different regular score events. For example, the file path `/home/user/.../benign` and the file path `/home/user/.../malicious` would be mapped to two very similar vectors, even though the regular score of the benign file is much higher than that of the malicious file.

This makes it difficult for the subsequent learning model to capture the distinguishing data features. So we assign different weights to each directory level, with directories closer to the file name receiving higher weights. The embedding of the file path is then calculated as the weighted sum of its directory levels. Formally, the embedding vector of a file name is defined as:

$$\mathbf{v}_{File\_name} = \frac{1}{\sum w_i} \sum w_i \cdot \mathbf{v}_i \tag{6}$$

where $\mathbf{v}_{File\_name}$ represents the embedding vector of the i-th element in the file path separated by a slash, and w_i is the weight. The weight w_i of the i-th element in a file path is defined as $w_i = \log \frac{n+1}{n-i}$, where n is the number of all elements in a file path. This weighting scheme is designed to reduce the influence of long file paths on the vectorization of file names, prioritizing the significance of directory levels closer to the file name.

One-hot Embedding Intuitively, higher-dimensional vectors can encode more information, allowing the subsequent graph representation learning model to capture richer features. As the categories of relationships and entities are limited, as shown in Table 1, we can use one-hot encoding to represent the categories of entities and relationships, allowing more dimensions within the same vector sapce to be allocated for representing more critical features, such as process names and network IPs. To optimize storage space, we can adopt a 3-dimensional one-hot vector for the categories of entities and an 8-dimensional one-hot vector for the types of relationships between entities. This approach reduces the dimensionality of individual event vectors and mitigates memory overhead. However, encoding these features as (1, 0, 0) can lead to conflicts with zero vectors used to encode OOV words, potentially reducing the predictive performance of the model. To avoid this issue, we adopt a variant representation such as (0.5, -0.5, -0.5) for one-hot encoding.

$$\mathbf{v}_{e_i} = \text{one-hot}_c(s) + \mathbf{v}_s + \text{one-hot}_r(r) + \text{one-hot}_c(d) + \mathbf{v}_d \tag{7}$$

Despite this adjustment, one-hot vectors lack the capacity to capture meaningful information about relationships between entities, which is critical for understanding their structure and properties.

3.2. Offline Baseline Learning

The task of learning the distribution of benign behaviors in provenance graphs is modeled as a frequency-based regression problem, where the goal is to predict the regular score of each event, which reflects the degree of abnormality. To alleviate memory pressure and efficiently process real-time log streams, we employ a lightweight regression model that functions as a learned baseline and facilitates real-time detection by integrating with the tag-propagation algorithm. So we select three popular regression models, namely, the Multilayer Perceptron (MLP) model, the Long Short-Term Memory (LSTM) model, and the Convolutional Neural Network (CNN) model.

In real-world scenarios, the model encounters unobserved events categorized as benign or malicious. Malicious events often feature unique entities and relationships, distinguishing them from benign behaviors. Due to the models' lack of exposure to malicious events during training, it frequently misclassifies such events. To mitigate this issure, we construct relevant negative samples by replacing subjects or objects in benign events with uncommon entities. For example, an uncommon network entity `<10.1.1.2>` replaces the common IP address in the benign event `<128.55.12.122, network connect, nginx>`. Similarly, uncommon file paths and process names are used to create diverse negative samples. Then we assign regular scores to every negative samples, as following:

$$S_{negative} = f(\Delta_{event}) \tag{8}$$

where Δ_{event} denotes the difference in event letters between the negative samples and their corresponding benign counterparts. By incorporating such artificially constructed negative events, the model is better equipped to differentiate between benign and malicious events, thereby enhancing its predictive accuracy.

The encoding model maps an event sequence $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$ into a series of vectors $\mathbf{V}_\mathcal{E} = \{\mathbf{v}_{e_1}, \mathbf{v}_{e_2}, \ldots, \mathbf{v}_{e_n}\}$, where each event e_i is represented by a vector $\mathbf{v}_{e_i} \in \mathbb{R}^d$, with d denoting the dimensionality of the vector representation. Each event corresponds to a regular score $y_i \in \mathbb{R}$, and the model outputs a predicted score $\hat{y}_i \in \mathbb{R}$. When applying three different models, we only add or modify specific layers unique to each model to flexibly construct the respective architectures. We parameterize a function f_θ using a model, which maps the input vector to the predicted score:

$$\hat{y}_i = f_\theta(\mathbf{v}_{e_i}) \tag{9}$$

To ensure that the predicted scores $\hat{y}_i$ closely match the true regular scores y_i, we adopt the Mean Squared Error (MSE), one of the most commonly used loss functions in regression tasks. The loss function $\mathcal{L}_\theta$is defined as:

$$\mathcal{L}_\theta = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{10}$$

Using backpropagation, the model computes the gradient of the loss function with respect to the model parameters. The gradient of the loss with respect to the parameters of layer l

is computed as:

$$\nabla_{\theta^{(l)}}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \cdot \prod_{k=L}^{l+1} \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{h}^{(k-1)}} \cdot \frac{\partial \mathbf{h}^{(l)}}{\partial \theta^{(l)}} \tag{11}$$

where $\mathcal{L}$ denotes the total number of layers in the neural network, i.e., the index of the output layer. And then the model updates the parameters of the l-th layer $\theta^{(l)}$ via gradient descent to minimize the loss:

$$\theta^{(l)} \leftarrow \theta^{(l)} - \eta \cdot \nabla_{\theta^{(l)}}\mathcal{L} \tag{12}$$

where η is the learning rate.

As the regular scores of events range in $(0, 1)$, we use the sigmoid function in the output layer to limit the prediction output of the model. The sigmoid function tacks its value in (0,1) and reads

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{13}$$

where x presents the input of the output layer.

3.3. Online Anomaly Path Mining

We further conduct an anomaly path mining task, leveraging the embeddings learned from provenance graphs generated from benign system logs to establish a learned baseline for normal system behaviors. This allows us to evaluate the effectiveness of our method for embedding provenance graphs in practical scenarios.

Given each newly observed system event, the baseline model effectively and adaptively assesses the normality of the event, initializing a continuous regular score ranging from 0 to 1. Higher scores indicate more normal events, while lower scores suggest anomalies. Intuitively, while a single infrequent event might be coincidental, a sequence of such events is more likely indicative of an attack. Therefore, we employ the tag-propagation algorithm to aggregate tags of causally related events, deriving the regular score of the path (Γ_P), as the following shows:

$$\Gamma_P = \frac{1}{\alpha} \prod_{e \in P} (\Gamma_e \cdot \alpha) \tag{14}$$

where α represents a decay factor, and Γ_e is generated by Equation 2. Since Γ_e ranges from 0 to 1, simply multiplying them cumulatively means that Γ_P will monotonically decrease, eventually falling below a specific threshold. The decay factor (α) (Hossain et al., 2020) can cause the normal score to increase when there are no anomalous events. The tag-propagation framework consists of four main stages: tag initialization, propagation, removal, and alert triggering, tailored for anomaly path mining, to balance overhead and analytical capabilities. The detailed process of the tag-propagation framework is provided in the Appendix C. While reducing caching overhead and computational resource, it also ensures that alerts are triggered immediately once threshold-reaching events are processed, effectively shortening the system's response time.

We initialize the tags using our learned baseline model to predict the frequency of each event. We analyze each event in the log stream. Firstly, each event e_i is encoded into a vector representation $\mathbf{v}_{e_i}$. The trained machine learning model f_θ is then applied to compute the predicted regular score $\hat{y}_i$. We use $\hat{y}_i$ as the initial tag. In order to avoid excessive tags accumulation, only when the regular score of the event (Γ_e) is lower than the preset threshold and the source node does not have a tag, a new tag will be assined and stored in the destination node. The regular score of the event Γ_e propagates along the provenance graphs together with the initialized tags, resulting in the regular score of the path Γ_P according to Equation 14. An alert will be triggered when Γ_P decreases below a predefined threshold during propagation.

4. Experiments

4.1. Experimentanl Setting

We utilize the TensorFlow library to construct neural network models, enabling flexible modifications of layer configurations to implement various architectures such as MLP, LSTM and CNN. This approach facilitates the evaluation of different machine learning models in the provenance graph embedding and anomaly path mining task. Here, we evaluated the efficiency of various encoding and regression models, the accuracy of regression models in predicting regular scores, the effectiveness of our approach in handling the real-time anomaly path mining task, and a comparison with state-of-the-art provenance-based intrusion detection systems. We conducted all experiments on an Ubuntu 18.04.6 LTS server with an Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz, 251GiB of memory, and three NVIDIA GeForce RTX 3090 GPUs.

Datasets In our experiments, we utilized datasets from the DARPA Transparent Computing (TC) dataset (tra, 2015.2), which contains millions of benign and hundreds of malicious events collected from platforms with diverse background activities, providing provenance-rich data capturing system events and dependencies over time. This dataset includes a series of realistically simulated Advanced Persisitent Threats (APT), such as malware execution, privilege escalation, remote exploitation, and data exfiltration. We primarily use the E3-CADETS dataset, constructing a training dataset with 1,042k system events and a testing dataset with 26k events. Additionally, we demonstrate the adaptability of our method on other dataests in Appendix D.3.

We categorize event encoding methods into two main types: one that encodes out-of-vocabulary (OOV) words as zero

Table 2. Time of Encoding

Embedding Models	Training Set (s)	Testing Set (s)
Word2Vec*	28.84	6.06
Doc2Vec*	30.78	6.41
Doc2Vec	488.24	122.23
FastText*	43.07	9.46
FastText	52.17	13.43
FastText(weight)*	85.42	20.62
FastText(weight)	177.91	44.21
FastText(one-hot)*	14.2	3.16
FastText(one-hot)	43.41	10.53
TinyBERT	5152.08	1338.71

vectors and another that does not apply special handling for OOV words. Since Word2Vec throws an exception when encoding OOV words, it can only be used for the first type. In contrast, the training corpus for TinyBERT is unknown, so it's only applicable to the second type. Among all the encoding models that we employed, FastText demonstrated the best performance. Therefore, we further conducted experiments using FastText in combination with weight encoding and one-hot encoding to compare their effectiveness. In addition to the ablation experiments, we trained encoding models on system logs and mapped each event to a 500-dimensional vector. Apart from using the one-hot method to encode each event into a 214-dimensional vector, TinyBERT treats each event as a sentence and encodes it into a 312-dimensional vector.

In training the baseline model for the regression task, we use the TensorFlow library to modify the regression models by adjusting the number of layers. Through controlled variable experiments, we ultimately employed the Adam optimizer with a learning rate of 0.001, and 200 training epochs for each model configuration. we evaluated the learned model's performance based on the prediction accuracy of event regular scores. A prediction is considered true if the difference between the predicted regular score and the true score is less than a threshold of 0.2, which is selected because the frequencies of negative saples we constructed are generally below this value. For the anomaly path mining task, we use path-level precision, recall, and F1 score as evaluation metrics. In the comparative experiments, due to the varying detection granularity across methods, we adopt node-level metrics for consistency.

4.2. Results

Events Embedding For the two types of encoding methods, encoding OOV words as zero vectors and not applying special handling for OOV words, we use the asterisk (*) symbol to distinguish between them. Encoders marked with this symbol indicate the first encoding method.

Table 3. Average Training Time of Models

Learning Models	Training Time (s)	Testing Time (s)
MLP	276.94	11.35
LSTM	456.11	20.41
CNN	2601.59	27.21

Table 2 presents the event encoding times for different encoding models and methods. Similar results were obtained on other datasets, so we only display the results on the CADETS dataset as an example, showing in Table 8. We notice that if we directly encode the OOV words with zero vectors, it will accelerate the embedding process. In contrast, encoding models tend to speed more time handling unknown words. For instance, in Doc2Vec, inferring the vector for an unseen event takes significantly more time compared to directly using pre-trained event vectors. In addition, we find that if we use one-hot encoding method, it will drastically reduce the encoding time. But we must remember that it only uses 214 dimensions to represent an event, while other methods use 500 dimensions (except TinyBERT, which uses 312 dimensions).

The encoding time of TinyBERT is significantly longer compared to other models, as they operate on entirely different scales of computational complexity. TinyBERT is based on the Transformer architexture, a deep neural network where every input token interacts with all other tokens in events. This results in substantial computational overhead during the encoding process. In contrast, other models such as Word2Vec are shallow neural networks with much lower computational complexity than TinyBERT. Overall, except for TinyBERT, the encoding time for each event using the other encoding models is in the microsecond range, which meets the requirements for processing real-time stream graphs and providing an immediate response. Due to the inherent characteristics of provenance graphs and the real-time requirements of intrusion detection, complex encoding models are not suitable.

Frequency Prediction We evaluated the regression performance of the baseline model using the model's prediction of the regular scores of events. We analyzed the prediction results on the testing dataset (Accuracy1(%)) and for malicious events not present in the frequency dtabase (Accuracy2(%)), indicating that regular scores of these events are expected to be zero. Since we found that the event encoding methods have a significant impact on provenance graph embedding performance, while the effect of the learning models is relatively small, Table 4 presents the mean and standard deviations of the prediction for different encoding models and methods. More detailed results are provided in the Appendix D. Similarly, the training and testing time of the models was only dependent on the dimensionality of

Table 4. Detection Accuracy of Single Events and Paths

Embedding Models	Provenance Graph Embedding		Anomaly Path Mining		
	Accuracy1(%)	Accuracy2(%)	Precision	Recall	F1 Score
Word2Vec*	0.8886±0.0001	1.0000±0.0000	0.9545±0.0643	0.6852±0.1593	0.7809±0.0865
Doc2Vec*	0.9055±0.0058	1.0000±0.0000	0.9772±0.0004	0.7778±0.1571	0.8568±0.1050
FastText*	**0.9058±0.0058**	**1.0000±0.0000**	**0.9927±0.0052**	**1.0000±0.0000**	**0.9963±0.0026**
FastText(weight)*	0.9057±0.0017	0.5714±0.0117	0.9464±0.0758	0.3518±0.0262	0.5124±0.0350
FastText(onehot)*	0.8661±0.0195	0.8809±0.1279	0.8302±0.1322	0.5556±0.3175	0.6275±0.2331
Doc2Vec	0.6121±0.0992	0.2905±0.0135	0.0953±0.0710	0.1297±0.0524	0.0867±0.0657
FastText	0.9049±0.0005	0.5952±0.0828	0.0467±0.0334	0.4259±0.2579	0.0694±0.0437
FastText(weight)	**0.9057±0.0003**	**0.5619±0.0067**	**0.8493±0.2025**	**0.3333±0.0000**	**0.4716±0.3357**
FastText(onehot)	0.9054±0.0002	0.5619±0.0243	0.4741±0.2373	0.2593±0.2283	0.2734±0.2001
TinyBERT	0.9038±0.0014	0.1143±0.0117	0.6752±0.4591	0.0556±0.0485	0.0785±0.0871

the events. Table 3 presents the average training and testing time when each event is encoded into a 500-dimensional vector.

In general, special handling of OOV words by mapping them to zero vectors can improve the prediction performance of models, among which FastText demonstrated the best performance in this case. Since our processing unit is an event, where each event consists of two entities and the interaction between them (i.e., adjacent nodes and the connecting edge in the provenance graph), malicious events typically contain a higher proportion of OOV elements. In contrast, unseen benign events tend to include fewer such elements. As a result, the model assigns lower regular scores to the former, enabling it to distinguish actual malicious behavior from unseen benign behavior, thereby reducing false positives. Therefore, for models that do not incorporate explicit handling of OOV words (as indicated by rows without asterisks in Table 4), although they achieve comparable performance on the overall testing set (*Accuracy1%* column), their predictions for actual malicious events are significantly less effective (*Accuracy2%* column). Meanwhile, prediction performance using the one-hot encoding method is the poorest. The reason is that one-hot encoding fails to capture the inherent meaning of the categories of entities and relationships, resulting in a loss of semantic information during the encoding process.

In the absence of special handling for OOV words, the method of assigning weights to file paths can, to some extent, enhance the model's prediction performance. For a long file path, if a benign path and a malicious path differ only in the final file name while sharing identical directories in the preceding segments, the two paths will be encoded into vectors that are relatively close to each other. By assigning greater weights to the directories towards the end of the path, this problem can be mitigated to a certain extent, thereby improving the model's prediction performance.

As Table 3 shows, the training time for the CNN model is much longer than other two models, and for events encoded into the same dimensionality (except for the TinyBERT model and the one-hot embedding method), the testing time of three models increases. Moreover, reducing the dimensionality of event embeddings does not necessarily reduce training time. This is because lower-dimensional embeddings may lose some semantic information, making it more difficult for the model to capture data features and requiring more epochs to converge.

Anomaly Detection Table 4 presents the experimental results of integrating the learned baseline model with the tag-propagation algorithm for the anomaly path mining task in real-time log streams. Our method enables the identification of anomalous paths and the generation of real-time alerts, thereby further reducing false positives.

In general, handling OOV words as zero vectors can significantly improve detection performance. This observation is closely related to the regression performance of the baseline model. The model predicts the regular score for each event and uses them as initial tags, which are subsequently propagated and aggregated following the information flow in the provenance graph. Among the evaluated methods, using the FastText model for events embedding and mapping OOV words to zero vectors performed best. The detection results achieved an F1 score that exceeded 99%.

The encoding methods that do not handle OOV words specifically, compared to the above methods, show little difference in the provenance graph embedding phase. However, the distinction between the vectors of malicious and benign events is smaller, making it more difficult to detect malicious paths during the detection phase and leading to a higher rate of false positives.

Although *TinyBERT* uses Transformer architecture and surpasses traditional Natural Language Process (NLP) models in contextual modeling capability, the detection performance of using TinyBERT as the event encoder is suboptimal. This is due to the long-term and distributed nature of network attacks necessitates a broader analytical scope beyond a

Table 5. Node-Level Comparison with SOTA PIDSes

E3-CADETS				
	# of TPs	# of FNs	# of FPs	F1 score
Ours	32	1	69	0.4778
Nodoze	31	2	667	0.0885
ProvDetector	20	13	77	0.3636
Flash	7	25	6996	0.0019
Karios	23	10	119878	0.0003
E3-THEIA				
	# of TPs	# of FNs	# of FPs	F1 score
Ours	12	5	46	0.3200
Nodoze	12	5	105	0.1791
ProvDetector	0	17	91	0.0000
Flash	2	15	23330	0.0002
Karios	11	6	10219	0.0021
E3-TRACE				
	# of TPs	# of FNs	# of FPs	F1 score
Ours	17	1	1448	0.0229
Nodoze	17	1	2689	0.0125
ProvDetector	5	13	44	0.1493
Flash	4	14	60484	0.0001
Karios	N/A	N/A	N/A	N/A

limited n-hop neighbors. We employ the tag-propagation algorithm to aggregate information along long paths in the provenance graph, while requiring only minimal cache memory.

Comparison We also provide a comparison with SOTA provenance-based intrusion detection systems in Table 5, including Nodoze (Hassan et al., 2019), which uses the frequency database as a baseline, and Flash (Rehman et al., 2024) and Kairos (Cheng et al., 2023), which adopt GNN-based approaches. Since different methods have different detection granularities, we uniformly use node-level granularity for statistics.

The results demonstrate the advantages of our approach in terms of accuracy and the reduction of false positives. Meanwhile, GNN-based methods not only incur high computational and memory overhead when processing large-scale provenance graphs, but also struggle to support real-time intrusion detection tasks. Kairos (Cheng et al., 2023) adopts an approach based on time-window; however, due to the complex and prolonged spatio-temporal characteristics of APT attacks, a complete attack may span multiple disjoint time windows, thereby hindering the detection of the entire attack chain.

5. Conclusion and Discussion

In this paper, we address the out-of-vocabulary (OOV) problem caused by the constant emergence of new entities and relationships, and significant computational and memory overhead resulting from processing large-scale provenance graphs. To alleviate the issues, we propose an adaptive and scalable provenance-based intrusion detection model. For real-time provenance graph streams, we use a lightweight neural model to assign a regular score to every event, combining it with the tag-propagation algorithm, to conduct efficient detection.

The prolonged spatio-temporal characteristics of APT attacks, along with the structural complexity and scale of provenance graphs, pose significant challenges to the widespread adoption of GNN-based models and transformer architectures for this task. We evaluated our method on multiple datasets and conducted a comparative analysis against SOTA approaches, demonstrating the efficiency, adaptability, and scalability of our method, as well as its effectiveness in reducing false positives.

Acknowledgement

This work was supported in part by the National Key Research & Development Project of China (2023YFB3106800), the NSFC under No. U244120033, U24A20336, 62172243, 62402425, 62402418 and 62402419, the China Postdoctoral Science Foundation under No. 2024M762829, the Zhejiang Provincial Natural Science Foundation under No. LD24F020002, the Zhejiang Provincial Priority- Funded Postdoctoral Research Project under No. ZJ2024001, by the "Pioneer and Leading Goose" R&D Program of Zhejiang (2024C03288, 2025C02263), and by the Ningbo Yongjiang Talent Programme.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Darpa transparentcomputing, 2015.2. URL https://www.darpa.mil/program/transparent-computing.

Alahmadi, B. A., Axon, L., and Martinovic, I. 99% false positives: A qualitative study of SOC analysts' perspectives on security alarms. In *31st USENIX Security Symposium (USENIX Security 22)*, pp. 2783–2800, Boston, MA, August 2022. USENIX Association. ISBN

978-1-939133-31-1. URL https://www.usenix.org/conference/usenixsecurity22/presentation/alahmadi.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 06 2017. ISSN 2307-387X. doi: 10.1162/tacl_a_00051.

Cheng, Z., Lv, Q., Liang, J., Wang, Y., Sun, D., Pasquier, T., and Han, X. Kairos:: Practical intrusion detection and investigation using whole-system provenance. *arXiv preprint arXiv:2308.05034*, 2023.

Devlin, J. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Ding, H., Zhai, J., Deng, D., and Ma, S. The case for learned provenance graph storage systems. In *Proceedings of the 32nd USENIX Conference on Security Symposium*, SEC '23, USA, 2023. USENIX Association. ISBN 978-1-939133-37-3.

Gehani, A. and Tariq, D. SPADE: support for provenance auditing in distributed environments. pp. 101–120. Springer-Verlag New York, Inc., 2012.

Han, X., Pasquier, T. F. J., Bates, A., Mickens, J., and Seltzer, M. I. Unicorn: Runtime provenance-based detector for advanced persistent threats. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

Hassan, W. U., Guo, S., Li, D., Chen, Z., Jee, K., Li, Z., and Bates, A. Nodoze: Combatting threat alert fatigue with automated provenance triage. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

Hossain, M. N., Milajerdi, S. M., Wang, J., Eshete, B., Gjomemo, R., Sekar, R., Stoller, S., and Venkatakrishnan, V. $sleuth$: Real-time attack scenario reconstruction from $cots$ audit data. In *26th USENIX Security Symposium (USENIX Security 17)*, pp. 487–504, 2017.

Hossain, M. N., Sheikhi, S., and Sekar, R. Combating dependence explosion in forensic analysis using alternative tag propagation semantics. In *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 1139–1155. IEEE, 2020.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. Tinybert: Distilling bert for natural language understanding, 2020.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. Bag of tricks for efficient text classification, 2016. URL https://arxiv.org/abs/1607.01759.

Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pp. 489–504, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450347037. doi: 10.1145/3183713.3196909.

Le, Q. V. and Mikolov, T. Distributed representations of sentences and documents, 2014. URL https://arxiv.org/abs/1405.4053.

Li, Z., Chen, Q. A., Yang, R., Chen, Y., and Ruan, W. Threat detection and investigation with system-level provenance graphs: a survey. *Computers & Security*, 106:102282, 2021.

Li, Z., Wei, Y., Shen, X., Wang, L., Chen, Y., Xu, H., Ji, S., and Zhang, F. Tags: Real-time intrusion detection with tag-propagation-based provenance graph alignment on streaming events. *arXiv preprint arXiv:2403.12541*, 2024a.

Li, Z., Wei, Y., Shen, X., Wang, L., Chen, Y., Xu, H., Ji, S., Zhang, F., Hou, L., Liu, W., Zhang, X., and Ying, J. Marlin: Knowledge-driven analysis of provenance graphs for efficient and robust detection of cyber attacks, 2024b. URL https://arxiv.org/abs/2403.12541.

Manzoor, E., Milajerdi, S. M., and Akoglu, L. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 1035–1044, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939783.

Mikolov, T. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Milajerdi, S. M., Gjomemo, R., Eshete, B., Sekar, R., and Venkatakrishnan, V. Holmes: Real-time apt detection through correlation of suspicious information flows. In *IEEE Symposium on Security and Privacy (SP)*, 2019. doi: 10.1109/SP.2019.00026.

program, D. T. Transparent Computing Engagement 3 Data Release, 2020. URL https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md.

Rehman, M. U., Ahmadi, H., and Hassan, W. U. Flash: A comprehensive approach to intrusion detection via provenance graph representation learning. In *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 139–139. IEEE Computer Society, 2024.

Wang, Q., Hassan, W., Li, D., Jee, K., Yu, X., Zou, K., Rhee, J., Chen, Z., Cheng, W., Gunter, C., and Chen, H. You are what you do: Hunting stealthy malware via data provenance analysis. 01 2020. doi: 10.14722/ndss.2020.24167.

Wang, S., Wang, Z., Zhou, T., Sun, H., Yin, X., Han, D., Zhang, H., Shi, X., and Yang, J. Threatrace: Detecting and tracing host-based threats in node level through provenance graph learning. *IEEE Transactions on Information Forensics and Security*, 17:3972–3987, 2022.

Xie, Y., Feng, D., Hu, Y., Li, Y., Sample, S., and Long, D. Pagoda: A Hybrid Approach to Enable Efficient Real-time Provenance Based Intrusion Detection in Big Data Environments. *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2018. ISSN 1545-5971. doi: 10.1109/TDSC.2018.2867595.

Zeng, J., Wang, X., Liu, J., Chen, Y., Liang, Z., Chua, T.-S., and Chua, Z. L. Shadewatcher: Recommendation-guided cyber threat analysis using system audit records. In *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 489–506. IEEE, 2022.

A. Related Work

A.1. Anomaly-based PIDSes

As cyberattacks become increasingly rampant, provenance graphs are widely used in network malicious activity detection. The anomaly detection approaches (Rehman et al., 2024; Cheng et al., 2023; Wang et al., 2022; Zeng et al., 2022) train learning models that learn the benign behavior patterns based on the historical provenance graphs, and subsequently detect anomalous behaviors that deviate from these learned benign baseline. To extract semantic information during detection, some methods employ embedding techniques to encode various node attributes present in provenance graphs, such as process names, command line arguments, file paths, and IP addresses, into semantically rich feature vectors. Some methods also use machine learning models to generate contextual embeddings, thereby establishing a baseline for benign behavior. For example, ShadeWatcher (Zeng et al., 2022) used TransR to learn a separate representation for a system entity conditioned on different relations and then used Graph Neural Network (GNN) to capture high-order connectivity. And Flash (Rehman et al., 2024) generated semantic and contextual embeddings using Word2Vec and GNN, respectively.

A.2. Learned Storage

The learned indexes, replacing traditional index structures with machine learning models, have been demonstrated that can offer substantial advantages in storage and query efficiency (Kraska et al., 2018). Leonard (Ding et al., 2023) applies learning-based indexing methods to the storage and querying of provenance graphs, simplifying both the storage and query process by optimizing the storage format and replacing traditional indexing methods with the Deep Neural Network (DNN).

Inspired by these works (Hassan et al., 2019; Ding et al., 2023; Kraska et al., 2018), we introduce a learned-baseline-based approach to overcome the challenges of PIDSes. This method adapts to various entities and events within real-world scenarios. Subsequently, we integrate an anomaly path mining algorithm into the tag-propagation framework, enabling efficiently process real-time event streams. This method optimizes the storage of raw provenance graphs, reduces caching overhead during real-time detection, and facilitates timely alerts.

B. Selected Models for Our Tasks

B.1. Embedding Models

- **Word2Vec.** Word2Vec (Mikolov, 2013) uses deep neural networks with two hidden layers, continuous bag-of-words (CBOW) model to create a low-dimensional dense vector for each word, ensuring that semantically similar words have similar vector representations.
- **Doc2Vec.** While bag-of-words features have notable limitations, Doc2Vec (Le & Mikolov, 2014) represents each document as a dense vector, offering a solution to overcome the shortcomings of bag-of-words models.
- **FastText.** Unlike Word2Vec which feeds individual words into the neural network, FastText (Joulin et al., 2016; Bojanowski et al., 2017) breaks words into n-grams (sub-words) and then represents words as the sum of the n-gram vectors to enhance its embedding capability of words.
- **TinyBert.** While contextual word embedding techniques like BERT (Devlin, 2018) effectively capture entity information, their high computational cost makes them inefficient for large-scale provenance graph processing; therefore, we adopt TinyBERT (Jiao et al., 2020) to perform the event embedding task.

B.2. Out-of-vocabulary Embedding

As Doc2Vec and FastText are capable of handling OOV words through specific encoding mechanisms.

Doc2Vec can use `infer_vector()` function to encode documents that do not appear in the training set. We treat an event as a sentence, which is the unit processed by Doc2Vec. The inference process optimizes the event vector $\mathbf{v}_e$ to maximize its compatibility with the model's trained parameters, given the context of the words in e_i. Formally, this can be

expressed as:

$$\mathbf{v}_{e_i} = \arg\max_{\mathbf{v}} \prod_{t \in e_i} P(t|\mathbf{v}, \Theta) \tag{15}$$

where t represents tokens in the event e_i and $P(t|\mathbf{v}, \Theta)$ is the probability of token t given the embedding $\mathbf{v}$ and the model parameters Θ. The optimization is typically performed through a few iterations of the stochastic gradient descent (SGD) algorithm, starting from a random vector $\mathbf{v}$.

As a subword-level embedding model, FastText could encode each word into a targeted length vector using the subwords. Each word w is decomposed into a set of subwords $S_w = \{s_1, s_2, ..., s_n\}$, where each subword is a substring of word w. If $w \notin L$, its embedding vector $\mathbf{v}_w$ is computed as the average embeddings of its constituent subwords. Formally, this process can be expressed as:

$$\mathbf{v}_w = \frac{1}{|S_w|} \sum_{s \in S_w} \mathbf{v}_s \tag{16}$$

where $|S_w|$ is the cardinality of the subword set S_w. After obtaining the vector for each word in an event, we concatenate them sequentially to form the completed vector representation of the event sentence.

B.3. Learning Models

- **MLP.** A Multi-Layer Perceptron (MLP) model is composed of fully connected layers that perform feature extraction and transformation. The input layer receives encoded event vectors, which are subsequently processed by the hidden layers. During training, the system employs the backpropagation algorithm to compute the gradients of the loss function with respect to weights and biases. These parameters are iteratively updated via the gradient descent method to minimize loss.
- **LSTM.** Long Short Term Memory (LSTM) networks forward the data not only in a spatial direction, but also in a time-dependent direction, allowing the network to capture temporal dependencies more effectively. A key advantage of LSTM lies in their ability to address the gradient vanishing problem often encountered by standard RNNs when processing long sequences. Utilizing memory cells, LSTM retains long-term context information, making them particularly well suited for tasks requiring the modeling of extended dependencies. Despite these advantages, training LSTM can be more computationally demanding than training simpler architectures like MLP, due to the complexity of their internal structure, which involves multiple gating mechanisms and memory updates.
- **CNN.** Convolutional Neural Network (CNN) models are featured by varying groups of convolutional and pooling layers, which make them computationally intensive, as each convolutional layer requires extensive matrix operations and local feature extractions, and pooling layers contribute to additional computational complexity. A CNN is formed by several convolution and pooling operations, usually followed by one or more fully connected layers. While CNNs are traditionally applied to spatial data, they can be adapted to regression tasks by modifying the architecture and loss functions. In an MLP, one just has to compute two kinds of backpropagations: from output to fully connected layers and fully connected layers to fully connected layers. In a traditional CNN, four new kinds of propagations have to be computed: fully connected layers to pool layers, pool layers to conv layers, conv layers to conv layers and conv layers to pool layers.

C. Tag-propagation Framework

Here, we provide a detailed description of the four stages of the tag-propagation framework: tag initialization, tag propagation, tag removel, and alert triggering.

- **Tag Initialization.** We analyze each event in the log stream. Firstly, each event e_i is encoded into a vector representation $\mathbf{v}_{e_i}$. The trained machine learning model f_θ is then applied to compute the predicted regular score $\hat{y}_i$. We use $\hat{y}_i$ as the event frequency Γ_e. In order to avoid excessive tags accumulation, only when the regular score of the event (Γ_e) is lower than the preset threshold and the source node does not have a tag, a new tag will be assined and stored in the destination node.
- **Tag Propagation.** We check whether there is a tag in the source node of each event. If so, we will update the regular score of the path (Γ_P) according to Equation 14 and log the new tag result, while retaining the previous tag for subsequent propagation. When different tags converge on a node, we will retain the lowest regular score and its corresponding path, while archiving other paths for potential backtracking and attack reconstruction.
- **Tag Removal.** To avoid the dependency explosion problem (Hossain et al., 2020), we use the dacay factor (α) to remove tags. If no suspicious events are encountered during the propagation process, Γ_P gradually increases and is removed when it reaches a certain threshold. On the other hand, tags that have not been updated for a long time will also be removed.
- **Alert Triggering.** If suspicious events are continuously encountered, Γ_P will gradually decrease. Once it falls below a predefined threshold, an alert will be triggered to provide timely notifications. The triggered alert tag is also retained and propagated for several more rounds to ensure that subsequent suspicious activities can be effectively captured.

Table 6. Detailed Detection Results of Single Events and Paths

Embedding Models	Learning Models	Provenance Graph Embedding			Anomaly Path Mining				
		Train Time(s)	Accuracy1(%)	Accuracy2(%)	True Positives	False Positives	Precision	Recall	F1 Score
Word2Vec*	MLP	302.22	87.03	100.00	67	0	1.0000	0.6667	0.8000
	LSTM	648.85	89.01	100.00	38	6	0.8636	0.8889	0.8761
	CNN	2120.37	90.54	100.00	598	0	1.0000	0.5000	0.6667
Doc2Vec*	MLP	157.62	90.55	100.00	87	2	0.9775	0.8889	0.9311
	LSTM	288.44	90.55	100.00	87	2	0.9775	0.8889	0.9311
	CNN	2148.61	90.54	100.00	84	2	0.9767	0.5556	0.7083
FastText*	MLP	184.02	90.59	100.00	90	1	0.9890	1.0000	0.9945
	LSTM	289.03	90.58	100.00	90	1	0.9890	1.0000	0.9945
	CNN	2435.54	90.58	100.00	90	0	1.0000	1.0000	1.0000
FastText(weight)*	MLP	170.80	90.56	57.14	978	0	1.0000	0.3333	0.5000
	LSTM	290.31	90.56	58.57	21	0	1.0000	0.3889	0.5600
	CNN	2289.49	90.55	55.71	1722	330	0.8392	0.3333	0.4771
FastText(onehot)*	MLP	328.71	85.58	70.00	11	6	0.6471	0.2778	0.3887
	LSTM	478.22	84.91	97.14	21	1	0.9545	0.3889	0.5526
	CNN	955.17	89.34	97.14	104	13	0.8889	1.0000	0.9412
Doc2Vec	MLP	631.38	67.40	27.14	154	1460	0.0954	0.0556	0.0703
	LSTM	1072.17	69.02	30.00	7	839	0.0083	0.1667	0.0158
	CNN	4543.2	47.22	30.00	271	1217	0.1821	0.1667	0.1471
FastText	MLP	298.32	90.54	48.57	47	9091	0.0051	0.7778	0.0102
	LSTM	317.98	90.42	68.57	36	379	0.0867	0.1667	0.1141
	CNN	2457.85	90.50	61.43	9	178	0.0481	0.3333	0.0841
FastText(weight)	MLP	194.24	90.58	55.71	22	0	1.0000	0.3333	0.5000
	LSTM	286.00	90.57	57.14	14	0	1.0000	0.3333	0.5000
	CNN	2216.07	90.55	55.71	39	32	0.5493	0.3333	0.4149
FastText(onehot)	MLP	216.71	90.57	58.57	48577	69868	0.4101	0.5556	0.4719
	LSTM	267.41	90.53	52.86	22824	6040	0.7907	0.2222	0.3469
	CNN	740.50	90.52	57.14	70	249	0.2194	0.0000	0.0000
TinyBERT	MLP	275.06	90.45	12.86	435	0	1.0000	0.0000	0.0000
	LSTM	298.42	90.16	10.00	85	3183	0.0260	0.0556	0.0354
	CNN	1838.12	90.45	11.43	8050	3	0.9996	0.1111	0.2000

D. More Results

Here, we present more detailed experimental results. To provide a more comprehensive explanation of the experimental design and to demonstrate the effectiveness of our method, Table 6 presents a more detailed breakdown of the results shown in Table 4. In addition, we conducted a parameter sensitivity analysis on the regression model, evaluated the model's adaptability on additional datasets, and performed ablation studies on the learned baseline.

D.1. Detection Results

Different representation methods and learning models can affect the performance of provenance graph embedding and representation learning. As shown in the *Provenance Graph Embedding* section of Table 6, we present the training time, testing time, and prediction accuracy of different methods and models. The meanings of *Accuracy1(%)* and *Accuracy2(%)* are consistent with those described in the main text. Similarly, Table 6 also presents the performance of using embedded provenance graphs as a baseline in the downstream task of anomaly detection, specifically in the *Anomaly Path Mining* section. In this case, an alert path is considered a *True Positive* (TP) if it contains at least one malicious node from the ground truth. Conversely, if the alert path does not contain any malicious nodes, it is condidered a *False Positive* (TP).

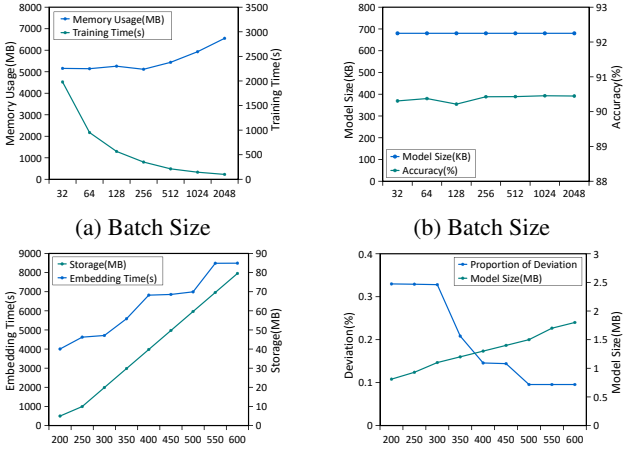
D.2. Parameter Sensitivity

Empirically chosen default parameters are used for the embedding model, whereas optimal hyperparameters for the regression model are identified through controlled experiments. Table 7 illustrates the controlled experiments for tuning hyperparameters in the regression model, such as learning rate and optimizer. Figure 3 represents the effects of the batch size and the dimension of the event embedding on memory usage, computational efficiency, prediction accuracy, and et al. Table 7 shows that hyperparameter tuning has limited influence on the predictive performance of the regression model. Therefore, in the experiments reported in the main text, we use an *L1* kernel regularizer with a coefficient of *0.001*, the *Adam* optimizer (*learning rate* = *0.001*), and the *Mean Squared Error* (MSE) loss function.

To study the impacts of batch sizes, we use different batch sizes from 32 to 2048 to train MLP models on the E3-

Table 7. Hyperparameters Tuning for Regression Models

	Training Time(s)	Accuracy 1(%)	Accuracy 2(%)
Kernal Regularizer			
L2(0.0001)	176.51	90.59	100.00
L2(0.001)	179.33	90.59	100.00
L2(0.01)	229.91	90.59	100.00
L1(0.0001)	174.01	90.59	100.00
L1(0.001)	278.39	90.59	100.00
L1(0.01)	123.69	35.86	0.00
Learning Rate			
Adam(0.0001)	176.30	90.59	100.00
Adam(0.001)	155.75	90.59	100.00
Adam(0.01)	124.24	90.50	100.00
SGD(0.0001)	605.49	90.47	100.00
SGD(0.001)	244.00	90.51	100.00
SGD(0.01)	1298.20	90.56	100.00
Loss Function			
MSE	117.25	90.59	100.00
MAE	169.29	90.44	100.00
Huber	148.53	90.53	100.00



(a) Batch Size (b) Batch Size

(c) Embedding Dimension (d) Embedding Dimension

Figure 3. Hyperparameters Tuning

CADETS dataset, using the FastText model to generate event embeddings. Figure 3a shows the maximum memory usage during training and the time cost of training models with different batch sizes. As the batch size increases, the memory usage during model training gradually rises. It remains relatively constant for batch sizes below 256, but exhibits a significant upward trend for batch sizes above 256. Meanwhile, the training time decreases as the batch size increases, though the rate of decrease becomes progressively smaller. Figure 3b shows the model size and the prediction accuracy. As the batch size increases, the model size remains unchanged, while the prediction performance shows slight flustuations, but remains largely consistent. In general, the batch size can be appropriately increased within the limits of the hardware capabilities.

The dimension of event embeddings represents the complexity of encoding entities and their relationships within an event. As shown in Figure 3c, the event encoding time exhibits a fluctuating upward trend with increasing embedding dimensions, and the storage space required for the encoded files increases linearly. The model prediction results of different embedding dimensions are shown in Figure 3d. As the embedding dimension increases, the size of the learning model rises approximately linearly. Meanwhile, the proportion of predictions with deviations smaller than the threshold gradually decreases. For dimensions less than 300 or greater than 500, the decrease in deviation is not obvious; whereas between 300 and 500 dimensions, the deviation decreases markedly. Therefore, encoding each event into a 500-dimensional space is considered more appropriate.

D.3. Adaptability Experiment

To evaluate the adaptability of our provenance graph embedding method, we tested the two best performing methods from Table 4 and Table 6 on other datasets, with the results presented in Table 8. Our method demonstrates good embedding performance and detection result on these datasets as well, indicating its adaptability.

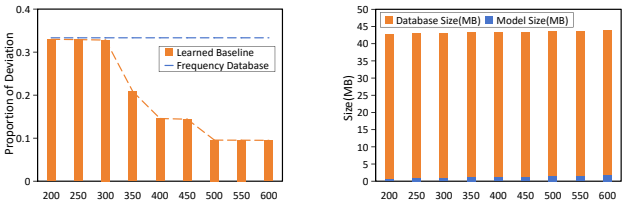
D.4. Ablation Study

To assess the effectiveness of the learned baseline constructed by our method compared to the traditional dababase baseline, we conducted an in-depth evalution of the prediction accuracy of the learned baseline and the frequency database in the ablation study, illustrated in Figure 4. Figure 4a shows the proportion of correct predictions made by the learning model and the frequency database prediction model. We find that even using a 200-dimensional vector to represent each event, the deviation rate remains smaller than the frequency database. Figure 4b illustrates the storage of the database compared to the learning-based baseline at different embedding dimensions. It is evident that the storage of the learning model is much smaller than the frequency database. When each event is encoded into a 500-dimensional vector space, the storage of the frequency database is approximately 28 times larger than the learning model. The results indicate that the learning-based baseline not only significantly optimizes storage but also outperforms the frequency database in terms of prediction accuracy for unseen events.

This discrepancy arises primarily from fundamental differences in their prediction mechanisms. The frequency database relies on the frequencies of historical events stored in its database for predictions. If an event is not stored in

Table 8. Results on Other Datasets

Datasets	Embedding Models	Provenance Graph Embedding			Anomaly Path Mining				
		Training Time(s)	Testing Time(s)	Accuracy1(%)	True Positive	False Positive	Precision	Recall	F1 score
E3-TRACE	FastText*	114.06	3.62	92.08	1403	47	0.9676	0.8182	0.8866
	FastText(weight)	57.21	2.73	91.76	1374	424	0.7642	0.8182	0.7903
E3-THEIA	FastText*	207.60	8.63	86.73	123	0	1.0000	0.7692	0.8695
	FastText(weight)	316.29	8.36	86.43	121	2	0.9837	0.6154	0.7571



(a) Comparison (b) Comparison

Figure 4. Ablation Study

the database or occurs infrequently, the prediction relied on the frequency database often exhibits considerable bias or is entirely inaccurate. While the database can provide frequency of known events, there is no effective strategy to predict the frequency of unseen or less frequent events in history.