PROVSYN: Synthesizing Provenance Graphs for Data Augmentation in Intrusion Detection Systems

Yi Huang*, Wajih UI Hassan†§, Yao Guo*, Xiangqun Chen*, Ding Li*§
*Key Laboratory of High-Confidence Software Technologies (MOE), School of Computer Science, Peking University
†University of Virginia
yihuang@stu.pku.edu.cn, hassan@virginia.edu, yaoguo,cherry,ding_li@pku.edu.cn

***Abstract*—Provenance graph analysis plays a vital role in intrusion detection, particularly against Advanced Persistent Threats (APTs), by exposing complex attack patterns. While recent systems combine graph neural networks (GNNs) with natural language processing (NLP) to capture structural and semantic features, their effectiveness is limited by class imbalance in real-world data. To address this, we introduce PROVSYN, an automated framework that synthesizes provenance graphs through a three-phase pipeline: (1) heterogeneous graph structure synthesis with structural-semantic modeling, (2) rule-based topological refinement, and (3) context-aware textual attribute synthesis using large language models (LLMs). PROVSYN includes a comprehensive evaluation framework that integrates structural, textual, temporal, and embedding-based metrics, along with a semantic validation mechanism to assess the correctness of generated attack patterns and system behaviors. To demonstrate practical utility, we use the synthetic graphs to augment training datasets for downstream APT detection models. Experimental results show that PROVSYN produces high-fidelity graphs and improves detection performance through effective data augmentation.**

1. Introduction

Data provenance [29], which involves recording the history of data modifications, access, and usage, serves as a critical tool in cybersecurity systems for tracing attacks [61]. However, as system scales expand, the volume of provenance data increases dramatically, posing challenges for data management and analysis. To address this, researchers have begun to represent the provenance of the data using graph structures and applying graph machine learning algorithms for the detection and tracking of intrusions [21, 53, 62]. This approach effectively captures the complexity of system activities and enables the detection of attacks that traditional methods struggle to identify, such as zero-day attacks and APT attacks.

The quality of provenance data sets largely determines the effectiveness of provenance-based intrusion detection systems (IDS) trained on them. However, there are very few datasets available, and even fewer datasets that can accurately describe benign and malicious activities in a system. One possible solution is to produce system logs captured from real-world scenarios. However, one of the key disadvantages of using such datasets is privacy concerns. Sharing such dataset across the community could result in leakage of critical privacy information, and hence it's difficult for organizations to release such datasets. Another solution is to collect benign and malicious activities in experimental environments. For example, the DARPA Transparent Computing (TC) [2] program contributes two widely used datasets in the cybersecurity community, DARPA TC dataset and DARPA OpTC dataset. However, such datasets, while giving us insights into system activities, convey limitations, one of which being the class imbalance issue [6], where less common or underrepresented program behaviors are undersampled. This imbalance hampers the generalization ability of machine learning models trained on such data, limiting their effectiveness in unseen environments [6, 9, 76, 77].

To mitigate the issue of data imbalance, a common approach is to synthesize high-fidelity provenance data that expands coverage of minority classes and patterns. Such data synthesis can be achieved through methods such as graph generation models or large language models (LLMs), yet each has critical limitations. On the one hand, most graph generation models focus on structural connectivity and lack support for generating rich textual attributes, which are essential for downstream intrusion detection tasks [12, 37, 53, 71]. On the other hand, although LLMs are strong at producing semantically rich log entries, their outputs often fail to preserve the underlying graph structure when transformed into provenance graphs [1, 4].

To overcome these challenges, we propose PROVSYN, a hybrid framework that integrates the structural modeling ability of graph generation models with the text generation ability of LLMs to jointly synthesize provenance graphs with both accurate topology and meaningful textual attributes. Specifically, PROVSYN is a three-phase synthesis framework comprising structural synthesis, topological refinement, and textual attribution. The initial phase constructs heterogeneous graph topology by generating nodes and edges with categorical attributes. Subsequent structural refinement applies domain-specific rules to eliminate unrea-

§ *Co-corresponding authors*

sonable connections and isolated entities. In the third phase, an LLM is used to generate textual attributes of nodes. However, the implementation of our framework confronts three principal technical challenges:

- **Scalability to provenance graphs.** Provenance graphs are heterogeneous graphs with more than 10,000 nodes and edges, whereas most heterogeneous graph generation models are designed for molecular graphs and are typically trained on graphs with fewer than 100 nodes [17, 72]. Effectively scaling graph generation models designed for other domains to large-scale provenance graphs thus remains a challenge.
- **Textual attributes synthesis.** First, LLMs inherently struggle with graph structure comprehension [33], necessitating transformation of provenance graphs into LLM-interpretable formats. Second, due to lacking prior knowledge of typical provenance entity names, direct LLM prompting produces simplistic, low-diversity outputs.
- **Evaluation framework gap.** The absence of a multidimensional evaluation framework hinders comprehensive assessment of synthesized graphs, particularly in verifying the semantic correctness of attack patterns and system behavior within generated provenance structures.

To address the first challenge, we adopt a domain-agnostic heterogeneous graph generation network [17] to jointly model node and edge labels along with structural connectivity. To make the training process scalable to large graphs, we adopt a restart-based random walk for subgraph sampling. Through a novel restart mechanism, this sampling algorithm escapes local regions and captures subgraphs that better reflect the overall graph structure.

To address the second challenge, we convert the graph structure into depth-first search (DFS) sequences to enhance the LLM's understanding of the graph. In addition, we also designed two different masking strategies, including fully masking and partially masking, to construct a training set for fine-tuning the LLM, thereby providing it with knowledge of entity names in the provenance graph.

To address the third challenge, we design a comprehensive framework that evaluates fidelity from five perspectives: structural, textual, temporal, embedding, and semantic correctness. For structural fidelity, we adopt five *Mean Discrepancy (MMD)*-based metrics. For textual fidelity, we use standard NLP metrics, including *BLEU*, *GLEU*, and *ROUGE*. For temporal fidelity, we use sequence comparison metrics including *Longest Common Subsequence (LCS)* and *Dynamic Time Warping (DTW)*. For embedding fidelity, we utilize various embedding methods, including *DeepWalk* and *Doc2Vec*. To accurately assess semantic correctness—the most challenging aspect—we adopt a model-based approach. Specifically, we sample positive and negative triples from real provenance graphs and train a classification model using contrastive learning.

In the experiments, we used our proposed evaluation framework to evaluate the provenance data synthesized by LLMs, including GPT-4o and Claude 3.7, as well as by PROVSYN. The results show that PROVSYN achieves higher fidelity than these strong LLMs. Moreover, by generating 100% novel and 100% diverse graphs, PROVSYN effectively mitigates the data imbalance issue in provenance datasets. When applied to an intrusion detection system, models trained on datasets augmented with PROVSYN-generated graphs exhibited reduced false positive rates during testing.

The main contributions of our work are as follows:

- We propose PROVSYN, a three-phase provenance graph synthesis framework that enables the construction of high-fidelity heterogeneous graphs with rich textual attributes.
- We design a comprehensive multi-dimensional evaluation framework that evaluates the fidelity of synthesized provenance graphs, including: structural, textual, temporal, embedding, semantic correctness.
- We mitigate the data imbalance problem in provenance datasets by synthesizing novel and diverse provenance graphs, thereby enhancing the generalizability of downstream intrusion detection models.
- The proposed generation and evaluation frameworks can also be extended to other domains involving Heterogeneous Information Networks (HINs), such as social networks and citation networks.

2. Background and Motivation

2.1. Class imbalance in Provenance Graphs

We statistically analyzed the distributions of *entity type* and *event type* across four commonly-used public provenance datasets (in Appendix A Figure 3). Among the datasets, Cadets, Theia, and Trace are collected from DARPA Engagement 3, and Nodlink [37] is simulated provenance data collected on an Ubuntu system using Sysdig. The distributions exhibit a typical long-tail pattern and reveal significant data imbalance. In both entity type and event type, a single category accounts for over 50% of instances, exceeding 70% in some cases. In contrast, most other categories contribute less than 20%, with many below 5%. This imbalance further reflects the uneven distribution of system behaviors and attack patterns in most provenance datasets.

The class imbalance problem poses a significant challenge for learning-based intrusion detection [21, 22, 39, 62, 74]. Previous studies in the deep learning community have revealed that class imbalance has a detrimental effect on a model's classification performance as well as its generalization abilities [8, 34]. When deployed in open-world out-of-distribution (OOD) environments, detection models trained on imbalanced datasets may significantly exhibit performance degradation [25, 67]. Hence, a more balanced dataset is vital for deep learning-based provenance-based IDS (PIDS) to achieve optimal results. However, this problem is largely ignored by the community, as there are very few studies considering this issue.

2.2. Problems with Existing Methods

To mitigate class imbalance, conventional techniques such as undersampling [44], oversampling [47], and the Synthetic Minority Over-Sampling Technique (SMOTE) [9] have been proposed. However, these methods are not suitable for provenance graph data. Undersampling breaks the causality chain between system entities, while oversampling does not add variety to benign activities and can cause overfitting. SMOTE, which relies on linear interpolation in the feature space, cannot be applied to provenance graphs due to overlapping distributions of different classes in the feature space.

Another approach for augmenting provenance datasets is to leverage graph generation models to synthesize provenance graphs. However, most existing models focus primarily on generating the structural connectivity among nodes and edges, without supporting additional node or edge attributes [12, 71]. In contrast, provenance graphs contain rich textual attributes associated with nodes and edges, and the text attributes are crucial features for downstream intrusion detection systems [37, 53]. Yet, to date, no graph generation model inherently supports the joint generation of both graph structure and textual attributes.

With the recent advances in large language models, their capabilities in natural language understanding and generation have been widely demonstrated [1, 4]. This motivates the exploration of the use of LLMs to synthesiz provenance data. Specifically, LLMs can be tasked with creating new provenance log entries. These generated logs contain rich texual and semantic information. However, when converting LLM-generated logs into graphs and evaluating them using graph-based metrics, their structural fidelity falls significantly short of graphs produced by heterogeneous graph generation models (see details in Section 5.2). This indicates that although current LLMs perform well in text synthesis, they lack a strong capability for synthesizing accurate graph topology and structure.

To address these limitations, we propose PROVSYN, a framework that integrates the complementary strengths of graph generation models for structural synthesis and large language models for text attribute generation.

3. Related Work

Provenance-based Intrusion Detection System Provenance-based detection methods can be divided into three main categories [74]: rule-based, statistics-based, and learning-based techniques. Rule-based methods [24, 28, 46] rely on predefined heuristics derived from known attack patterns to identify malicious activities. Statistics-based [23, 45, 61] approaches analyze deviations in provenance graph elements by measuring their statistical anomalies. Learning-based techniques employ deep learning models to capture either normal system behavior [21, 62] or malicious patterns [39, 41, 74], framing APT detection as either a classification [39] or anomaly detection task [22, 74]. These learning-based methods further branch into sequence-based approaches [41], which focus on execution workflows, and graph-based [21, 22, 39, 62, 74] methods, which use graph neural networks to model entity relationships and detect behavioral anomalies.

Network Traffic Augmentation To address class imbalance in network traffic datasets, one line of work applies sampling techniques. Jiang et al. [32] adopt one-side selection to remove noisy majority samples and apply SMOTE [9] to augment minority instances. Liu et al. [42] propose the Difficult Set Sampling Technique to improve sample representativeness. Another line of research explores synthetic network traffic generation using deep learning. Generative models, especially GANs, have been employed for various scenarios, including flow-based traffic [55], manipulated traffic [11], and social media traffic [54], addressing challenges in realistic traffic synthesis [5].

LLM-driven Synthetic Data Generation LLM-based synthetic data generation methods can be categorized into three main types: prompt engineering-based approaches [16, 20, 38, 43, 57, 68, 70], which utilize task descriptions, condition-value prompts, and in-context examples to steer generation; multi-step generation-based approaches [14, 15, 27, 63, 64], which break down complex tasks either at the sample or dataset level; and knowledge-enhanced approaches [10, 30, 50], which incorporate external knowledge such as knowledge graphs or web sources to improve the factual quality of outputs. The evaluation of synthetic data includes both direct metrics, which measure faithfulness [36] and diversity [73], and indirect metrics, which assess performance on downstream tasks through benchmark testing [58] and open-ended evaluation using human [26] or model-based [65] methods.

4. PROVSYN Design

We implement the generation of provenance graphs in three stages. In the first stage, we train a heterogeneous graph generation model using real provenance graph dataset to produce the graph structure. After that, we apply rule-based post-processing to remove invalid edges and isolated nodes. In the third stage, we fine-tune a large language model to label the nodes with appropriate names.

4.1. Problem Definition

A system audit log dataset can be represented as a provenance graph, where nodes correspond to system entities and directed edges represent system events. Formally, a provenance graph is a directed heterogeneous graph $G = (V, E)$, where each node $v_i \in V$ is associated with a type $t_i = f(v_i)$ and a name $n_i = h(v_i)$, and each edge $e_i \in E$ is associated with a type $l_i = g(e_i)$. Each system event is represented as a labeled 5-tuple:

$$e_i = (t_i, n_i, t_j, n_j, l_i)$$

where t_i and t_j denote the types of the source and target nodes, n_i and n_j denote their names, and l_i denotes the

edge type. PROVSYN is to learn a distribution over such provenance graphs from real system audit data, enabling the generation of synthetic graphs that preserve both the structural and semantic properties of the original dataset. The generated graphs are also heterogeneous, with diverse node and edge types, and each node is assigned a detailed name.

The objectives of PROVSYN are threefold: (1) to synthesize high-fidelity provenance graphs. This is our primary goal, as high-quality provenance data is essential for training reliable detection models; (2) to mitigate data imbalance in existing provenance graph datasets. PROVSYN is designed to learn the underlying distribution of real-world data and generate samples that represent underrepresented scenarios, thereby improving data diversity; and (3) to enhance the performance of downstream intrusion detection systems. By augmenting existing datasets with PROVSYN-generated graphs, the coverage of system behaviors can be expanded, leading to improved generalization and robustness of IDS models.

4.2. Real-World Provenance Graph Construction

We begin by parsing each log entry to extract five key elements: source node type, source node name, destination node type, destination node name, and edge type. These elements are used to construct a directed provenance graph, where each node is assigned two attributes (type and name), and each edge is labeled with a type attribute. To avoid duplication, we maintain a hash table indexed by the MD5 hash of node names.

For log formats with simple key-value structures, we extract these elements directly using predefined keys. For formats with more complex attribute representations of entities and interactions, we apply regular expressions to extract the target elements. Extracting node names is particularly challenging; to improve accuracy, we define type-specific regular expressions tailored to each node type, ensuring that a valid name is extracted for every node.

4.3. Graph Structure Generation

Existing graph generation models primarily focus on structural connectivity, often overlooking the modeling of node and edge labels [12, 71]. Many approaches also depend on domain-specific heuristics (e.g., those tailored to molecular graphs), limiting their generalizability across graph types [56]. In contrast, provenance graphs represent a new class of heterogeneous graphs, where nodes correspond to system entities such as processes, files, and network components, and edges represent relations such as read, write, clone, and send. Generating such graphs requires a domain-agnostic model that captures both structural and label information.

To this end, we adopt GraphGen [17], a domain-independent graph generation framework capable of jointly modeling node and edge labels along with structural connectivity. GraphGen converts graphs into sequences using minimum DFS codes, a canonical representation that preserves both topology and semantics. An Long Short-Term Memory (LSTM)-based model is trained to learn the joint distribution over these sequences, enabling the generation of labeled heterogeneous graphs in a domain-agnostic fashion.

However, GraphGen is typically trained on small graphs with around 100 nodes, whereas provenance graphs often exceed 10,000 nodes, making it infeasible to feed the entire graph into the model. To make the training process scalable to large graphs, we adopt a random walk sampling strategy to extract subgraphs, and then the subgraphs can be used to construct a GraphGen-compatible training dataset. During sampling, we perform a ***restart-based random walk***. At each step, there is a probability of returning to the starting node; otherwise, the next node is uniformly selected from the current node's neighbors. Visited nodes and their connecting edges are dynamically added to the subgraph. The walk terminates when either a predefined number of iterations is reached or the subgraph exceeds the maximum allowed number of nodes or edges. The number of walks per starting node is determined by its degree, calculated as $\texttt{factor} \times \sqrt{\texttt{degree}}$, ensuring that high-degree nodes generate more subgraphs. After sampling, each subgraph is normalized by remapping node IDs and removing self-loops. Only subgraphs that meet the node and edge limits and remain connected are retained. The reason why we adopt restart-based random walk for subgraph sampling is its ability to balance local exploration with broader graph coverage. Unlike simple random walk, which tends to get trapped in densely connected areas, and Metropolis-Hastings random walk, which often mixes slowly and introduces sampling bias in heterogeneous graphs, restart-based random walk periodically returns to the starting node. This mechanism helps it escape local regions and sample subgraphs that better reflect the overall graph structure. The complete procedure is presented in Algorithm 1.

The training process of GraphGen involves converting graphs into minimum DFS codes, followed by learning the distribution of these sequences using a LSTM network. Given a graph dataset $\mathcal{G} = \{G_1, G_2, \ldots, G_m\}$, each graph G_i is first transformed into its minimum DFS code sequence $S_i = F(G_i) = [s_1, s_2, \ldots, s_{|E_i|}]$, where $|E_i|$ denotes the number of edges in graph G_i, and each s_t is an *edge tuple* representing the t-th edge in the DFS traversal.

Each edge tuple s_t is defined as:

$$s_t = (t_u, t_v, L_u, L_e, L_v)$$

where t_u and t_v are the timestamps (DFS indices) of the source and target nodes, respectively; L_u and L_v are the labels of the source and target nodes, respectively; and L_e is the label of the edge between the two nodes.

At each time step t, the LSTM updates its hidden state h_t based on the embedding of the previous edge tuple s_{t-1}, and then predicts the distribution over the next edge tuple:

$$h_t = f_{\text{trans}}(h_{t-1}, f_{\text{emb}}(s_{t-1}))$$

$$s_t \sim p(s_t \mid h_t)$$

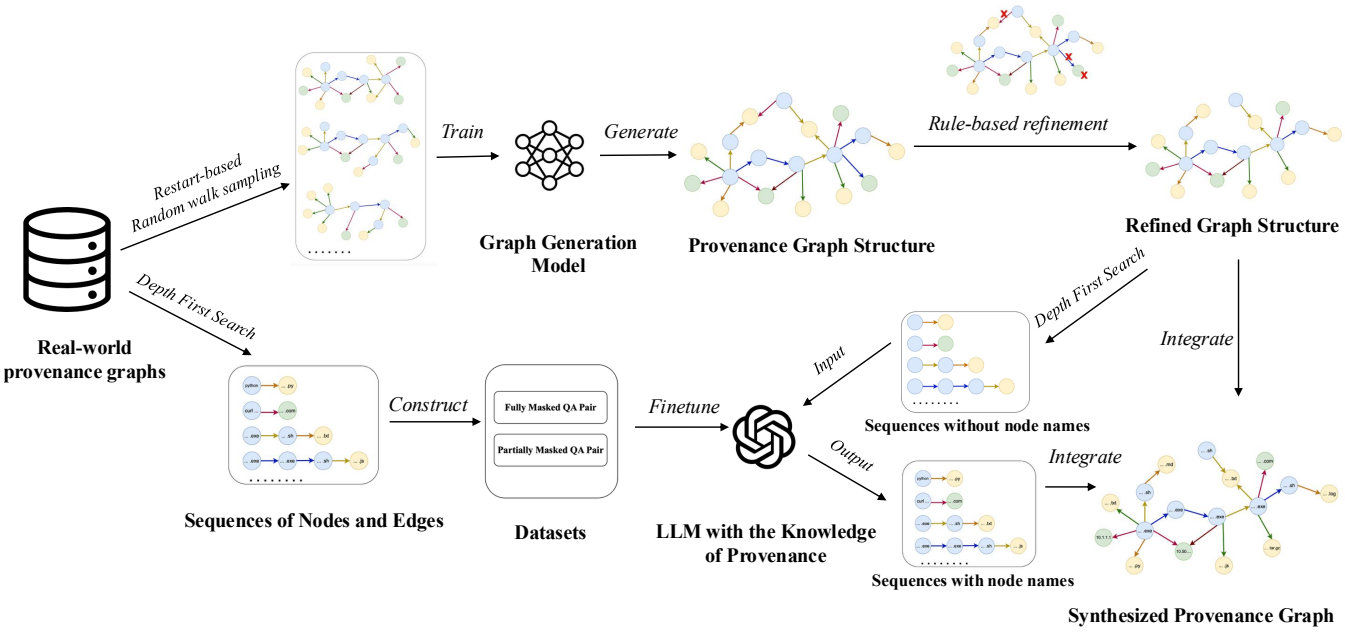


Figure 1: PROVSYN Architecture. First, a heterogeneous graph generation model is employed to generate the initial structure of a provenance graph. Subsequently, the topology of the graph is refined according to predefined rules. Finally, a large language model is used to synthesize the textual attributes of the nodes within the provenance graph.

Here, f_{emb} is the embedding function that maps edge tuples to a vector space, and f_{trans} is the transition function implemented by the LSTM.

The training objective is to minimize the binary cross-entropy (BCE) loss between the predicted and ground-truth edge tuples across the entire dataset:

$$\mathcal{L} = \sum_{i=1}^{m} \sum_{t=1}^{|E_i|} \text{BCE}(p(s_t \mid s_{<t}), s_t)$$

During inference, GraphGen generates new graphs using the trained LSTM model. Starting from an initial hidden state h_0 and a start-of-sequence symbol SOS, the model iteratively generates a minimum DFS code sequence $S = [s_1, s_2, \ldots, s_{|E|}]$. The process of updating the hidden state and sampling the next edge tuple s_t is the same as in the training phase. The generation continues until an end-of-sequence symbol EOS is produced. The resulting DFS code sequence S is then converted back into a graph G by reconstructing the edges based on the timestamps and labels in the sequence, yielding the generated graph structure and labels. This approach allows the model to generate graphs that reflect the structural and semantic patterns present in the training data.

4.4. Rule-based Post-processing

After the first stage, we obtain an undirected heterogeneous graph, where both nodes and edges have distinct types. However, this graph is not directly suitable as a provenance graph structure for two main reasons. First, provenance graphs are directed by nature, whereas the generated graph is undirected. Second, despite being trained on a large dataset, the heterogeneous graph generation model occasionally produces invalid connections—such as a `process-read-process` relation—which do not occur in real-world scenarios.

To ensure the fidelity of generated provenance graphs, we implement a rule-based post-processing pipeline. First, we convert the undirected graph into a directed graph by replacing each undirected edge (u, v) with two directed edges, (u, v) and (v, u). We then remove edges violating domain-specific constraints, such as connections between incompatible entity types or semantically invalid interactions. Finally, we remove all isolated nodes resulting from edge removal, as such nodes do not appear in real-world provenance graphs.

The domain-specific constraints applied during post-processing vary by dataset. For example, for the Nodlink dataset, the constraints are as follows: (1) all operations must originate from process nodes; (2) file operations must terminate at file nodes; (3) process operations must terminate at process nodes; and (4) network operations must terminate at either network or file nodes. For the Cadets, Theia, and Trace datasets, the constraints are: (1) non-read operations must originate from process nodes; and (2) read operations (e.g., read, receive, load) must terminate at process nodes.

4.5. Textual Attribute Generation

After the first two stages, we obtain a directed heterogeneous graph in which nodes of different types are

Algorithm 1: RESTART-BASED RANDOM WALK SUBGRAPH SAMPLING

Input : Graph $G = (V, E)$, starting node $u \in V$, iterations T, restart probability $\alpha = 0.15$, node constraints min_nodes, max_nodes, edge constraints min_edges, max_edges
Output: Valid subgraph G_{sampled} or $\emptyset$ if invalid

$G_{\text{sampled}} \leftarrow$ graph with node u (copy attributes from G)
$current \leftarrow u$
for $t \leftarrow 1$ **to** T **do**
  $r \sim \text{Uniform}[0, 1]$
  if $r < \alpha$ **then**
    $current \leftarrow u$ // Restart mechanism
  else
    $v \leftarrow$ random neighbor of $current$ in G
    if $v \notin V(G_{sampled})$ **then**
      Add v to G_{sampled} (copy attributes from G)
    Add edge $(current, v)$ to G_{sampled} (copy attributes from G)
    $current \leftarrow v$
  if $|V(G_{sampled})| \geq max\_nodes$ **then**
    break // Node limit reached
  if $|E(G_{sampled})| \geq max\_edges$ **then**
    break // Edge limit reached
Remove self-loops from G_{sampled}
Relabel nodes to consecutive integers
$n \leftarrow |V(G_{\text{sampled}})|$
$e \leftarrow |E(G_{\text{sampled}})|$
if $\neg is\_connected(G_{sampled})$ ***or*** $n < min\_nodes$ ***or*** $n > max\_nodes$ ***or*** $e < min\_edges$ ***or*** $e > max\_edges$ **then**
  return $\emptyset$ // Subgraph is invalid
return G_{sampled} // Return valid subgraph

connected by edges of different types, and all connections conform to predefined rules. However, this graph is not yet sufficient to serve as a complete provenance graph. In real-world scenarios, nodes typically carry detailed names, such as command lines or file paths. These names are essential, as many provenance-based intrusion detection methods rely on features extracted from node names using natural language processing techniques [37, 53]. Therefore, assigning meaningful node names is crucial for supporting accurate detection.

In the third stage, we leverage the natural language generation capabilities of large language models to assign names to nodes in provenance graphs. This process involves two main challenges. First, prior studies suggest that LLMs are not inherently effective at understanding graph structures [33]. Thus, a key issue is how to transform the structure of a provenance graph into a format that is more interpretable by LLMs. Second, LLMs lack prior knowledge of typical node names in provenance graphs, and prompting them directly often results in overly simplistic outputs with limited diversity.

To address the first challenge, prior work has shown that LLMs are less effective at processing graph-structured data but perform well on sequence-based inputs [69]. Motivated by this, we propose a method that converts directed heterogeneous graphs into sequences of node and edge types using depth-first search. These sequences are then input to the LLM to generate names for the corresponding nodes. Specifically, to extract complete and non-overlapping DFS paths, we use all nodes with zero in-degree as starting points for DFS traversal. However, due to the presence of self-loops in the provenance graph, some paths may not be captured using this strategy alone. To address this, we additionally include any node involved in a self-loop as a starting point. In each extracted sequence, a node is represented as a pair (τ_i, n_i), where $\tau_i \in \mathcal{T}$ denotes the node type and $n_i \in \mathcal{N}$ the node name. Each edge e_j represents the interaction type between adjacent nodes. A DFS-derived sequence is defined as:

$$S = [(\tau_1, n_1) \xrightarrow{e_1} (\tau_2, n_2) \xrightarrow{e_2} \cdots \xrightarrow{e_{k-1}} (\tau_k, n_k)]$$

For nodes without names, we use [null] as a placeholder. During the name generation process, we do not input all sequences to the LLM in parallel. This is because nodes may appear in multiple sequences, and parallel processing could lead to inconsistencies in assigned names. Instead, we sequentially feed each DFS-derived sequence to the LLM. Once a node name is generated, it is filled into the graph, ensuring that subsequent traversals can retrieve the updated name.

To address the second challenge, we construct a training dataset to provide the LLM with prior knowledge about entity names in provenance data. Each training instance is a question–answer pair, where the question (Q) consists of a DFS sequence of node and edge types, and the answer (A) is the same sequence with node names annotated. All training instances are derived from real-world provenance graphs.

To generate realistic Q&A pairs, we adopt a masking strategy that preserves structural context while requiring the model to infer node names. During the name generation process, two types of sequence are inputted to the LLM. The *first* type consists of sequences in which all nodes are unnamed, serving as a cold start and requiring the LLM to infer node names based solely on the order of node and edge types. The *second* type includes sequences where some nodes are already named, requiring the LLM to infer the remaining names using both structural order and existing node names as context. Accordingly, we define two corresponding Q&A pair formats:

Fully Masked Q&A Pairs In this setting, all node names are masked with [null] while retaining node types and edge types, creating a template for cold-start name generation. This design encourages the model to learn a mapping $f : (\tau_i, \{\tau_j, e_l\}_{j=1}^{k}) \rightarrow n_i$, while preserving structural consistency through fixed sequences of node and edge types.

$$Q_{\text{full}} = [(\tau_1, [\text{null}]) \xrightarrow{e_1} (\tau_2, [\text{null}]) \xrightarrow{e_2} \cdots \xrightarrow{e_{k-1}} (\tau_k, [\text{null}])],$$
$$A_{\text{full}} = [(\tau_1, n_1) \xrightarrow{e_1} (\tau_2, n_2) \xrightarrow{e_2} \cdots \xrightarrow{e_{k-1}} (\tau_k, n_k)].$$

Partially Masked Q&A Pairs To simulate realistic generation scenarios where some node names are known from prior context, we randomly mask a subset $\mathcal{M} \subset \{1, \ldots, k\}$ of node names with a masking rate ρ:

$$Q_{\text{part}} = [(\tau_1, \tilde{n}_1) \xrightarrow{e_1} \cdots \xrightarrow{e_{k-1}} (\tau_k, \tilde{n}_k)],$$

where

$$\tilde{n}_i = \begin{cases} [\text{null}] & \text{if } i \in \mathcal{M}, \\ n_i & \text{otherwise.} \end{cases}$$

The corresponding answer sequence $A_{\text{part}} = A_{\text{full}}$ provides full supervision of node names. This adaptive masking strategy allows the model to learn to predict masked node names based on both the observed names and the structural context:

$$P(n_{\mathcal{M}} \mid n_{\neg\mathcal{M}}, \{\tau_j\}_{j=1}^{k}, \{e_l\}_{l=1}^{k-1}).$$

By constructing these two types of QA pairs, we provide the LLM with a comprehensive training dataset that helps it learn the correspondence between sequences and node names, thereby improving its ability to generate accurate node names in provenance graphs. This approach leverages the strengths of LLMs in sequence-based tasks while addressing the challenges of graph structure understanding, leading to more effective node name generation in provenance graphs.

4.6. Semantic Correctness Validation

Evaluating semantic correctness is essential for assessing the quality of synthetic provenance graphs, as it indicates whether the generated graphs accurately reflect the underlying logic and intent of real-world system behaviors. This evaluation examines whether the relationships among entities and the sequences of interactions conform to meaningful and valid operational patterns. Despite its importance, evaluating semantic correctness remains difficult due to the lack of standardized metrics and the inherent complexity in defining ground-truth semantics for diverse and dynamic system activities.

Model-based Evaluation Mechanism. To evaluate semantic correctness, we formally define a semantic unit in the graph as a triplet: the starting node name s, edge type e, and ending node name t. This unit is represented as (s, e, t), where $s, t \in \mathcal{N}$ (the set of all node names) and $e \in \mathcal{E}$ (the set of all edge types). Concatenating these elements forms a natural language sentence, denoted Sentence(s, e, t). This sentence captures the semantics of the interaction between entities, expressing that the source node s performs an action of type e on the target node t.

To assess the semantic correctness of Sentence(s, e, t), we employ a BERT(Bidirectional Encoder Representations from Transformers) [13]-based encoder to obtain its embedding. This embedding captures the semantic representation of the interaction described by the triplet and serves as the basis for subsequent classification-based evaluation. Formally, let BERT : $\mathcal{S} \to \mathbb{R}^d$ be the encoding function that maps a sentence Sentence$(s, e, t) \in \mathcal{S}$ (the set of all possible semantic triplets in sentence form) to a dense vector $\mathbf{v} \in \mathbb{R}^d$. The embedding is computed as:

$$\mathbf{v} = \text{BERT}(\text{Sentence}(s, e, t))$$

However, embedding of Sentence(s, e, t) captures only pairwise semantic relationships, resulting in a localized evaluation. To achieve graph-level assessment, we employ a multi-layer Graph Attention Network (GAT) [60]. This architecture aggregates neighborhood information through stacked self-attention layers, where each layer updates node representations by adaptively weighting features from adjacent nodes. We employ BERT to represent the text attributes of individual nodes as the initial node embeddings. Formally, for each node v with text attribute T_v, its initial embedding $\mathbf{h}_v^{(0)}$ is derived via BERT:

$$\mathbf{h}_v^{(0)} = \text{BERT}(T_v)$$

Let GAT : $\mathcal{V} \to \mathbb{R}^{d'}$ denote the encoder that maps a node to its global representation in $\mathbb{R}^{d'}$ using these initial embeddings. For a semantic unit (s, e, t), we obtain the global embeddings of both the starting node s and the ending node t:

$$\mathbf{u}_s = \text{GAT}\left(s; \left\{\mathbf{h}_v^{(0)}\right\}_{v\in\mathcal{V}}\right), \quad \mathbf{u}_t = \text{GAT}\left(t; \left\{\mathbf{h}_v^{(0)}\right\}_{v\in\mathcal{V}}\right)$$

We then concatenate the global embeddings $\mathbf{u}_s$ and $\mathbf{u}_t$ with the local embedding $\mathbf{v}$ of the semantic unit to form a comprehensive representation:

$$\mathbf{z} = [\mathbf{v}; \mathbf{u}_s; \mathbf{u}_t]$$

This comprehensive representation $\mathbf{z}$ captures both the local and global context of the semantic unit. Finally, we feed $\mathbf{z}$ into a multilayer perceptron (MLP) discriminator to determine the correctness of the semantic unit. The MLP discriminator is defined as MLP : $\mathbf{R}^{d+2d'} \to [0, 1]$. The correctness score of the semantic unit is computed as:

$$\text{Correctness Score} = \text{MLP}(\mathbf{z})$$

The correctness score is a value between 0 and 1, where a higher score indicates higher semantic correctness.

Contrastive Learning-based Training Framework. To train the GAT and MLP discriminator, we employ a contrastive learning approach. We use real provenance graphs as the training set, extracting all semantic units from them and labeling them as positive samples. Additionally, to provide negative samples for contrastive learning, we construct them from the positive samples using three distinct strategies to help the classifier model learn to distinguish between semantically correct and incorrect instances. The strategies are:

Subject-Object Inversion (SOI): We swap the names of the start node and the end node. This trains the model to recognize cases where the subject and object are inverted in semantics. Formally, for a semantic unit (s, e, t), we generate a negative sample by inverting the subject and object:

$$(s, e, t) \to (t, e, s)$$

Predicate Replacement (PR): We replace the edge type with another type of edge. This trains the model to identify incorrect predicates in semantics. However, in this method,

replacing the edge type with another type may still result in semantically correct cases. To guarantee the constructed semantic units are semantically incorrect, we manually classify edge types into disjoint categories. Let $\mathcal{E} = \bigcup_{i=1}^{k} \mathcal{E}_i$ be the partition of edge types into k categories. When replacing an edge type $e \in \mathcal{E}_i$, we ensure the new edge type e' is from a different category, i.e., $e' \in \mathcal{E}_j$ where $i \neq j$. For a semantic unit (s, e, t), we generate a negative sample by replacing the predicate:

$$(s, e, t) \rightarrow (s, e', t) \quad \text{where } e' \in \mathcal{E}_j \text{ and } j \neq i$$

Entity Substitution (ES): We replace the name of either the start node or the end node with the name of another node. This trains the model to recognize cases where irrelevant entities appear in semantics. It is important to note that in this construction method, replacing one element of a semantic triple with another element may still result in correct semantics. This is because the replaced element and the new element may be similar. Therefore, to construct negative samples, we first cluster the node names using K-means into disjoint clusters. Let $\mathcal{V} = \bigcup_{i=1}^{m} \mathcal{V}_i$ be the partition of node names into m clusters. When replacing a node name $s \in \mathcal{V}_i$ or $t \in \mathcal{V}_i$, we select a node $s' \in \mathcal{V}_j$ or $t' \in \mathcal{V}_j$ where $i \neq j$. For a semantic unit (s, e, t), we generate negative samples by substituting either the subject or the object:

$$\begin{aligned} &(s, e, t) \rightarrow (s', e, t) \quad \text{where } s' \in \mathcal{V}_j \text{ and } j \neq i \\ \text{or} \quad &(s, e, t) \rightarrow (s, e, t') \quad \text{where } t' \in \mathcal{V}_j \text{ and } j \neq i \end{aligned}$$

To maintain a balanced training set, we generate one negative sample for each positive sample in the real graph. Specifically, we randomly select one of the three strategies (SOI, PR, ES) to construct the negative sample. This ensures that positive and negative samples each account for 50% of the training data. For the MLP discriminator, which determines the semantic correctness of each triple, we adopt a binary cross-entropy loss. Given a batch of semantic units $\{(s_i, e_i, t_i)\}_{i=1}^{N}$ with corresponding labels $y_i \in \{0, 1\}$, the loss function is defined as:

$$\mathcal{L}_{\text{MLP}} = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log f_{\text{MLP}}(\mathbf{z}_i) + (1 - y_i) \log \left(1 - f_{\text{MLP}}(\mathbf{z}_i)\right) \right]$$

5. Evaluation

To thoroughly evaluate PROVSYN, we address the following research questions. Experiments on provenance graph structure generation were conducted on a machine equipped with Intel Xeon Silver CPU, 128 GB RAM, NVIDIA RTX 3090 GPUs, and Ubuntu 22.04.4 LTS. Experiments on text attribute synthesizing were performed on a machine with Intel Xeon Platinum CPU, 512 GB RAM, NVIDIA RTX A6000 GPUs, and Ubuntu 22.04.3 LTS. The research questions are:

RQ1: What is the fidelity of synthetic provenance graphs in terms of structural, textual, temporal, and embedding-based characteristics?

TABLE 1: Comparison of dataset statistics.

Dataset	Nodes	Edges	Node Types	Edge Types
Nodlink	16,818	30,531	3	13
Cadets	89,207	276,883	6	20
Theia	77,071	202,517	3	14
Trace	74,241	113,808	7	20

RQ2: What is the semantic correctness of synthetic provenance graphs?

RQ3: Can synthetic provenance graphs mitigate class imbalance in the dataset?

RQ4: When synthetic provenance graphs are used for training set augmentation in downstream intrusion detection tasks, what is the performance of the detection task?

RQ5: What are the time and resource costs of generating synthetic provenance graphs?

Further aspects of our study, including hyperparameter analysis, ablation study are discussed extensively in the Appendix C D.

5.1. Experimental Setup

Datasets. We evaluate the effectiveness of PROVSYN on two public datasets: the Nodlink Ubuntu dataset [37] and the DARPA Engagement 3 datasets [2]. The Nodlink dataset [37] we use is a simulated provenance dataset collected on Ubuntu 20.04 using Sysdig. It provides both benign logs and attack logs containing a complete multi-stage attack chain. The detailed statistics of the Nodlink provenance graphs used for training and evaluation are shown in Table 1. The DARPA E3 datasets [2] are collected from an enterprise network during adversarial engagements as part of the DARPA Transparent Computing program. APT attacks are launched by a red team, while blue teams attempt to detect them through host auditing and causality analysis. TRACE, THEIA and CADETS sub-datasets are included in our evaluation. For E3 dataset, we utilize labels from ThreaTrace. As shown in Table 1, the E3 dataset, compared to Nodlink, has a larger number of nodes and edges, and also exhibits greater diversity in node and edge types.

Baselines. In our approach, we first use a heterogeneous graph generation model to construct the structure of the provenance graph and then employ an LLM to generate the textual attributes of the graph. In contrast, an alternative method directly calls an LLM to synthesize provenance logs and then converts these logs into a graph. This method does not require an additional heterogeneous graph generation model. We systematically evaluate and compare the provenance data synthesized by these strong LLMs with the data synthesized by PROVSYN.

To ensure a comprehensive comparison, we selected baseline models encompassing both large-parameter closed-source models and smaller-parameter open-source models. For large-parameter LLMs, we chose GPT-4o [4], Claude-3.7-Sonnet [1], and Doubao-pro-32k [3]. For smaller-

parameter LLMs, we chose Qwen2.5-7B-Instruct [66] and Llama3.1-8B-Instruct [18].

To generate provenance logs using LLMs listed above, we designed structured prompts that included task descriptions, illustrative examples, and clearly defined output formats. Each dataset was paired with a customized prompt, primarily differing in the examples provided. These examples, formatted as JSON or TSV entries, contained five elements: source node type, source node name, destination node type, destination node name, and edge type. They were selected from real audit logs to ensure both representativeness and coverage of various node and edge types. For each dataset, we collected 100 responses from each LLM. After obtaining the responses, we parsed them line by line and filtered out entries that did not conform to the expected JSON or TSV format. We also discarded entries containing node or edge types outside the predefined type sets. The remaining entries were then used to construct provenance graphs for comparison with graphs generated by PROVSYN.

Configurations. For training the heterogeneous graph generation model, we sampled 20,000 subgraphs from the real provenance graphs using a random walk strategy. Among them, 16,000 subgraphs were used for training, 2,000 for validation, and 2000 for test. The model was trained for 3,000 epochs with a batch size of 32 and a learning rate of 0.003. After training, we selected the model that achieved the best performance on the validation set for use in PROVSYN. For example, on the Theia dataset, the best-performing model was obtained after 2,380 epochs and was used in the final PROVSYN pipeline.

For the text attribute synthesizing task, we used Llama3.2-3B-Instruct as the base model. We constructed a training set of 20,000 samples, consisting of 10,000 fully masked QA pairs and 10,000 partially masked QA pairs. During training, we applied the LoRA fine-tuning method, freezing most of the model parameters and updating only 0.81% of them. The model was trained for 60 steps with a learning rate of 2e-4. During inference, the temperature was set to 1.5, and the maximum number of generated tokens was set to 2048.

5.2. Fidelity Evaluation

Structural Evaluation. To quantitatively evaluate the structural fidelity of generated graphs, we employ five Maximum Mean Discrepancy (MMD)-based metrics [19, 59] that measure distributional divergence between generated graphs and real graphs. These metrics include: (1)*degree MMD* for local connectivity patterns, (2)*orbit MMD* for higher-order structural motifs, (3)*node label MMD* for vertex attribute distributions, (4)*edge label MMD* for edge label distributions, and (5)*joint node label-degree MMD* for attribute-topology interactions.

To bridge the characterization of graph features with distributional divergence measurement, we leverage kernel methods to project these heterogeneous graphs into comparable representations. We construct kernel similarity matrices using two distinct Gaussian kernel formulations. The Earth Mover's Distance-optimized Gaussian kernel is applied to categorical distribution metrics including degree MMD, node/edge label MMD, and joint node label-degree MMD, while the Euclidean-based Gaussian kernel handles continuous feature spaces in orbit MMD. The MMD² values are computed via the unbiased estimator:

$$\text{MMD}^2 = \mathbb{E}_{x,x' \sim P}[k(x, x')] + \mathbb{E}_{y,y' \sim Q}[k(y, y')] - 2\mathbb{E}_{x \sim P, y \sim Q}[k(x, y)]$$

where P and Q denote reference and generated graph distributions, x, x' are samples from P, and y, y' from Q.

TABLE 2: Comparison of MMD metrics across different models and datasets. Lower values indicate better performance.

Dataset	Model	Degree	Orbit	Node Label	Edge Label	Node Label & Degree
Nodlink	GPT-4o	0.75	0.94	0.11	0.75	0.89
	Claude-3.7	0.56	0.85	0.07	0.92	0.77
	Doubao-pro	0.58	0.82	0.18	1.50	0.59
	Qwen2.5-7B	0.43	0.59	0.05	0.94	0.40
	Llama3.1-8B	0.62	-	0.10	0.91	0.81
	PROVSYN	**0.07**	**0.07**	**0.002**	**0.01**	**0.07**
Cadets	GPT-4o	0.82	0.28	0.37	0.90	0.48
	Claude-3.7	0.76	0.36	0.11	1.33	0.83
	Doubao-pro	0.73	**0.25**	0.27	1.31	0.66
	Qwen2.5-7B	0.76	0.39	0.67	0.77	0.33
	Llama3.1-8B	0.88	0.40	0.72	0.85	0.81
	PROVSYN	**0.08**	0.32	**0.003**	**0.01**	**0.12**
Theia	GPT-4o	0.94	**0.35**	0.46	1.34	0.53
	Claude-3.7	0.92	0.85	0.18	1.44	0.81
	Doubao-pro	0.82	0.67	0.46	1.86	0.89
	Qwen2.5-7B	0.91	0.46	0.48	1.00	0.61
	Llama3.1-8B	0.68	0.87	0.53	1.01	0.66
	PROVSYN	**0.16**	0.54	**0.005**	**0.01**	**0.29**
Trace	GPT-4o	0.66	1.04	0.80	1.07	0.54
	Claude-3.7	0.77	1.10	0.68	1.74	0.80
	Doubao-pro	0.75	1.06	0.57	1.12	0.51
	Qwen2.5-7B	0.24	0.23	0.44	1.06	0.47
	Llama3.1-8B	0.25	0.87	1.04	1.16	0.63
	PROVSYN	**0.17**	**0.23**	**0.02**	**0.04**	**0.33**

In the experimental results shown in Table 2, PROVSYN consistently outperformed the baselines across all datasets. In Nodlink dataset, PROVSYN achieved values an order of magnitude lower than the baselines on all metrics, indicating superior structural fidelity. In Cadets, Theia, and Trace datasets, PROVSYN also maintained a clear lead, particularly on label-related metrics. In contrast, baseline methods exhibit inconsistent performance patterns across the four datasets, suggesting fundamental limitations in structural generation capabilities regardless of the scale of model parameters. This finding underscores the necessity of an additional heterogeneous graph generation network, which produces structurally superior graphs compared to those generated solely by an LLM.

Texutual Evaluation. To evaluate the accuracy of generated text attributes, we use standard NLP metrics that capture different types of similarity. These include: (1) *BLEU* [51], which measures n-gram precision; (2) *GLEU* [49], which balances precision and recall; and (3) *ROUGE* [40], which focuses on recall by assessing how much of the reference text is covered.

For each node type, we construct a reference corpus $\mathcal{R}_t = \{r_1, r_2, ..., r_n\}$ by aggregating all valid node names

of type t from the real-word provenance graph. When evaluating a generated name g_i, we compute its similarity with every reference $r_j \in \mathcal{R}_t$ through parallel metric calculations, then select the maximum similarity score as the final assessment:

$$S(g_i) = \max_{r_j \in \mathcal{R}_t} \text{sim}(g_i, r_j)$$

The choice of maximum similarity over average similarity is driven by the following considerations. Focusing on the highest match ensures that the generated text is evaluated based on its optimal alignment with the reference corpus, thereby highlighting its capacity to produce accurate and relevant attributes. The complete procedure is presented in Algorithm 2 in Appendix B.

In our experiments, we used the Python NLTK library to compute the *BLEU*, *GLEU*, and *ROUGE* scores for the process and file nodes. We didn't evaluate the text quality of network nodes because their names are purely numerical, making natural language metrics inappropriate. Additionally, our analysis shows that the network node names generated by both the baselines and PROVSYN conform to the IPv4 address format.

The results in Table 3 show that the node names generated by the baselines generally achieve scores below 0.1, indicating that large language models lack prior knowledge of the names of operating system entities. In contrast, PROVSYN was exposed to real entity names from audit logs during the finetuning stage, enabling a smaller 3B model to generate more accurate and realistic names. Interestingly, we observed that Llama-8B performed relatively well in the Cadets datasets. Further analysis revealed that many of the node names generated by Llama were directly copied from the examples in the prompt, even though the corresponding log entries did not make sense.

Temporal Evaluation. A provenance graph is a directed graph in which directed sequences also represent temporal sequences of events occurring in chronological order. For synthetic provenance graphs, accurately preserving the temporal structure of event sequences is essential for ensuring fidelity. To evaluate the temporal consistency of synthetic provenance graphs, we employ two metrics: *Longest Common Subsequence (LCS)* [7] and *Dynamic Time Warping (DTW)* [48].

LCS measures the length of the longest sequence of events that appears in both the real and synthetic graphs in the same order. LCS is a strict matching metric—it requires exact alignment of events and their temporal order without allowing for any deviations or shifts. *DTW*, on the other hand, is a more flexible metric that allows for non-linear alignments between sequences. Unlike strict matching methods, DTW allows sequences to be stretched or compressed along the time axis, thereby accommodating local variations. This flexibility makes DTW particularly suitable for scenarios where the synthetic graph captures the general pattern of the real sequence but exhibits slight shifts or inconsistencies.

In our experiments, we used depth-first search to extract sequences from the graphs. Each sequence consisted of node types and edge types, e.g. `(process, clone, process, read, file)`. For LCS computation, we used a dynamic programming approach, and for DTW, we used the `dtw` library in Python. To compare synthetic graphs with the real graphs in the test set, we adopted a strategy similar to Algorithm 2 in Appendix B: each sequence from the synthetic graph was compared with all sequences from the real graph, and the best match was recorded as its score. The overall graph score was then computed by aggregating the scores of all sequences in the synthetic graph. The results in Table 4 show that PROVSYN achieves higher LCS scores and lower DTW values than the baselines across all four datasets, indicating its advantage in preserving temporal consistency.

Graph Embedding Evaluation. To evaluate the fidelity of synthetic graphs from an embedding perspective, we employ graph representation learning techniques to project both synthetic and real graphs into a shared vector space, and measure their similarity using cosine similarity. Specifically, we adopt two representative embedding methods: *DeepWalk* [52] and *Doc2Vec* [35].

DeepWalk is a foundational algorithm in graph representation learning. It learns node embeddings by simulating truncated random walks on the graph and treating the resulting node sequences as sentences for a language modeling task. Given a graph G, we perform N random walks of length L starting from each node, generating a corpus $\mathcal{W} = \{w_1, w_2, \ldots, w_N\}$, where each w_i is a walk sequence of node identifiers. These sequences are then used to train a Word2Vec model, producing node-level embeddings. The graph-level embedding $\mathbf{v}_{\text{deepwalk}}$ is computed as the mean of all node embeddings:

$$\mathbf{v}_{\text{deepwalk}} = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \text{deepwalk}(v)$$

Notably, *DeepWalk* does not consider textual attributes of nodes.

Unlike *DeepWalk*, *Doc2Vec* leverages node name information to capture the textual semantics of the graph. It treats each graph as a document, which is constructed from a collection of random walks. For a given graph G, we perform N random walks of length L, resulting in a sequence set $\mathcal{W} = \{w_1, w_2, \ldots, w_N\}$, where each w_i is a sequence of node names. These sequences are used to train a Doc2Vec model, yielding a graph-level embedding $\mathbf{v}_{\text{doc2vec}}$:

$$\mathbf{v}_{\text{doc2vec}} = \text{doc2vec}(\mathcal{W})$$

To assess the similarity between a synthetic graph G_s and a real graph G_r, we compute the cosine similarity between their respective embeddings:

$$\text{Similarity} = \frac{\mathbf{v}_{G_s} \cdot \mathbf{v}_{G_r}}{\|\mathbf{v}_{G_s}\| \|\mathbf{v}_{G_r}\|}$$

TABLE 3: Comparison of textual quality across different models, types and datasets. Lower values indicate better performance.

Dataset	Type	Model	BLEU	GLEU	ROUGE-L-F
Nodlink	Process	GPT-4o	0.05	0.11	0.23
		Claude-3.7	0.08	0.15	0.27
		Doubao-pro	0.02	0.03	0.09
		Qwen2.5-7B	0.07	0.15	0.29
		Llama3.1-8B	0.12	0.17	0.33
		PROVSYN	**0.89**	**1.00**	**0.99**
	File	GPT-4o	0.09	0.11	0.29
		Claude-3.7	0.10	0.11	0.29
		Doubao-pro	0.06	0.11	0.25
		Qwen2.5-7B	0.11	0.17	0.41
		Llama3.1-8B	0.26	0.26	0.37
		PROVSYN	**0.31**	**0.32**	**0.44**
Cadets	Process	GPT-4o	0.24	0.24	0.24
		Claude-3.7	0.17	0.17	0.17
		Doubao-pro	0.03	0.03	0.03
		Qwen2.5-7B	0.03	0.03	0.04
		Llama3.1-8B	1.00	1.00	1.00
		PROVSYN	**1.00**	**1.00**	**1.00**
	File	GPT-4o	0.01	0.01	0.20
		Claude-3.7	0.03	0.03	0.14
		Doubao-pro	0.01	0.01	0.14
		Qwen2.5-7B	0.02	0.02	0.16
		Llama3.1-8B	0.00	0.00	0.03
		PROVSYN	**0.31**	**0.32**	**0.60**
Theia	Process	GPT-4o	0.08	0.08	0.09
		Claude-3.7	0.04	0.04	0.05
		Doubao-pro	0.01	0.01	0.03
		Qwen2.5-7B	0.19	0.19	0.20
		Llama3.1-8B	0.00	0.00	0.00
		PROVSYN	**0.76**	**0.89**	**0.47**
	File	GPT-4o	0.03	0.03	0.21
		Claude-3.7	0.05	0.05	0.16
		Doubao-pro	0.01	0.01	0.11
		Qwen2.5-7B	0.00	0.00	0.29
		Llama3.1-8B	0.00	0.00	0.35
		PROVSYN	**0.37**	**0.44**	**0.48**
Trace	Process	GPT-4o	0.02	0.02	0.02
		Claude-3.7	0.02	0.02	0.02
		Doubao-pro	0.00	0.00	0.00
		Qwen2.5-7B	0.02	0.02	0.02
		Llama3.1-8B	0.00	0.00	0.00
		PROVSYN	**0.50**	**0.50**	**0.50**
	File	GPT-4o	0.00	0.00	0.03
		Claude-3.7	0.00	0.00	0.03
		Doubao-pro	0.00	0.00	0.01
		Qwen2.5-7B	0.00	0.00	0.02
		Llama3.1-8B	0.00	0.00	0.00
		PROVSYN	0.00	0.00	**0.67**

A higher similarity score indicates that the synthetic graph exhibits greater fidelity to the real graph in the embedding space.

For DeepWalk-based similarity, we use the Word2Vec model from the Gensim library, and for Doc2Vec-based similarity, we use Gensim's Doc2Vec model. In our experiments results shown in Table 4, PROVSYN outperforms the baselines on the Nodlink, Cadets, and Trace datasets, while performing slightly worse than GPT and Claude on Theia. The results show that PROVSYN maintains stable embedding similarity, with DeepWalk scores above 0.80 and Doc2Vec scores above 0.65 across four datasets. In comparison, the baselines exhibit greater variation across different datasets.

TABLE 4: Comparison of Temporal and Embedding metrics across different datasets and models. For LCS, higher is better; for DTW, lower is better; for DeepWalk and Doc2Vec, higher is better.

Dataset	Model	Temporal Metrics		Embedding Metrics	
		LCS	DTW	DeepWalk	Doc2Vec
Nodlink	GPT-4o	2.44	1.54	0.66	0.69
	Claude-3.7	3.03	1.33	0.83	0.40
	Doubao-pro	2.64	0.89	0.60	0.59
	Qwen2.5-7B	2.84	0.56	0.58	0.32
	Llama3.1-8B	2.88	0.83	0.50	0.16
	PROVSYN	**3.30**	**0.41**	**0.89**	**0.74**
Cadets	GPT-4o	2.99	0.48	0.70	0.40
	Claude-3.7	2.87	1.20	0.73	0.39
	Doubao-pro	2.68	3.62	0.64	0.71
	Qwen2.5-7B	2.13	24.30	0.79	0.25
	Llama3.1-8B	2.96	0.48	0.48	0.67
	PROVSYN	**3.01**	**0.21**	**0.90**	**0.82**
Theia	GPT-4o	2.39	0.65	**0.89**	0.53
	Claude-3.7	1.80	0.74	0.82	**0.70**
	Doubao-pro	2.78	1.43	0.56	0.39
	Qwen2.5-7B	1.36	8.90	0.74	0.48
	Llama3.1-8B	1.04	8.94	0.39	0.66
	PROVSYN	**3.02**	**0.00**	0.81	0.65
Trace	GPT-4o	2.16	0.46	0.70	0.27
	Claude-3.7	2.15	0.54	0.67	0.41
	Doubao-pro	2.09	0.40	0.57	0.08
	Qwen2.5-7B	2.10	0.47	0.75	0.09
	Llama3.1-8B	2.08	4.45	0.71	-0.01
	PROVSYN	**2.80**	**0.10**	**0.81**	**0.75**

5.3. Semantic Correctness Evaluation

We use the semantic correctness validation mechanism introduced in Section 4.6 to evaluate the semantic of synthetic graphs. In the experiment, the GAT model was trained for 2500 epochs with a learning rate of 0.0001, while the MLP classifier was trained for 1500 epochs with the same learning rate. To validate the effectiveness and interpretability of our semantic accuracy evaluation, we tested it on real-world provenance graphs from the same datasets but different from the training graphs. The results show that the semantic accuracy of these real graphs exceeds 90% across the Nodlink, Cadets, Theia, and Trace datasets. This supports the validity of our semantic accuracy evaluation method. The results in Figure 2 show that the provenance graphs generated by PROVSYN consistently outperform the baselines in semantic accuracy across all four datasets, showing robustness to differences in data distribution. In contrast, the baselines exhibit substantial fluctuations in semantic accuracy across datasets. Notably, in the more challenging Theia and Trace datasets, the semantic accuracy of baseline-generated graphs remained below 0.20, whereas PROVSYN achieved 0.57 and 0.82, respectively. These results demonstrate the robustness of PROVSYN in handling complex, low signal-to-noise provenance data.

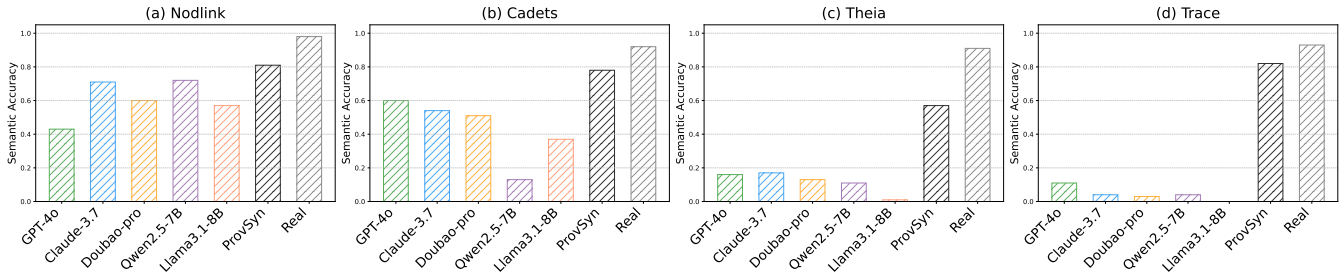


Figure 2: Comparison of semantic accurary across different models and datasets. Higher values indicate better performance.

5.4. Data Imbalance Evaluation

To evaluate whether the inclusion of PROVSYN-generated synthetic graphs mitigates class imbalance, we use two metrics—*entropy* and the *Gini index*—to quantify the distributional balance of node and edge labels. Given a label distribution with probabilities $p_1, p_2, \ldots, p_n$, the two metrics are defined as:

$$\text{Entropy} = -\sum_{i=1}^{n} p_i \log(p_i), \quad \text{Gini} = 1 - \sum_{i=1}^{n} p_i^2$$

Higher values of both metrics indicate a more uniform and balanced label distribution.

We conduct experiments on the Nodlink, Cadets, Theia, and Trace datasets, both in their original data and after incorporating PROVSYN-generated data. For each dataset, we add 100 synthetic graphs merged with the original graph as graph communities. The results are summarized in Table 5. As shown, the inclusion of PROVSYN increases both entropy and Gini index for node and edge labels across all datasets. In the Nodlink dataset, the metrics improve significantly due to the dataset's small scale. These results suggest that PROVSYN contributes to a more balanced label distribution by enriching underrepresented classes in provenance graph datasets.

To evaluate whether the synthesized graphs extend beyond the training data distribution and contain novel structures that cover more provenance scenarios, we use two metrics: *Novelty* and *Uniqueness*. Novelty is computed by verifying that each generated graph is neither a subgraph of any reference graph in the training set nor contains any reference graph as a subgraph, indicating whether novel structures exist in the synthesized graphs. Uniqueness measures the proportion of generated graphs that are structurally distinct from one another, reflecting the diversity of the generated graphs. As shown in Table 6, the graphs generated by PROVSYN achieve 100% novelty and 100% uniqueness across all four datasets. This indicates that the synthetic data exhibits strong diversity and effectively captures patterns that were underrepresented or missing in the original data.

5.5. Application in IDS

To demonstrate the practical utility of PROVSYN, we focus on the task of APT attack detection. We conduct experiments using two detection algorithms, Magic[31] and Nodlink[37]. Specifically, we augment the training set with synthesized provenance graphs and then observe whether the detection performance improves or declines during testing.

For the Nodlink algorithm, we enhanced its dataset by adding 50 synthetic graphs merged with the original provenance graph as graph communities. This resulted in a 15% increase in node counts. For the Magic algorithm, we enhanced the Cadets, Theia, and Trace datasets. Due to their larger scale, we added 300 synthetic graphs merged into their original provenance graphs. This led to a 10% to 15% increase in node counts.

To evaluate the effectiveness of synthetic graphs in intrusion detection tasks, we maintain identical training configurations before and after incorporating synthetic data. The detection performance is presented in Table 7. In addition to Precision and Recall, we report the number of False Positives (#FP), an crucial metric for intrusion detection system, as a high false positive rate can increase the workload of subsequent security analysis. As shown in the results, on the Nodlink dataset, the inclusion of synthetic graphs leads to no change in Precision, Recall, or the number of false

TABLE 5: Comparison of node and edge label entropy and Gini index across different datasets with and without synthetic data. Higher values indicate better performance.

Dataset	Data Source	Node Label		Edge Label	
		Entropy	Gini Index	Entropy	Gini Index
Nodlink	*Real*	1.03	0.42	2.41	0.75
	+PROVSYN	1.31 (+0.28)	0.55 (+0.13)	2.63 (+0.22)	0.82 (+0.07)
Cadets	*Real*	1.11	0.44	1.84	0.52
	+PROVSYN	1.31 (+0.20)	0.53 (+0.09)	1.92 (+0.08)	0.54 (+0.02)
Theia	*Real*	1.31	0.53	1.33	0.39
	+PROVSYN	1.41 (+0.10)	0.58 (+0.05)	1.36 (+0.03)	0.39 (+0.00)
Trace	*Real*	1.52	0.57	1.57	0.45
	+PROVSYN	1.62 (+0.10)	0.61 (+0.04)	1.61 (+0.04)	0.47 (+0.02)

TABLE 6: Evaluation of PROVSYN-generated graphs in terms of Novelty and Uniqueness across different datasets. Higher values indicate better performance.

Dataset	Novelty (%)	Uniqueness (%)
Nodlink	100.0	100.0
Cadets	100.0	100.0
Theia	100.0	100.0
Trace	100.0	100.0

positives, indicating that the synthetic data is of comparable quality to the original data. On the Cadets, Theia, and Trace datasets, Recall remains constant, and Precision improves. Specifically, false positives decrease by 506 on Cadets (from 6435 to 5929), by 95 on Theia (from 931 to 836), and by 379 on Trace (from 2390 to 2011). These results suggest that augmenting with synthetic graphs produced by PROVSYN improves model performance, and effectively reduces false positives in detection systems.

TABLE 7: IDS performance across different datasets with and without synthetic data. Higher values indicate better performance.

Dataset	Data Source	Precision(%)	Recall(%)	#FP
Nodlink	*Real*	100.00	100.00	0
	+ PROVSYN	100.00	100.00	0
Cadets	*Real*	66.58	99.79	6435
	+ PROVSYN	68.37	99.77	5929
Trace	*Real*	96.61	99.98	2390
	+ PROVSYN	97.13	99.98	2011
Theia	*Real*	96.45	99.99	931
	+ PROVSYN	96.80	99.99	836

5.6. Overhead Analysis

TABLE 8: Time and memory requirements of PROVSYN's core modules.

Module	Phase	Time	Memory
Graph Generation	Training	18 hours	0.84 GB
	Inference	28.09 s	0.38 GB
Text Generation	Training	109 s	3.44 GB
	Inference	2.78 hours	28 GB

In this section, we analyze the time and memory requirements of PROVSYN (see Table 8). The most time- and resource-intensive components of the PROVSYN framework are the heterogeneous graph generation module and the text attribute generation module.

For the heterogeneous graph generation module, we use 16,000 samples for training, with a batch size of 32 and 3,000 training epochs. The average training time is 18 hours, with a GPU memory usage of 0.84 GB. Inference for generating 1,000 graphs takes 28.09 seconds and consumes 0.38 GB of memory.

For the text attribute generation module, we adopt Llama3.2-3B-Instruct as the base model, applying the LoRA algorithm with 4-bit quantization. The FastLanguageModel library is used to accelerate training. The training consists of 60 steps, takes an average of 109 seconds, and requires a peak memory of 3.44 GB. During inference, generating text attributes for a single graph takes an average of 80 seconds. In our experiments, we deploy 8 fine-tuned LLMs in parallel to synthesize 1,000 graphs, completing the process in 2.78 hours. The total GPU memory required for the 8 models is 28 GB.

This analysis demonstrates that PROVSYN is a time- and memory-efficient framework for provenance graph synthesis.

6. Discussion and Limitations

Temporal Modeling. A provenance graph is not only a directed heterogeneous graph with textual attributes, but also a temporal graph in which each event is associated with a timestamp indicating its occurrence time in the system. These temporal information has recently been leveraged by downstream intrusion detection systems [53] to better capture the patterns of benign and anomalous behaviors. However, in PROVSYN, we do not synthesize explicit timestamps for events. Instead, we approximate temporal order by relying on the sequence of directed edges. Future work may explore modeling temporal dynamics using temporal graph generation networks [75], or designing strategies that leverage LLMs to generate realistic timestamps, thereby enabling more comprehensive and temporally coherent provenance graph synthesis.

Graph Scale. In PROVSYN, the generated provenance graphs are smaller in scale compared to real-world provenance graphs. To enable data augmentation, we embed multiple synthetic graphs as distinct communities within large real-world provenance graphs. The results show that this approach effectively mitigates data imbalance and reduces the false positive rate of detection models trained on the augmented datasets. Future work can focus on scaling up synthetic provenance graphs by enhancing graph generation models for large-scale synthesis and exploring graph merging algorithms that combine smaller graphs into larger ones while preserving structural fidelity.

Efficiency of Text Attribute Synthesis. During textual attribute synthesis, overlaps among nodes in different sequences generated by DFS can lead to naming conflicts when these sequences are processed in parallel to generate node names. To address this issue, we serialize the sequences and input them into the LLM sequentially. The generated node names from earlier sequences are then used as context for subsequent sequences. This approach is effective for small-scale graphs. However, the synthesis efficiency decreases for large-scale graphs. Future work may explore strategies to partition large graphs into batches of non-overlapping sequences, enabling parallel processing while avoiding node naming conflicts.

Application Scenarios. To date, the graphs synthesized by PROVSYN have demonstrated utility within the domain of provenance graph datasets and for detection algorithms targeting APT attacks. However, PROVSYN is a general framework capable of synthesizing high-fidelity heterogeneous graphs with textual attributes. This gap between PROVSYN's inherent capabilities and current applications presents an opportunity for future work: expanding the exploration of PROVSYN's capabilities to encompass a wider range of application scenarios and downstream algorithms.

7. Conclusion

This paper proposes PROVSYN, a novel provenance graph generation framework consisting of three stages: heterogeneous graph structure generation, rule-based topological refinement, and LLM-based synthesis of textual attributes. To evaluate the fidelity of the generated graphs, we assess structural, textual, temporal, embedding, and semantic aspects. Compared with provenance data directly synthesized by LLMs, PROVSYN consistently produces graphs with higher fidelity. When integrated into existing datasets, PROVSYN-generated graphs help mitigate class imbalance and lead to reduced false positive rates in detection models trained on the augmented data.

References

[1] Claude-3.7-sonnet. URL https://claude.ai/. (2025, May 27).

[2] Darpa transparent computing program engagement 3 data release. URL https://github.com/darpa-i2o/Transparent-Computing. (2025, May 27).

[3] Doubao-pro. URL http://doubao.com. (2025, May 27).

[4] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[5] T. J. Anande and M. S. Leeson. Generative adversarial networks (gans): a survey of network traffic generation. *International Journal of Machine Learning and Computing*, 12(6):333–343, 2022.

[6] M. M. Anjum, S. Iqbal, and B. Hamelin. Analyzing the usefulness of the darpa optc dataset in cyber threat detection research. In *Proceedings of the 26th ACM symposium on access control models and technologies*, pages 27–32, 2021.

[7] L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000*, pages 39–48. IEEE, 2000.

[8] M. Buda, A. Maki, and M. A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural networks*, 106:249–259, 2018.

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[10] Z. Chen, K. Liu, Q. Wang, J. Liu, W. Zhang, K. Chen, and F. Zhao. Mindsearch: Mimicking human minds elicits deep ai searcher. *arXiv preprint arXiv:2407.20183*, 2024.

[11] A. Cheng. Pac-gan: Packet generation of network traffic using generative adversarial networks. In *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 0728–0734. IEEE, 2019.

[12] N. De Cao and T. Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL https://arxiv.org/abs/1810.04805.

[14] N. Ding, Y. Chen, B. Xu, Y. Qin, Z. Zheng, S. Hu, Z. Liu, M. Sun, and B. Zhou. Enhancing chat language models by scaling high-quality instructional conversations. *arXiv preprint arXiv:2305.14233*, 2023.

[15] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, H. Wang, and H. Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2:1, 2023.

[16] F. Gilardi, M. Alizadeh, and M. Kubli. Chatgpt outperforms crowd workers for text-annotation tasks. *Proceedings of the National Academy of Sciences*, 120(30):e2305016120, 2023.

[17] N. Goyal, H. V. Jain, and S. Ranu. Graphgen: A scalable approach to domain-agnostic labeled graph generation. In *Proceedings of The Web Conference 2020*, pages 1253–1263, 2020.

[18] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[19] A. Gretton, K. Borgwardt, M. Rasch, B. Schölkopf, and A. Smola. A kernel method for the two-sample-problem. *Advances in neural information processing systems*, 19, 2006.

[20] S. Gunasekar, Y. Zhang, J. Aneja, C. C. T. Mendes, A. Del Giorno, S. Gopi, M. Javaheripi, P. Kauffmann, G. de Rosa, O. Saarikivi, et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

[21] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer. Unicorn: Runtime provenance-based detector for advanced persistent threats. *arXiv preprint arXiv:2001.01525*, 2020.

[22] X. Han, X. Yu, T. Pasquier, D. Li, J. Rhee, J. Mickens, M. Seltzer, and H. Chen. {SIGL}: Securing software installations through deep graph learning. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2345–2362, 2021.

[23] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates. Nodoze: Combatting threat alert fatigue with automated provenance triage. In *network and distributed systems security symposium*, 2019.

[24] W. U. Hassan, A. Bates, and D. Marino. Tactical provenance analysis for endpoint detection and response systems. In *2020 IEEE symposium on security and privacy (SP)*, pages 1172–1189. IEEE, 2020.

[25] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[26] X. He, Z. Lin, Y. Gong, A. Jin, H. Zhang, C. Lin, J. Jiao, S. M. Yiu, N. Duan, W. Chen, et al. Annollm: Making large language models to be better crowdsourced annotators. *arXiv preprint arXiv:2303.16854*, 2023.

[27] O. Honovich, T. Scialom, O. Levy, and T. Schick. Unnatural instructions: Tuning language models with (almost) no human labor. *arXiv preprint arXiv:2212.09689*, 2022.

[28] M. N. Hossain, S. Sheikhi, and R. Sekar. Combating dependence explosion in forensic analysis using alternative tag propagation semantics. In *2020 IEEE symposium on security and privacy (SP)*, pages 1139–1155. IEEE, 2020.

[29] M. A. Inam, Y. Chen, A. Goyal, J. Liu, J. Mink, N. Michael, S. Gaur, A. Bates, and W. U. Hassan. Sok: History is a vast early warning system: Auditing the provenance of system intrusions. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2620–2638. IEEE, 2023.

[30] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE transactions on neural networks and learning systems*, 33(2): 494–514, 2021.

[31] Z. Jia, Y. Xiong, Y. Nan, Y. Zhang, J. Zhao, and M. Wen. {MAGIC}: Detecting advanced persistent threats via masked graph representation learning. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 5197–5214, 2024.

[32] K. Jiang, W. Wang, A. Wang, and H. Wu. Network intrusion detection combined hybrid sampling with deep hierarchical network. *IEEE access*, 8:32464–32476, 2020.

[33] B. Jin, G. Liu, C. Han, M. Jiang, H. Ji, and J. Han. Large language models on graphs: A comprehensive survey. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[34] J. M. Johnson and T. M. Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of big data*, 6(1):1–54, 2019.

[35] J. H. Lau and T. Baldwin. An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv preprint arXiv:1607.05368*, 2016.

[36] N. Lee, W. Ping, P. Xu, M. Patwary, P. N. Fung, M. Shoeybi, and B. Catanzaro. Factuality enhanced language models for open-ended

text generation. *Advances in Neural Information Processing Systems*, 35:34586–34599, 2022.

[37] S. Li, F. Dong, X. Xiao, H. Wang, F. Shao, J. Chen, Y. Guo, X. Chen, and D. Li. Nodlink: An online system for fine-grained apt attack detection and investigation. *arXiv preprint arXiv:2311.02331*, 2023.

[38] Y. Li, S. Bubeck, R. Eldan, A. Del Giorno, S. Gunasekar, and Y. T. Lee. Textbooks are all you need ii: phi-1.5 technical report. *arXiv preprint arXiv:2309.05463*, 2023.

[39] Z. Li, X. Cheng, L. Sun, J. Zhang, and B. Chen. A hierarchical approach for advanced persistent threat detection with attention-based graph neural networks. *Security and Communication Networks*, 2021 (1):9961342, 2021.

[40] C.-Y. Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[41] F. Liu, Y. Wen, D. Zhang, X. Jiang, X. Xing, and D. Meng. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1777–1794, 2019.

[42] L. Liu, P. Wang, J. Lin, and L. Liu. Intrusion detection of imbalanced network traffic based on machine learning and deep learning. *IEEE access*, 9:7550–7563, 2020.

[43] R. Liu, J. Wei, F. Liu, C. Si, Y. Zhang, J. Rao, S. Zheng, D. Peng, D. Yang, D. Zhou, et al. Best practices and lessons learned on synthetic data. *arXiv preprint arXiv:2404.07503*, 2024.

[44] X.-Y. Liu, J. Wu, and Z.-H. Zhou. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):539–550, 2008.

[45] Y. Liu, M. Zhang, D. Li, K. Jee, Z. Li, Z. Wu, J. Rhee, and P. Mittal. Towards a timely causality analysis for enterprise security. In *NDSS*, 2018.

[46] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan. Holmes: real-time apt detection through correlation of suspicious information flows. In *2019 IEEE symposium on security and privacy (SP)*, pages 1137–1152. IEEE, 2019.

[47] R. Mohammed, J. Rawashdeh, and M. Abdullah. Machine learning with oversampling and undersampling techniques: overview study and experimental results. In *2020 11th international conference on information and communication systems (ICICS)*, pages 243–248. IEEE, 2020.

[48] M. Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[49] A. Mutton, M. Dras, S. Wan, and R. Dale. Gleu: Automatic evaluation of sentence-level fluency. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 344–351, 2007.

[50] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599, 2024.

[51] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[52] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[53] M. U. Rehman, H. Ahmadi, and W. U. Hassan. Flash: A comprehensive approach to intrusion detection via provenance graph representation learning. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3552–3570. IEEE, 2024.

[54] M. Rigaki and S. Garcia. Bringing a gan to a knife-fight: Adapting malware communication to avoid detection. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 70–75. IEEE, 2018.

[55] M. Ring, D. Schlör, D. Landes, and A. Hotho. Flow-based network traffic generation using generative adversarial networks. *Computers & Security*, 82:156–172, 2019.

[56] B. Samanta, A. De, G. Jana, V. Gómez, P. Chattaraj, N. Ganguly, and M. Gomez-Rodriguez. Nevae: A deep generative model for molecular graphs. *Journal of machine learning research*, 21(114):1–33, 2020.

[57] S. Sudalairaj, A. Bhandwaldar, A. Pareja, K. Xu, D. D. Cox, and A. Srivastava. Lab: Large-scale alignment for chatbots. *arXiv preprint arXiv:2403.01081*, 2024.

[58] Z. Sun, Y. Shen, Q. Zhou, H. Zhang, Z. Chen, D. Cox, Y. Yang, and C. Gan. Principle-driven self-alignment of language models from scratch with minimal human supervision. *Advances in Neural Information Processing Systems*, 36:2511–2565, 2023.

[59] R. Thompson, B. Knyazev, E. Ghalebi, J. Kim, and G. W. Taylor. On evaluation metrics for graph generative models. *arXiv preprint arXiv:2201.09871*, 2022.

[60] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

[61] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter, et al. You are what you do: Hunting stealthy malware via data provenance analysis. In *NDSS*, 2020.

[62] S. Wang, Z. Wang, T. Zhou, H. Sun, X. Yin, D. Han, H. Zhang, X. Shi, and J. Yang. 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.

[63] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*, 2022.

[64] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[65] C. Xu, Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, and D. Jiang. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*, 2023.

[66] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

[67] J. Yang, K. Zhou, Y. Li, and Z. Liu. Generalized out-of-distribution detection: A survey. *International Journal of Computer Vision*, 132 (12):5635–5662, 2024.

[68] J. Ye, J. Gao, J. Feng, Z. Wu, T. Yu, and L. Kong. Progen: Progressive zero-shot dataset generation via in-context feedback. *arXiv preprint arXiv:2210.12329*, 2022.

[69] R. Ye, C. Zhang, R. Wang, S. Xu, and Y. Zhang. Language is all a graph needs. *arXiv preprint arXiv:2308.07134*, 2023.

[70] K. M. Yoo, D. Park, J. Kang, S.-W. Lee, and W. Park. Gpt3mix: Leveraging large-scale language models for text augmentation. *arXiv preprint arXiv:2104.08826*, 2021.

[71] J. You, B. Liu, Z. Ying, V. Pande, and J. Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *Advances in neural information processing systems*, 31, 2018.

[72] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pages 5708–5717. PMLR, 2018.

[73] Y. Yu, Y. Zhuang, J. Zhang, Y. Meng, A. J. Ratner, R. Krishna, J. Shen, and C. Zhang. Large language model as attributed training data generator: A tale of diversity and bias. *Advances in Neural Information Processing Systems*, 36:55734–55784, 2023.

[74] J. Zengy, X. Wang, J. Liu, Y. Chen, Z. Liang, T.-S. Chua, and Z. L.

Chua. Shadewatcher: Recommendation-guided cyber threat analysis using system audit records. In *2022 IEEE symposium on security and privacy (SP)*, pages 489–506. IEEE, 2022.

[75] L. Zhang, L. Zhao, S. Qin, and D. Pfoser. Tg-gan: Continuous-time temporal graph generation with deep generative models. *arXiv preprint arXiv:2005.08323*, 2020.

[76] T. Zhao, X. Zhang, and S. Wang. Graphsmote: Imbalanced node classification on graphs with graph neural networks. In *Proceedings of the 14th ACM international conference on web search and data mining*, pages 833–841, 2021.

[77] Y. Zhou, M. Kantarcioglu, and C. Clifton. On improving fairness of ai models with synthetic minority oversampling techniques. In *Proceedings of the 2023 SIAM international conference on data mining (SDM)*, pages 874–882. SIAM, 2023.

Appendix A. Entity and Event Distribution in Our Datasets

We present the distributions of *entity type* and *event type* across four commonly-used public provenance datasets in Figure 3. The distributions exhibit a typical long-tail pattern and reveal significant data imbalance.

Appendix B. Textual Quality Evaluation

In Algorithm 2, we compute the maximum similarity score between the text attribute of each node and the reference corpus. This metric serves to evaluate the model's maximum potential in generating accurate and relevant attributes.

Algorithm 2: TEXTUAL QUALITY EVALUATION

Input : Generated graph nodes $V_{\text{gen}} = \{g_1, g_2, ..., g_m\}$ with predicted names and types;
Real graph nodes $V_{\text{real}} = \{r_1, r_2, ..., r_n\}$ with ground-truth names and types;
Node type set $\mathcal{T} = \{t_1, t_2, ..., t_k\}$
Output: Final textual quality score S_{final}

foreach $t \in \mathcal{T}$ **do**
  $\mathcal{R}_t \leftarrow \{r_j \mid r_j \in V_{\text{real}}, \text{type}(r_j) = t\}$ `// Construct reference corpus`
$S_{\text{total}} \leftarrow 0$
$count \leftarrow 0$
foreach $g_i \in V_{gen}$ **do**
  $t \leftarrow \text{type}(g_i)$
  $max\_sim \leftarrow 0$
  foreach $r_j \in \mathcal{R}_t$ **do**
    $sim \leftarrow \text{metric}(g_i.\text{name}, r_j.\text{name})$
    if $sim > max\_sim$ **then**
      $max\_sim \leftarrow sim$
  $S_{\text{total}} \leftarrow S_{\text{total}} + max\_sim$
  $count \leftarrow count + 1$
$S_{\text{final}} \leftarrow S_{\text{total}} / count$ `// Aggregate final score`

return S_{final}

Appendix C. Hyperparameters Setting

Hidden size is of LSTM for heterogeneous graph generation. Hidden size is a key parameter for LSTM, as it determines the capacity of the model to capture temporal and structural dependencies in sequential node and edge generation. We conducted experiments on the Nodlink dataset and evaluated structural fidelity using five MMD-based metrics in the validation set. The results in Figure 4 show that a hidden size of 64 yields significantly lower fidelity compared to the other settings. For hidden sizes of 128, 256, and 512, the MMD values remain at a comparable level, with 256 performing slightly better than 128 and 512. Considering both performance and computational cost, we select 256 as the hidden size for PROVSYN.

Inference temperature is a key parameter in LLM-based inference, controlling the randomness of generated text. Higher temperatures yield more diverse outputs, while lower temperatures produce more deterministic results. We conducted experiments on the Nodlink dataset using temperature values of 0.1, 0.5, 1.0, and 1.5, with all other inference parameters held constant. Evaluation focused on the quality of text generated for process and file node attributes, as well as semantic correctness. As shown in Figure 5, the best performance is achieved when the temperature is set to 1.5. In particular, at lower temperatures, the generated node names tended to be repetitive and lacked diversity. Based on these findings, we selected 1.5 as the final inference temperature for PROVSYN, consistent with the recommendation of the FastLanguageModel library.

Appendix D. Ablation Study

Alternative models to GraphGen. In the heterogeneous graph generation module, we adopt the design of GraphGen, which encodes graphs as DFS code sequences and employs an LSTM for training and inference. To validate the effectiveness of this design, we compare it with alternative models, including the widely used GraphRNN [72]. In our experiments, we evaluate the graphs generated by GraphRNN and GraphGen using five MMD metrics. As shown in Figure 6(a), the graphs generated by GraphGen consistently achieve significantly lower MMD scores across all five metrics, indicating substantially better fidelity compared to those generated by GraphRNN. These results demonstrate the effectiveness of using GraphGen for heterogeneous graph generation in PROVSYN.

Masking strategies. For text attribute generation, we employ two masking strategies to construct the training set, corresponding to two different input sequences during inference. To evaluate the effectiveness of the masking design, we perform ablation experiments by disabling one of the strategies and using only the other. Model performance is then evaluated using text quality metrics and semantic correctness. The experiments are conducted on the Trace dataset, with the number of training samples fixed at 20,000 for both the single-strategy and dual-strategy settings. In the results shown in Figure 6(b), we observe that using only the Full Masking strategy yields better performance than using only the Part Masking strategy. This may be because the Part Masking strategy provides partial node names within

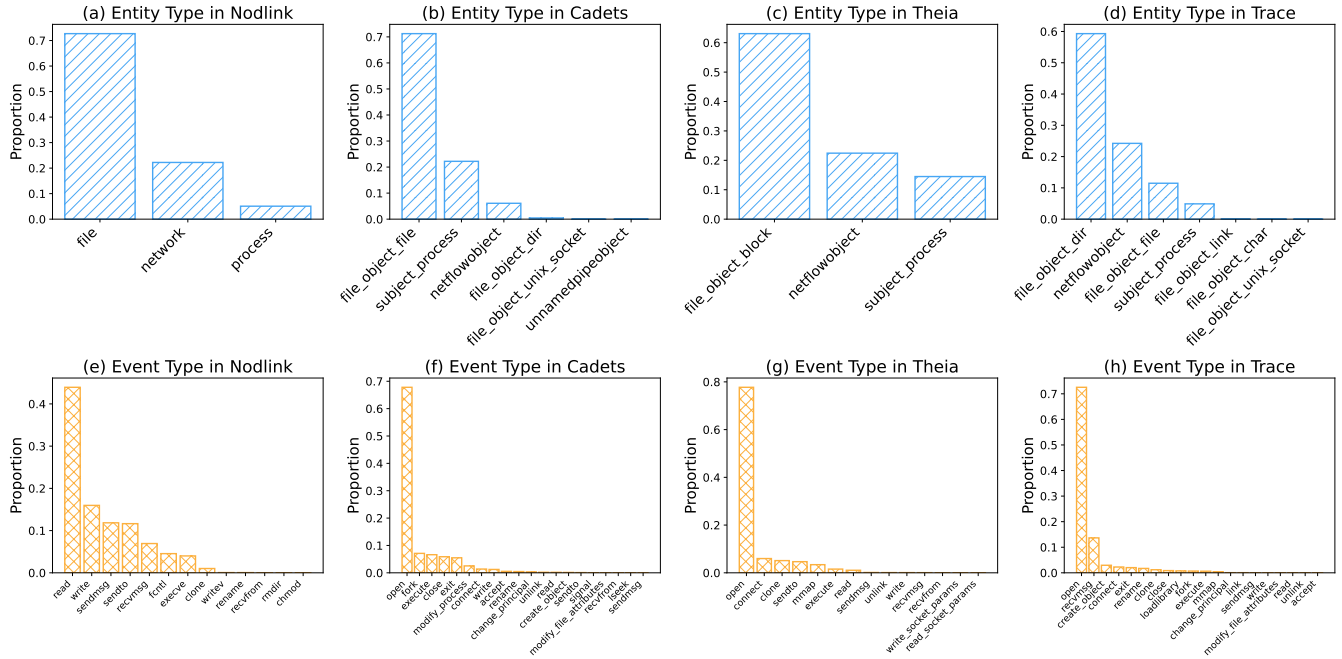


Figure 3: Entity Type and Event Type Distribution in Provenance Dataset including Nodlink, Cadets, Theia and Trace.

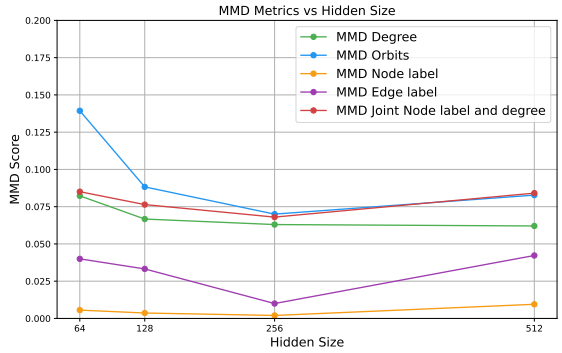
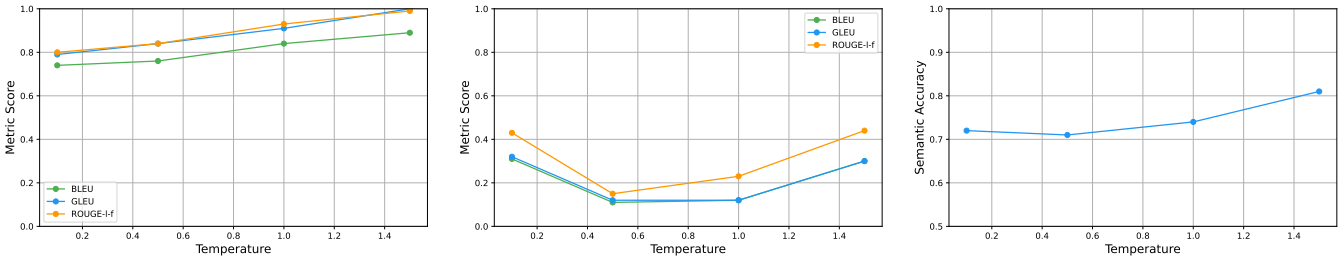


Figure 4: Hidden size setting in Nodlink dataset.

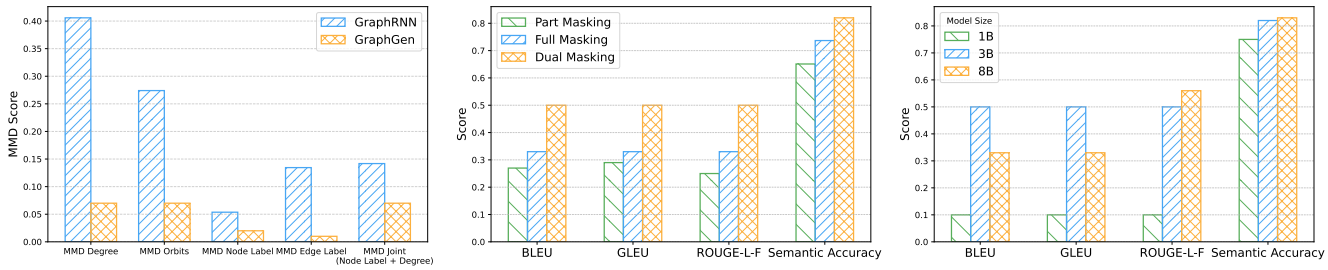
the sequence, offering more cues for the LLM to leverage. As a result, a model trained solely with Part Masking performs poorly when tested on Full Masking sequences, which contain fewer clues and less information. In contrast, using both strategies together achieves better performance than either alone, as it exposes the model to a broader range of sequence types that may appear during inference. These results demonstrate the effectiveness of our proposed approach that combines both masking strategies.

LLM parameter size. In our experiments on text attribute generation, we employ the Llama-3.2-3B-Instruct model as the base model for training. We also compare its performance against models of different sizes, including Llama-3.2-1B-Instruct and Llama-3.1-8B-Instruct. All models are trained on the same training datasets and evaluated using both text quality metrics and the semantic correctness metric. The results, presented in Figure 6(c), demonstrate that the 3B model achieves the highest scores on the BLEU and GLEU metrics. In contrast, the 8B model shows marginally better performance on the ROUGE and semantic correctness metrics. Given that the advantage of the 8B model is not substantial and considering computational resource constraints, we select the 3B model as the base for synthesizing text attributes. In practice, PROVSYN introduces a general training strategy to leverage LLMs for synthesizing text attributes, where the choice of the base model involves balancing the complexity of the dataset, the application requirements and the limitations of computational resources.



(a) Textual Quality for Process vs Temperature (b) Textual Quality for File vs Temperature (c) Semantic Accuracy vs Temperature

Figure 5: Inference temperature setting in Nodlink dataset.



(a) GraphRNN vs GraphGen on MMD Metrics (b) Comparison of Strategies (c) Comparison of Model Sizes

Figure 6: Ablation Study.