Anomaly Detection in Continuous-Time Temporal Provenance Graphs

Jakub Reha[*]
University of Amsterdam
j.reha@uva.nl

Giulio Lovisotto
Huawei Munich Research Center
giulio.lovisotto@huawei.com

Michele Russo
Huawei Munich Research Center
michele.russo@huawei.com

Alessio Gravina[*]
University of Pisa
alessio.gravina@phd.unipi.it

Claas Grohnfeldt
Huawei Munich Research Center
claas.grohnfeldt@huawei.com

Abstract

Recent advances in Graph Neural Networks (GNNs) have matured the field of learning on graphs, making GNNs essential for prediction tasks in complex, interconnected, and evolving systems. In this paper, we focus on self-supervised, inductive learning for continuous-time dynamic graphs. Without compromising generality, we propose an approach to learn representations and mine anomalies in provenance graphs, which are a form of large-scale, heterogeneous, attributed, and continuous-time dynamic graphs used in the cybersecurity domain, syntactically resembling complex temporal knowledge graphs. We modify the Temporal Graph Network (TGN) framework to heterogeneous input data and directed edges, refining it specifically for inductive learning on provenance graphs. We present and release two pioneering large-scale, continuous-time temporal, heterogeneous, attributed benchmark graph datasets. The datasets incorporate expert-labeled anomalies, promoting subsequent research on representation learning and anomaly detection on intricate real-world networks. Comprehensive experimental analyses of modules, datasets, and baselines underscore the effectiveness of TGN-based inductive learning, affirming its practical utility in identifying semantically significant anomalies in real-world systems.

1 Introduction

Recent advancements in research on Graph Neural Networks (GNNs) and increasing efforts by practitioners to solve real-world problems using GNNs across domains have furthered the applicability of GNN-based approaches to solving prediction tasks on interconnected data [1]. Most research on graph learning has been devoted to *static* graphs to date. Conversely, exploring *dynamic* graphs is less prevalent, arguably attributable to the scarcity of public datasets supportive of this line of research [2, 3]. However, most real-world networks are inherently dynamic, continuously evolving over time, necessitating a shift of focus toward dynamic graphs. Early approaches to learning on dynamic graphs used *discrete-time* temporal graph representations, where the underlying system is modeled as a sequence of regularly timestamped graphs (snapshots). Such representation is

[*]Work done while at Huawei Technologies.

Temporal Graph Learning Workshop @ NeurIPS 2023, New Orleans.

sub-optimal, as it incurs in the loss of edge-level granularity of temporal information, naturally present in real-world systems [4]. A more general approach to modeling dynamic networks is to use *continuous-time* temporal graph representations, commonly derived from sequences or streams of irregularly timestamped events which correspond to operations on the graph [5].

Despite the natural heterogeneity often encountered in real-world dynamic systems, a large corpus of recent literature on temporal graph learning is still limited to *homogeneous* graph representations not explicitly accounting for types of nodes or edges [6]. However, due to the growing interest and recent advances in the area of knowledge graphs, which are generally heterogeneous, progress has been made also in the domain of representation learning for temporal knowledge graphs [7, 8]. Yet, to the best of our knowledge, there exists no public representative dataset of a real-world *continuous-time* temporal and heterogeneous network to date, which contains edge-level temporal information rather than regular timestamps on snapshots of graphs [9], and which would support explorative and comparative research in this area.

In real-world systems represented as heterogeneous, continuous-time dynamic graphs, data is often collected as a stream of events with previously unseen entities and relationships occurring frequently. In this case, the learning approach must be *inductive* rather than transductive [4, 10]. In this realm, the Temporal Graph Network (TGN) has been one of the foundational frameworks allowing for inductive self-supervised learning on continuous-time dynamic graphs [11]. Nevertheless, despite the generality of its formulation, TGN has not yet been tailored to accommodate the inherent heterogeneity of real-world graphs to the best of our knowledge.

Anomaly detection, particularly in dynamic graphs, is a critical component of network analysis, with applications in fault detection, abuse detection, fraud detection, and intrusion detection [12, 13]. However, leveraging inductive graph learning-based approaches for graph anomaly detection remains unexplored, arguably due to the lack of representative datasets with labeled anomalies.

In this paper, we address the research gaps above by presenting an approach to self-supervised, inductive learning for heterogeneous, continuous-time dynamic graphs and its application to graph anomaly detection. We introduce and showcase the effectiveness of our method by learning representations and mining anomalies in *provenance graphs*, which are a form of large-scale, heterogeneous, attributed, and continuous-time dynamic graphs capturing various types of causal relationships between typed entities [14]. Provenance graphs have become particularly prevalent for anomaly-based threat detection and root cause analysis in the domain of cybersecurity recently [13].

In this paper, we make the following contributions:

- We pioneer inductive graph representation learning for provenance graphs, modeled as continuous-time temporal, directed, attributed, and heterogeneous graphs. Here, we leverage the self-supervised learning task of link prediction to learn node representations and detect cyber threats as edge-level graph anomalies.
- We introduce two heterogeneous, attributed, continuous-time temporal graph benchmark datasets with expert-labeled anomalies corresponding to cyber threats. We make these available online.
- We adapt the TGN architecture to the heterogeneous setting and explicitly extend it to capture edge directionality information. We conduct extensive experimental analyses of the method's components and our extensions, including memory usage, edge directionality, and heterogeneity.

We release datasets and source code implementing the considered methods online[2].

2 Preliminaries

2.1 Static Heterogeneous Graphs

Static heterogeneous graphs, also known as Heterogeneous Information Networks (HINs) [16], can be described as a set of vertices and edges, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, which are associated with a vertex type mapping function $\phi : \mathcal{V} \to \mathcal{T}$ and an edge type mapping function $\psi : \mathcal{E} \to \mathcal{R}$, where $\mathcal{V}$ is the set of n nodes, $\mathcal{T}$ is the set of node types and $\mathcal{R}$ the set of relationship types. Nodes and edges are also equipped with features, which are denoted $\mathbf{x}_i \in \mathbb{R}^{d_v}$ and $\mathbf{e}_{ij} \in \mathbb{R}^{d_e}$ respectively, where $i, j \in \mathcal{V}$. For undirected graphs, where an edge can be represented as a set $\{j, i\}$ we define the

[2]https://github.com/JakubReha/ProvCTDG/

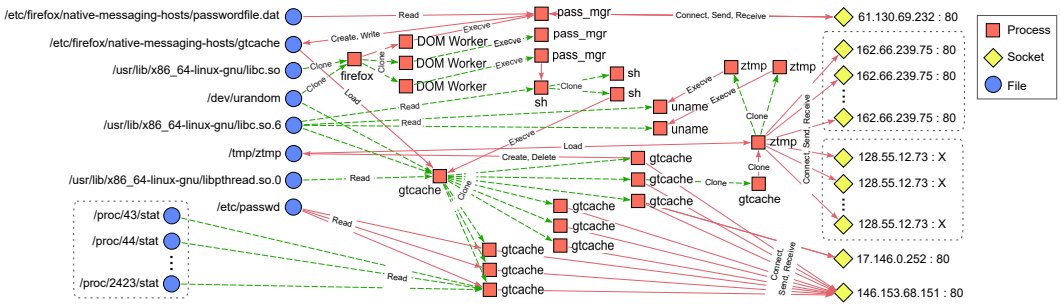


Figure 1: A snapshot view of a provenance (sub)graph, created from a stream of timestamped edges derived from endpoint logs collected on a single computer over 10 minutes. Malicious (anomalous) edges associated with a Browser Extension attack [15] are colored red, while benign (normal) edges are shown as dashed green lines. Node attributes and edge timestamps are omitted, and parallel edges are summarized for clarity.

neighborhood of i as $\mathcal{N}_i = \{j : \{i, j\} \in \mathcal{E}\}$ [17]. In the case of a directed graph, the neighborhood of i is $\mathcal{N}_i = \{j : (j, i) \in \mathcal{E}\}$. Unless stated otherwise, we consider the undirected neighborhood. In this paper, the word *vertex* and *node* are used interchangeably, as well as the words *edge* and *link*.

2.2 Continuous-Time Dynamic Heterogeneous Graphs

We refer to our settings and data types as Continuous-Time Dynamic Graphs (CTDGs) [2]. Instead of operating with periodic graph snapshots (i.e., Discrete-TDGs), we work with a stream of timestamped events. Given such conditions, we extend the definition of CTDGs to heterogeneous graphs. We define a Continuous-Time Dynamic Heterogeneous Graph (CTDHG) $\mathcal{G}$ as a sequence of events

$$\mathcal{G} = \{(i, r, j, t, \mathbf{e}) : i, j \in \mathcal{V}(t), r \in \mathcal{R}\}, \tag{1}$$

where each event $(i, r, j, t, \mathbf{e})$ identifies the addition of an edge of type r between nodes i and j at time t. Each event also comprises a feature vector, $\mathbf{e}$, which defines the state associated with the current event. Notably, the set of nodes is a function of time, thus, $\mathcal{V}(t)$ retrieves the set of nodes observed up to and including time t. Despite the dynamic setting, $\mathcal{R}$ remains the set of relationship types, and nodes remain assigned features, $\mathbf{x}_i$, and types, $\tau_i \in \mathcal{T}$. Note that in our setting the mappings between nodes and their types ($i \to \tau_i$) as well as nodes and their features ($i \to \mathbf{x}_i$) are unique, i.e., nodes always appear with the same type and features. Also note that the graph defined in Eq. 1 is directed, and that since edges between pairs of nodes can re-occur this describes a multi-graph. We define the (undirected) temporal neighborhoods of node i at time T analogously to the static case $\mathcal{N}_i(T) = \{j : \{j, r, i, t\} \in \mathcal{G}, t < T\}$. Note that CTDHG can be converted into a HIN by discarding temporal information and subsequently removing duplicate edges with the same type and direction.

2.3 Temporal Link Prediction for Anomaly Detection

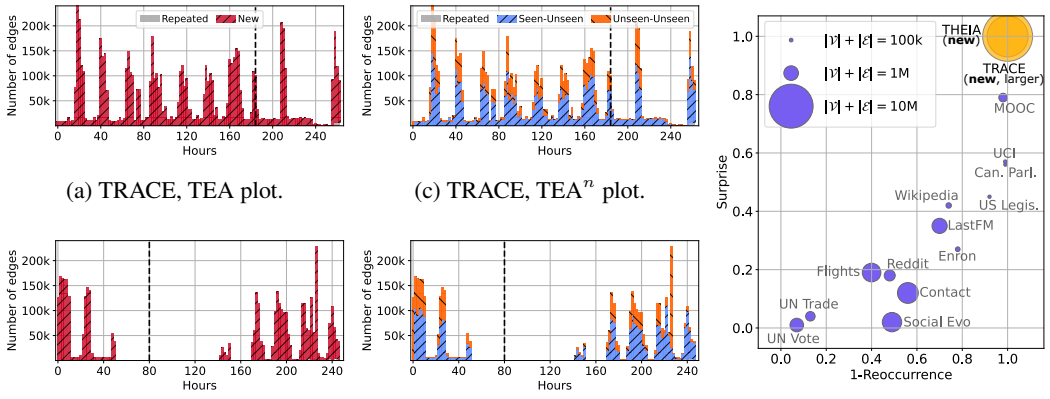
In graph learning, temporal link prediction is a common downstream task, with the goal to predict existence of edges at specific timestamps. Formally, the temporal link prediction task can be seen as learning a function that maps node pairs and time (i.e., an edge, ϵ) into a probability of existence: $f(\epsilon) : \mathcal{V} \times \mathcal{V} \times \mathbb{R}^+ \to [0, 1]$. Note that in CTDHG, edge types $\mathcal{R}$ are also part of the input domain. In practice, given that datasets only contain the so called *positive* edges (i.e. edges that have actually occurred), link prediction is formulated as a self-supervised task with the use of *negative sampling* [18]. Negative sampling samples one negative edge $\bar{\epsilon} = (i, k, t)$ per positive edge $\epsilon = (i, j, t)$, where the link between i and k was not observed at time t. In reference to Eq. 1, where the positive edges constitute the temporal graph $\mathcal{G}$, the negatively sampled edges constitute a negative (unobserved) temporal graph:

$$\bar{\mathcal{G}} = \{(i, k, t) : (i, k, t) \notin \mathcal{G}\}. \tag{2}$$

$\mathcal{G}$ and $\bar{\mathcal{G}}$ are used to optimize parametric models f_θ by minimizing a binary classification loss $\mathcal{L}$:

$$\text{argmin}_\theta \; \mathcal{L}\left(f_\theta(\mathcal{G}), 1\right) + \mathcal{L}\left(f_\theta(\bar{\mathcal{G}}), 0\right). \tag{3}$$

Models trained with such self-supervised loss should associate lower probability of occurrence to edges based on their abnormality; hence one application of such models is the detection of anomalous



(a) TRACE, TEA plot.

(c) TRACE, TEA^n plot.

(b) THEIA, TEA plot.

(d) THEIA, TEA^n plot.

(e) Temporal datasets comparison.

Figure 2: THEIA and TRACE TEA plots (Fig. 2b and 2a), TEA^n plot (Fig. 2d and 2c) and comparison with other temporal datasets across size, *reoccurrence* and *surprise* [19] (Fig. 2e). THEIA and TRACE present challenging properties of high novelty and low reoccurrence. For histograms, we discretize the continuous time in 2-hours bins and show the train/test split with a vertical line.

edges. The assumption behind this use-case is that the loss of Eq. 3 is a good proxy for the objective of finding anomalous edges. Nevertheless, analyzing the correspondence between raw performance and actual detection rate of anomalies is often impossible due to the lack of labeled anomalous edges. To enable such investigation, we introduce two new datasets with expert-labeled anomalies (see Sec. 3) and we evaluate using two separate tasks and metrics (see Sec. 5.2).

3 Provenance Graphs

Given the scarcity of available public temporal graphs [19], we identify a new suitable dynamic data source: system-level *data provenance* graphs. Provenance graphs describe the history of an operating system's execution [20]: these can be collected by software (i.e., audit loggers) which attaches to the OS's kernel to record low-level *system calls*. In provenance graphs, processes are the main actors as they interact with other entities. For example, when a process */usr/bin/vim* opens a file */etc/hosts*, this results in a directed, labeled ($read$), attributed and timestamped edge */etc/hosts* $\rightarrow$ */usr/bin/vim*. Here, *read* identifies the edge type, and both nodes are also associated with types (most commonly *File*, *Process* and *Socket*) - matching the description of CTDHG (Eq. 1). The edge direction indicates the direction of the information flow. See an example provenance graph in Fig. 1.

3.1 Anomaly Detection in Provenance Graphs

Provenance graphs are being widely used in threat detection research [21, 22, 23, 24, 25, 26, 27, 28] with the goal of identifying malicious activities on hosts which may originate from malware. The rationale is that a model can learn the benign behavior of such graphs as they evolve over time, with malicious behavior being associated with anomalies in this evolution. Streamspot [24], one of the earliest works in this direction, monitors edges in a streaming fashion, constructs (sub-)graph sketches periodically and detects anomalous graphs by comparing new graphs to old known clusters.

Several works followed, but many present limitations. Some methods discard temporal information [22, 28, 29, 21], which leads to sub-optimal modelling. Many methods cannot operate in the *inductive* setting, only in the transductive one [27, 28, 29, 21]: this leads to delays in the early detection of anomalies which may be crucial in order to stop a system from being compromised. Lastly, considering the large amounts of provenance data, most methods trade off computational efficiency the capability of pinpointing indicators of attacks at the entity- or edge-level. To do so, many approaches choose to detect anomalies at coarse granularity levels such as graph snapshots constructed over predefined time periods or from a fixed number of logs [24, 26, 30, 23]. Such methods have limited usefulness for security analysts due to the necessity of identifying event-level threat indicators manually. We explain how our method addresses these challenges in Sec. 4.

3.2 DARPA Datasets

We introduce two datasets constructed from the audit logs collection made by DARPA Engagement 3 [15], namely THEIA and TRACE. Each dataset contains audit data from a host for a period of two weeks, and contains both benign and malicious activities; the latter carried out by a red team. We detail the dataset pre-processing, feature extraction, data splits, and ground truth labeling in App. A. Here, we analyze the datasets temporally evolving patterns, using the insights provided in [19].

Table 1: Statistics of the DARPA E3 datasets. Cybersecurity experts labelled individual edges as malicious or benign using information on the execution of the malicious activities. $|\mathcal{T}|$ is the number of node types, $|\mathcal{R}|$ the number of relationship types. We extract 18 node features for Sockets, 23 for Files and 21 for Processes. THEIA and TRACE, respectively, are collected over a timespan of 247h and 264h, leading to 1.8M and 2.8M unique timestamps. See Appendix A for complete details.

Dataset	Nodes	Edges	Unq. Edges	Malic. Edges	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{R}\rvert$
THEIA	1,043k	8,416k	3,982k	211	3	8
TRACE	3,926k	10,102k	6,663k	262	3	10

TEA visualizations. We use the Temporal Edge Appearance plot, TEA [19], to illustrate the portion of repeated edges versus newly observed edges for each timestamp; TEA plots for are shown in Fig. 2b and 2a. The plots highlight how the datasets have an extremely high ratio of novel edges in each timestamp versus the number of repeated edges. While the TEA plots show that there is only little repetition of edges, it is more descriptive to consider whether the novel edge occurs between two previously unseen nodes (*Unseen-Unseen*) or one seen and one unseen node (*Seen-Unseen*). These two distributions are more indicative when considering whether memorizing past occurrences can capture information useful for solving the link prediction task. In light of this, we introduce a new plot, named TEA^n , which extends the original link-centric TEA plot to include a node-centric perspective with the Seen/Unseen differences; we report the TEA^n plots for TRACE and THEIA in Fig. 2d and 2c. As shown in the figures, while the overwhelming majority of edges are novel, the larger part of the novel edges belong to the Seen-Unseen distribution in which one of the two nodeshas been observed before. This indicates that historical information might still be beneficial for learning on the link prediction task.

Comparison to existing temporal graph datasets. To highlight the value of the datasets, we compare them with the set of temporal graph datasets presented in [19]. We use two metrics introduced in the same paper: (i) the *reoccurrence* index, i.e., the ratio between the number of edges that occur both in train- and testset versus the number of edges in the trainset, and (ii) the *surprise* index i.e., the ratio between the number of edges that occur in the test- and not in trainset versus the number of edges in the testset. Intuitively, the process of memorization, or remembering past edges, is most effective when reoccurrence is high and surprise is low [19]. On the other hand, memorization is less helpful and therefore the task becomes harder when reoccurrence is low and surprise is high. We plot these two indexes together with the sizes of the datasets measured as $|\mathcal{V}|+|\mathcal{E}|$ in Fig. 2e, showing how the DARPA datasets present a more challenging real-world setting for temporal graph learning.

4 Method

The method used addresses temporal and heterogeneous data types, focusing on the inductive setting and obtaining fine-grained predictions on single events. We adopt and evaluate the general TGN [11] framework for CTDGs (more details are provided in App. B) and extend it to the heterogeneous (CTDHG) setting. We train TGN for the link prediction task, producing a fixed-size representation for every node it encounters at each time t. The output of the link prediction is then used for the downstream task of anomaly detection.

4.1 Encoding Edge Directionality

While TGN encodes the directionality of edges when computing messages to update node memories (see Eq. 4), the model does not account for directionality during the creation of node neighborhoods, i.e., the neighborhood is undirected. Given repeated evidence that directionality is beneficial for learning [31, 32], we encode the direction of edges with two encoding variations:

- **OHD-TGN**: we learn one embedding per edge direction and we sum it to edge features $\hat{\mathbf{e}}_{ij}$.
- **DIR-TGN**: in the graph convolution we encode incoming or outgoing edges with two separate embedding layers $\mathbf{W}_e^{(\text{in})}$ and $\mathbf{W}_e^{(\text{out})}$ rather than the single $\mathbf{W}_e$ in Eq. 8 and Eq. 9.

4.2 Encoding Heterogeneous Information

We extend TGN to model the heterogeneous cases presented by our datasets. We do so by learning an embedding per node and edge type, and concatenating it to the existing edge and node features (i.e., $\mathbf{e}_{ij}$ and x_i); we refer to this model as **TGN**. As this is a straightforward extension which might not be sufficient to capture complex relationships across types, we introduce two approaches to allow TGN to better model heterogeneous data:

- **Hetero-TGN**: heterogeneous TGN. Inspired by [33], we use separate transformer convolutions per type triplet $(\tau_i, r_k, \tau_j) \in \mathcal{T} \times \mathcal{R} \times \mathcal{T}$, each triplet identifies the convolution used. In practice, we learn one set of the MLPs in Eq. 8 and Eq. 9 per triplet, with the triplet uniquely identifying the set of MLPs used for a sample. Note that, as in [33], we consider $\mathcal{R}$ to contain both relations in the forward and in the backward direction (e.g. *opens* and *is opened*). Per-triplet node embeddings are summed to produce final node embeddings.
- **HGT-TGN**: heterogeneous graph transformer (HGT) TGN. To enable message passing and attention across triplet types we replace the graph convolution specified in Eq. 8 with the heterogenous convolution introduced in [34]. To account for edge features, we adapt HGT by concatenating edge representations to the messages computed on node pairs, such edge representation is shared across the attention heads. We describe the obtained message passing layer in more detail in App. B.2.

We evaluate TGN and its extensions in Sec. 5.2.

5 Experiments

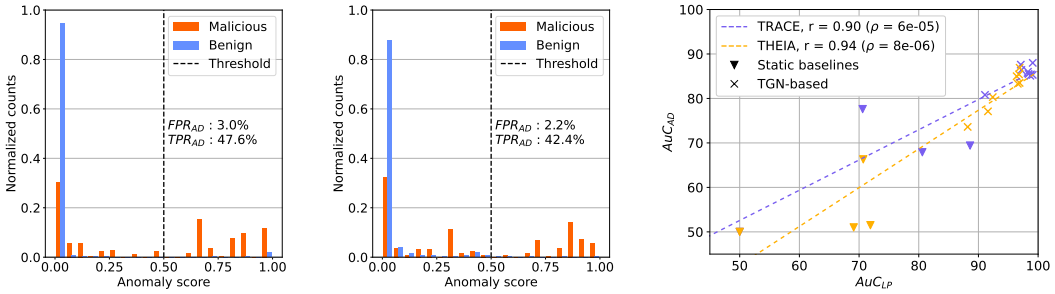
5.1 Experimental Setting

We introduce static (i.e., non-temporal) baselines to compare to our methods, as the static setting resembles the one in other provenance graph anomaly detection works [21, 28, 22]. To allow for the comparison, we create a static graph from the CTDHGs by converting them into HINs i.e., dropping temporal information and parallel edges. We choose three baselines: a multi-layer perceptron (MLP), the attentive graph convolution proposed in [35] (i.e., GAT), and the heterogeneous graph convolution proposed in [33] (i.e., RGCN). We note that both MLP and GAT include a latent representation of the encoded node and edge types (learned with an embedding layer) as part of the input node/edge features, RGCN deals with heterogeneous data by default. Moreover, for the graph convolutions, we aggregate the information as coming from an undirected neighborhood, as for **Hetero-TGN**.

To train models, we add an MLP head on top of node embeddings to predict the probability of edge existance. GRU is used as the mem function in Eq. 6. Where applicable, we conduct ablations where we consider models that operate without node and edge features, identified by $\dagger$, without memory, identified by $\S$, or without both ($\dagger\S$). We conduct the experiments on a CPU cluster equipped with 240 Intel(R) Xeon(R) CPU E5-2658A v3@2.20GHz. Results are averaged over five separate runs.

We employ a type-aware negative sampling strategy, where for each positive sample we only replace the destination node (not the edge type) with a random sample of the same node type, with the ratio 1:1. We also do the following: (1) during training we sample negative destinations only from nodes that appear in the trainset, (2) during validation we sample them from nodes that appear in trainset or validation set and (3) during testing we sample them from the entire node set.

Since our goal is to evaluate methods both on on temporal link prediction task and on the detection of anomalous edges, we introduce two metrics. For clarity, following Sec. 2.3, we identify three sets of edges leading to three temporal graphs: the positive edges $\mathcal{G}$ (i.e., Eq. 1), the negative sampled edges $\bar{\mathcal{G}}$ (see Eq. 2), and the subset of observed anomalous edges which were labelled by experts in the datasets $g_{\mathcal{A}} \subseteq \mathcal{G}$. For temporal link prediction, we compute area under the ROC curve (AuC_{LP}) on all positive non-anomalous edges and negative edges $\mathcal{G} \setminus g_{\mathcal{A}} \cup \bar{\mathcal{G}}$. For the anomaly detection task, we only use the events in $\mathcal{G}$ which contain both benign and malicious edges, and compute AuC_{AD}. To contextualize the detection results, we use true positive rates (TPR_{AD}) and false positive rates (FPR_{AD}) obtained at a specific decision threshold τ (see Tab. 4 in App. B.3 for reference).



(a) TRACE, Anomaly score histogram. Counts are normalized per label.

(b) THEIA, Anomaly score histogram. Counts are normalized per label.

(c) Correlation between AuC_{LP} and AuC_{AD}. Here r refers to Pearson's r and ρ to its p-value.

Figure 3: Results analysis. The anomaly score is computed as 1-$f_\theta(\epsilon)$, $f_\theta(\epsilon)$ being the model output.

5.2 Results

Link Prediction Tab. 2 presents the results on the link prediction task (AuC_{LP}). Temporal models outperform the static baselines on both datasets. We notice the benefit brought by node features: models using features (MLP and TGN) outperforms their feature-less (†) counterparts. For the memory part of TGN, we see large improvements when memory is present in the feature-less models (TGN†§ vs TGN†). Nevertheless, such gain is almost nullified in the models that use features (TGN§, TGN). Note that the memory-less TGN§ is equivalent to TGAT [4]. We hypothesize that in provenance graphs, while memory might still be important in rare causal sequences, informative features compensate for lack of memory and enable models to learn normal behavior.

Table 2: AuC_{LP} and AuC_{AD} for the two datasets. Blue marks the best performing model, Orange the second best. Models marked by † do not use node features and models marked by § do not use memory.

Benchmark	TRACE		THEIA	
Metric	AuC_{LP}	AuC_{AD}	AuC_{LP}	AuC_{AD}
MLP†	$50.0_{\pm 0.0}$	$50.0_{\pm 0.0}$	$50.0_{\pm 0.0}$	$50.0_{\pm 0.0}$
MLP	$70.6_{\pm 0.4}$	$77.6_{\pm 1.3}$	$70.7_{\pm 0.4}$	$66.3_{\pm 2.6}$
GAT	$80.6_{\pm 7.7}$	$67.9_{\pm 14.5}$	$69.1_{\pm 4.3}$	$51.0_{\pm 12.8}$
RGCN	$88.6_{\pm 0.8}$	$69.4_{\pm 6.6}$	$71.9_{\pm 1.8}$	$51.5_{\pm 16.5}$
TGN†§ **(TGAT**†**)**	$91.1_{\pm 1.1}$	$80.8_{\pm 0.7}$	$88.2_{\pm 0.6}$	$73.6_{\pm 1.2}$
TGN†	$97.2_{\pm 0.3}$	$86.4_{\pm 1.9}$	$92.4_{\pm 1.8}$	$80.3_{\pm 2.8}$
TGN§ **(TGAT)**	$99.1_{\pm 0.1}$	$88.0_{\pm 1.2}$	$96.4_{\pm 0.2}$	$85.0_{\pm 2.2}$
TGN	$99.1_{\pm 0.1}$	$85.3_{\pm 1.5}$	$96.9_{\pm 0.1}$	$83.5_{\pm 1.0}$
OHD-TGN	$98.4_{\pm 0.2}$	$85.5_{\pm 1.4}$	$96.9_{\pm 0.2}$	$86.9_{\pm 2.5}$
DIR-TGN	$98.7_{\pm 0.3}$	$85.1_{\pm 1.0}$	$96.6_{\pm 0.2}$	$83.3_{\pm 1.9}$
Hetero-TGN	$98.3_{\pm 0.2}$	$86.0_{\pm 0.5}$	$96.8_{\pm 0.3}$	$85.4_{\pm 2.8}$
HGT-TGN	$97.1_{\pm 0.3}$	$87.6_{\pm 0.8}$	$91.6_{\pm 0.5}$	$77.1_{\pm 4.6}$

In general, the results show that providing further directionality information to TGN does not bring any benefit. This suggests that encoding edge direction in the messages (Eq. 4) is sufficient for learning. Looking at the static baselines results, we find that RGCN with a single layer outperforms GAT with two layers, suggesting that explicitly modelling the heterogeneity in the model architecture has a beneficial effect. However, this is not the case for TGN, where simple learned encoding of heterogeneity (TGN, TGN§) brings rather consistent performance improvements compared to the heterogeneous models Hetero-TGN and HGT-TGN. This implies that temporal information compensates for the heterogeneous information. Results on heterogeneous models suggest that the additional complexity and computational cost incurred when modeling different types is not justified by performance gains, see Tab. 6 in App. B.3.

Anomaly Detection Tab. 2 also presents the anomaly detection results (AuC_{AD}). Temporal methods outperform static baselines, and extensions of TGN bring negligible improvements. Interestingly, TGN without memory (TGN§) outperforms TGN. We do not find clear winners across the results in Tab. 2, with OHD-TGN outperforming baseline TGN on THEIA, and HGT-TGN also outperforming TGN on TRACE, but improvements being inconsistent across the two datasets.

To investigate further the connection between the link prediction and anomaly detection task (see Sec. 2.3), we look at the correlation between the link prediction metric (AuC_{LP}) and the anomaly detection metric (AuC_{AD}), see Fig. 3c. The figure shows strong correlation between the two tasks,

confirming the intuition that malicious edges indeed correspond to anomalies in our data and link prediction with negative sampling is an effective optimization strategy for the detection such anomalies. Although, more negative sampling strategies could be investigated to reduce the offset between (AuC_{LP}) and (AuC_{AD}).

In Fig. 3a and 3b we plot the distribution of anomaly scores for TGN§. We observe that the model generally assigns high anomaly scores to malicious edges and low score to benign ones. Setting the decision threshold at $\tau = 0.5$ the model can detect a large portion of malicious events (TPR_{AD}) while keeping the false positives relatively low (FPR_{AD}). Note that in datasets with millions of events, acceptable detection tradeoffs might require lower false positive rates to be used successfully. Therefore, we further analyze false positives below.

5.3 False Positives Case Study

We study the false positives (FP) of the best performing model TGN§ on TRACE. The comparison between FP and true negatives (TN) can reveal what the model bases its decisions on. In addition, a comparison between FP at test time and edges occurring in the train set might reveal whether a portion of FP are benign anomalies, rather than malicious actions.

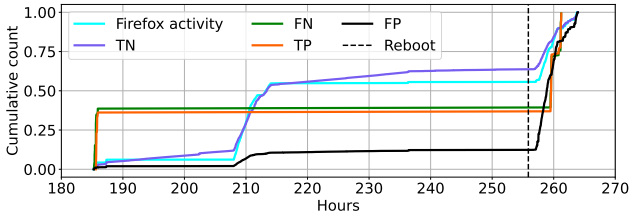


Figure 4: Lines represent normalized cumulative detection counts for TGN§ on the TRACE test set.

We find that the vast majority (86.6%) of edges detected as FP by the model involves nodes of type "Process" with the name `firefox`. This is in contrast to only 53.1% of edges resulting in TN that involve `firefox`, suggesting that these edges are harder to distinguish correctly, even though they are well represented in the train set (50.7% of the totality of train edges). Furthermore, edges that have a `firefox` process as the source are sizeably more common among FPs (75.2% of the total FP) than they are among TN (10.2% of the total TN) or even train set (14.3% of total train edges). We hypothesize that, due to the current negative sampling strategy and the low number of neighbors per-node, a large number of negative edges with `firefox` as source sampled during training are very similar to the positive edges - this leads to difficulty in distinguishing anomalous edges with these properties. We hypothesize that this issue problem can be reduced with specialized negative sampling strategies, which we leave for future work.

Fig. 4 shows the cumulative distribution of TN, FN, TP, FP and `firefox` activity over time. The figure highlights the correlation between `firefox` activity and FP. A large number of FP (more than 80% of the total) happens around the 255$^{\text{th}}$ hour. At that time, the machine logs reveal that there was a reboot, and shortly after, the start of the malicious attack - which involved the installation of a firefox backdoor malware. We hypothesize that the reboot and the malicious activity create noise in node embeddings produced by TGN, leading to widespread FPs even outside of the malicious activity.

6 Conclusion

We investigated the application of temporal graph networks to provenance graphs, focusing on anomaly detection in the inductive setting. To do so, we introduced and released two new, continuous-time, heterogeneous, and attributed provenance graph datasets with expert-labeled anomalies that correspond to malicious activities. We extended the TGN framework to account for edge directionality and heterogeneous data. We conducted extensive experimentation to identify the benefits and drawbacks of TGN modules and the proposed extensions. We demonstrated and analyzed the connection between the self-supervised optimization in link prediction and the detection of semantically meaningful anomalies. Our work paves the way for further research on representation learning and anomaly detection in intricate real-world networks and systems.

References

[1] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *Journal of Machine Learning Research*, vol. 24, no. 43, pp. 1–48, 2023.

[2] S. M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart, "Representation learning for dynamic graphs: A survey," *Journal of Machine Learning Research*, vol. 21, no. 70, pp. 1–73, 2020.

[3] C. D. T. Barros, M. R. F. Mendonça, A. B. Vieira, and A. Ziviani, "A survey on embedding dynamic graphs," *ACM Computing Surveys*, vol. 55, no. 1, 2021.

[4] D. Xu, C. Ruan, E. Körpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," in *ICLR*, 2020.

[5] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *WWW*, 2018.

[6] J. Zhao, X. Wang, C. Shi, B. Hu, G. Song, and Y. Ye, "Heterogeneous graph structure learning for graph neural networks," in *AAAI*, vol. 35, pp. 4697–4705, 2021.

[7] L. Zhang and D. Zhou, "Temporal knowledge graph completion with approximated Gaussian process embedding," in *International Conference on Computational Linguistics*, 2022.

[8] L. Bai, W. Yu, D. Chai, W. Zhao, and M. Chen, "Temporal knowledge graphs reasoning with iterative guidance by temporal logical rules," *Information Sciences*, vol. 621, 2023.

[9] J. Gastinger, T. Sztyler, L. Sharma, and A. Schuelke, "On the evaluation of methods for temporal knowledge graph forecasting," in *NeurIPS Temporal Graph Learning Workshop*, 2022.

[10] Y. Wang, Y. Chang, Y. Liu, J. Leskovec, and P. Li, "Inductive representation learning in temporal networks via causal anonymous walks," in *ICLR*, 2021.

[11] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," in *ICML Workshop on Graph Representation Learning*, 2020.

[12] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[13] M. Zipperle, F. Gottwalt, E. Chang, and T. Dillon, "Provenance-based Intrusion Detection Systems: A Survey," *ACM Computing Surveys*, 2022.

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

[15] DARPA, "Transparent computing engagement 3 data release." `https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md`, 2020.

[16] Y. Sun and J. Han, "Mining heterogeneous information networks: A structural analysis approach," *SIGKDD Explor. Newsl.*, vol. 14, 2013.

[17] D. Bacciu, F. Errica, A. Micheli, and M. Podda, "A gentle introduction to deep learning for graphs," *Neural Networks*, vol. 129, 9 2020.

[18] Z. Yang, M. Ding, C. Zhou, H. Yang, J. Zhou, and J. Tang, "Understanding negative sampling in graph representation learning," in *ACM SIGKDD*, 2020.

[19] F. Poursafaei, S. Huang, K. Pelrine, and R. Rabbany, "Towards better evaluation for dynamic link prediction," in *NeurIPS*, vol. 35, 2022.

[20] A. Gehani and D. Tariq, "Spade: Support for provenance auditing in distributed environments," in *Middleware 2012: ACM/IFIP/USENIX International Middleware Conference*, Springer, 2012.

[21] 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 *IEEE Symposium on Security and Privacy*, 2022.

[22] 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*, vol. 17, 2022.

[23] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *Network and Distributed System Security Symposium*, 2020.

[24] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in *ACM SIGKDD*, 2016.

[25] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows," in *IEEE Symposium on Security and Privacy*, 2019.

[26] D. Eswaran, C. Faloutsos, S. Guha, and N. Mishra, "Spotlight: Detecting anomalies in streaming graphs," in *ACM SIGKDD*, 2018.

[27] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. J. Rhee, Z. Chen, W. Cheng, C. A. Gunter, and H. Chen, "You are what you do: Hunting stealthy malware via data provenance analysis," in *Network and Distributed System Security Symposium*, 2020.

[28] J. Zeng, Z. L. Chua, Y. Chen, K. Ji, Z. Liang, and J. Mao, "Watson: Abstracting behaviors from audit logs via aggregation of contextual semantics.," in *Network and Distributed System Security Symposium*, 2021.

[29] Z. Xu, P. Fang, C. Liu, X. Xiao, Y. Wen, and D. Meng, "Depcomm: Graph summarization on system audit logs for attack investigation," in *IEEE Symposium on Security and Privacy*, 2022.

[30] 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 *ACM SIGSAC conference on computer and communications security*, 2019.

[31] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, 2009.

[32] J. Li, Z. Han, H. Cheng, J. Su, P. Wang, J. Zhang, and L. Pan, "Predicting path failure in time-evolving graphs," in *ACM SIGKDD*, 2019.

[33] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *Extended Semantic Web Conference*, Springer, 2018.

[34] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *WWW*, 2020.

[35] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," in *ICLR*, 2018.

[36] DARPA, "Transparent computing program." `https://darpa.mil/program/transparent-computing`, 2014.

[37] J. Khoury, T. Upthegrove, A. Caro, B. Benyo, and D. Kong, "An event-based data model for granular information flow tracking," in *International Workshop on Theory and Practice of Provenance*, USENIX Association, 2020.

[38] M. 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 *IEEE Symposium on Security and Privacy*, 2023.

[39] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[40] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," in *IJCAI*, pp. 1548–1554, 8 2021. Main Track.

[41] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR*, 2019.

[42] A. Longa, V. Lachi, G. Santin, M. Bianchini, B. Lepri, P. Lio, F. Scarselli, and A. Passerini, "Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities," *TMLR 23*, 2023.

[43] R. Trivedi, H. Dai, Y. Wang, and L. Song, "Know-evolve: Deep temporal reasoning for dynamic knowledge graphs," in *ICML*, PMLR, 2017.

[44] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *ACM SIGKDD*, 2019.

[45] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Dyrep: Learning representations over dynamic graphs," in *ICLR*, 2019.

[46] Y. Ma, Z. Guo, Z. Ren, J. Tang, and D. Yin, "Streaming graph neural networks," in *ACM SIGIR conference on research and development in information retrieval*, 2020.

[47] X. Wang, D. Lyu, M. Li, Y. Xia, Q. Yang, X. Wang, X. Wang, P. Cui, Y. Yang, B. Sun, *et al.*, "APAN: Asynchronous propagation attention network for real-time temporal graph embedding," in *International conference on management of data*, 2021.

[48] S. Tian, T. Xiong, and L. Shi, "Streaming dynamic graph neural networks for continuous-time temporal graph modeling," in *IEEE ICDM*, 2021.

[49] A. H. Souza, D. Mesquita, S. Kaski, and V. Garg, "Provably expressive temporal graph networks," in *NeurIPS*, 2022.

[50] J. Gao and B. Ribeiro, "On the equivalence between temporal and static equivariant graph representations," in *ICML*, PMLR, 2022.

[51] A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, and A. Zimmermann, "Knowledge graphs," *ACM Computing Surveys*, 2022.

[52] D. Busbridge, D. Sherburn, P. Cavallo, and N. Y. Hammerla, "Relational graph attention networks," in *ICLR*, 2019.

[53] Y. Dong, Z. Hu, K. Wang, Y. Sun, and J. Tang, "Heterogeneous network representation learning," in *IJCAI*, vol. 20, 2020.

[54] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *ACM SIGKDD*, 2019.

[55] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *WWW*, 2019.

[56] A. Mitra, P. Vijayan, S. R. Singh, D. Goswami, S. Parthasarathy, and B. Ravindran, "Revisiting link prediction on heterogeneous graphs with a multi-view perspective," in *IEEE ICDM*, 2022.

[57] S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar, "Composition-based multi-relational graph convolutional networks," in *ICLR*, 2020.

[58] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," in *NeurIPS*, vol. 32, 2019.

[59] H. Sun, J. Zhong, Y. Ma, Z. Han, and K. He, "Timetraveler: Reinforcement learning for temporal knowledge graph forecasting," *arXiv preprint arXiv:2109.04101*, 2021.

[60] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, "Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, 2018.

[61] S. Bhatia, M. Wadhwa, K. Kawaguchi, N. Shah, P. S. Yu, and B. Hooi, "Sketch-based anomaly detection in streaming graphs," 2022.

[62] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[63] Y. Liu, Z. Li, S. Pan, C. Gong, C. Zhou, and G. Karypis, "Anomaly detection on attributed networks via contrastive self-supervised learning," *IEEE transactions on neural networks and learning systems*, vol. 33, no. 6, pp. 2378–2392, 2021.

[64] H. Wang, C. Zhou, J. Wu, W. Dang, X. Zhu, and J. Wang, "Deep structure learning for fraud detection," in *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 567–576, IEEE, 2018.

[65] X. Yuan, N. Zhou, S. Yu, H. Huang, Z. Chen, and F. Xia, "Higher-order structure based anomaly detection on attributed networks," in *2021 IEEE International Conference on Big Data (Big Data)*, pp. 2691–2700, IEEE, 2021.

[66] K. Liu, Y. Dou, Y. Zhao, X. Ding, X. Hu, R. Zhang, K. Ding, C. Chen, H. Peng, K. Shu, L. Sun, J. Li, G. H. Chen, Z. Jia, and P. S. Yu, "Bond: Benchmarking unsupervised outlier node detection on static attributed graphs," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27021–27035, 2022.

[67] K. Liu, Y. Dou, Y. Zhao, X. Ding, X. Hu, R. Zhang, K. Ding, C. Chen, H. Peng, K. Shu, G. H. Chen, Z. Jia, and P. S. Yu, "PyGOD: A python library for graph outlier detection," *arXiv preprint arXiv:2204.12095*, 2022.

A DARPA Dataset

We utilize the Engagement 3 (E3) [15] data collection released by the DARPA Transparent Computing Program [36]. This engagement involved a professional red team conducting multiple cyber attacks in a simulated enterprise environment. The collection contains audit data from multiple hosts over a duration of up to two weeks. To mix benign and malicious audit logs, a scripted set of benign activities, such as web browsing and emailing, was continuously running on each host in parallel to manual attacks. We use the vast collection of system calls occurring in two Linux-based hosts, THEIA and TRACE. These datasets come as a set of temporally ordered .json files containing audit data in the DARPA Common Data Model format [37].

Table 3: Statistics of the DARPA E3 datasets. Cybersecurity experts labelled individual edges as malicious or benign using information on the execution of the malicious activities. $|\mathcal{T}|$ is the number of node types, $|\mathcal{R}|$ the number of relationship types. We extract 18 node features for Sockets, 23 for Files and 21 for Processes.

Dataset	Nodes	Edges	Unq. Edges	Anom. Edges	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{R}\rvert$	Unq. Timestamps	Duration
THEIA	1,043k	8,416k	3,982k	211	3	8	1,766k	247h
TRACE	3,926k	10,102k	6,663k	262	3	10	2,838k	264h

Pre-processing. Given the volume of system audit data, we pre-process them; note that such log reduction is active area of research [38] with no clear winner. We use the pre-processing steps of [21] and transform the remaining audit records into a provenance graph containing three types of nodes, i.e., $\{File, Process, Socket\}$ and 32 types of edges corresponding to Linux system calls. Note that not all of system call types occur during the data collection, see Table 3). We further reduce the graph by aggregating the edges between a process and the connected sockets having the same IP address if they occur within a time window of one second; this eases a problem where port-scanning activities would generate thousands of edges in very short bursts. The pre-processing removes around 2M edges from TRACE and 6M edges from THEIA; we report in Table 3 the statistics of the preprocessed datasets.

Feature extraction. In addition to node and edge types, we extract node-specific categorical features and encode them using a one-hot representation. For sockets we encode the IP address as public, private or malformed. The encoding scheme of socket ports involves mapping them to either specific categories designated for ports associated with services available in the DARPA-simulated network environment (e.g., `ssh` on port 22, `http` on port 80) or to one of the port intervals defined by the Internet Assigned Numbers Authority. Files are grouped into categories based on the semantic meaning of their extensions (e.g., images, executables) as well as their root directories (e.g., `/var`, `/lib`). Individual categories are also created for well known Linux system processes and other processes that are the most common in the environment. Finally, in order to accommodate unforeseen instances during inference and support the inductive setting, a dedicated category labeled as "other" is reserved for each of these categorical variables. The feature extraction process extracts 18 node features for Sockets, 23 for Files and 21 for Processes.

Ground truth. Despite the widespread utilization of the DARPA dataset, we could not obtain the edge-level ground truth anywhere online, even after contacting several researchers involved with it. The original ground truth for the DARPA dataset consists of a very *general* description of attack steps (e.g., the executed terminal commands), provided in a PDF file. Such description does not allow to straightforwardly label single events as benign or malicious, that is why we relied on an internal team

of security experts. Due to space constraints, we cannot report details of the labelling process in this appendix but we will provide them at our code repository.

Data splits. We partitioned the data into distinct sets for training, validation, and testing. To guarantee the absence of attacks in the training set, we make sure the test set spans from the JSON file in the raw data that contains the earliest presence of an attack until the end of the data (see Figure 2). The rest of the data (before the first attack) is split by time into training and validation sets, following a split ratio of 0.85 / 0.15. The splitting leads to a train/val/test edges split of 0.366 / 0.079 / 0.555 for THEIA and 0.670 / 0.123 / 0.207 for TRACE.

B The TGN Framework

We first describe the TGN architecture for the homogeneous graph. This architecture comprises (i) a message module, which computes messages from events, (ii) a memory module, which stores a memory vector $\mathbf{s}_i(t)$ for each encountered node i, and (iii) a graph progagation module, which aggregates information from the neighborhood $\mathcal{N}_i(t)$ to produce the final node embedding.

B.1 Message module

Whenever a new event occurs at time t, messages are computed to update the memory of both the source node i and the target node j:

$$\begin{aligned}\mathbf{m}_i(t) &= \mathbf{s}_i\left(t^-\right) \| \, \mathbf{s}_j\left(t^-\right) \| \, \text{MLP}(t - t_i) \| \, \mathbf{e}_{ij}(t), \\ \mathbf{m}_j(t) &= \mathbf{s}_j\left(t^-\right) \| \, \mathbf{s}_i\left(t^-\right) \| \, \text{MLP}(t - t_j) \| \, \mathbf{e}_{ij}(t).\end{aligned} \tag{4}$$

Here t_i and t_j are the timestamps of the last interactions of node i and node j respectively, and $\mathbf{s}_i\left(t^-\right), \mathbf{s}_j\left(t^-\right)$ are the memory vectors of nodes i, j computed in the previous update. We identify with $\|$ the concatenation operator.

If there are multiple events in a batch involving the same node, the corresponding messages are aggregated by the aggr function to form a single message:

$$\overline{\mathbf{m}}_i(t) = \mathrm{agg}\left(\mathbf{m}_i\left(t_1\right), \ldots, \mathbf{m}_i\left(t_b\right)\right), \tag{5}$$

where agg can be implemented as mean, sum, or last.

B.1.1 Memory

The memory is responsible for storing and updating the representation of node histories. For such a reason, the memory is updated after every event by transforming the previous memory state and the incoming messages from the previous module. Recalling that $\overline{\mathbf{m}}_i(t)$ is the incoming message computed by node i at time t from the message module and $\mathbf{s}_i(t^-)$ is the memory state of node i before time t, then the new memory representation is computed as:

$$\mathbf{s}_i(t) = \mathrm{mem}\left(\overline{\mathbf{m}}_i(t), \mathbf{s}_i\left(t^-\right)\right), \tag{6}$$

where mem is an RNN-based architecture, e.g., GRU [39].

B.1.2 Graph propagation module

After the memory of a node is updated with the new events, the node representation is enriched with the information coming from the neighborhood by the graph propagation module. The final node embedding, $\mathbf{z}_i^\ell$, of a node i at time t and ℓ-th layer is computed by using node features and memory states of the neighbors:

$$\mathbf{z}_i^\ell = f\left(\mathbf{z}_i^{\ell-1}, \bigoplus_{j \in \mathcal{N}_i(t)} g\left(\mathbf{z}_i^{\ell-1}, \mathbf{z}_j^{\ell-1}, \mathbf{s}_i(t), \mathbf{s}_j(t), \mathbf{e}_{ij}(t)\right)\right), \tag{7}$$

where g computes the message of each node that is dispatched among the neighbors, f is the function that update the node embedding with the neighborhood representation, and $\bigoplus$ is an aggregation

invariant function, e.g., sum or mean. The node embeddings are initialized with node features, i.e., $\mathbf{z}_i^0 = \mathbf{x}_i$.

As proposed in [11], the graph propagation module can be implemented as a GNN with transformer multihead attention [40]. Thus, node embedding $\mathbf{z}_i^\ell$ are updated as:

$$\mathbf{z}_i^\ell = \sigma\left(\mathbf{W}_s^\ell \mathbf{z}_i^{\ell-1} + \sum_{j \in \mathcal{N}_i(t)} \alpha_{ij} \left(\mathbf{W}_n^\ell \mathbf{z}_j^{\ell-1} + \mathbf{W}_e^\ell \hat{\mathbf{e}}_{ij}\right)\right), \tag{8}$$

where σ is an activation function, as usual; the initial node embedding is the result of the concatenation between the memory state and the node features, i.e., $\mathbf{z}_i^0 = \mathbf{s}_i(t) \| \mathbf{x}_i$; and $\hat{\mathbf{e}}_{ij} = \mathbf{e}_{ij}(t) \| \text{MLP}(t - t_j^-)$ is the new edge representation computed as the concatenation between the original edge attributes and a learned embedding of the elapsed time from the previous neighbor interaction. α_{ij} is the attention coefficient computed as:

$$\begin{aligned} \mathbf{q} &= \mathbf{W}_q^\ell \mathbf{z}_i^{\ell-1}, \\ \mathbf{K} &= \mathbf{W}_k^\ell \mathbf{z}_j^{\ell-1} + \mathbf{W}_e^\ell \hat{\mathbf{e}}_{ij}, \\ \alpha_{ij} &= \text{softmax}\left(\frac{\mathbf{q}^\top \mathbf{K}}{\sqrt{h}}\right), \end{aligned} \tag{9}$$

with h the hidden size of each head. By drawing a connection with Eq. 7, we can identify $\bigoplus$ to be the sum operator, g the multihead attention operator, and f is the aggregation of linear transformations and sum operators.

In the following, we present extensions of the TGN framework [11] that enable TGN to account for edge directionality and to model heterogeneous data.

B.2 Details of Adapted HGT Layer

Using the notation of Eq. 8, the heterogeneous convolution layer in [34] for an edge (i, j) or type r whose nodes are of types τ_i, τ_j is described by the following formula (superscript ℓ indicating the layer number is omitted for parameters):

$$\mathbf{z}_i^\ell = \mathbf{z}_i^{\ell-1} + \sigma\left[\mathbf{W}_A^{(\tau_j)}\left(\bigoplus_{j \in \mathcal{N}_i(t)} \Big(\text{Attn}(i, r, j) \cdot (\text{Msg}(i, r, j)\Big)\right)\right]$$

with

$$\begin{aligned} \text{Msg}(i, r, j) &= \|_{k \in [1,h]} \text{Msg-head}^k(i, r, j), \\ \text{Msg-head}^k(i, r, j) &= \mathbf{W}_M^{(\tau_i)} \mathbf{W}_{\text{MSG}}^{(r)} \mathbf{z}_i^{\ell-1}, \end{aligned}$$

where superscripts $(\tau_i), (\tau_j), (r)$ indicate that one linear layer is learned across each source node, destination node or edge type. The subscripts A, M are taken from [34], A indicates the final layer output transformation, M refers to the *values* path in the attention block. We replace Msg-head^k with the following in order to account for edge features:

$$\text{Msg-head}^k(i, r, j) = \mathbf{W}_M^{(\tau_i)} \mathbf{W}_{\text{MSG}}^{(r)} \mathbf{z}_i^{\ell-1} \| \mathbf{W}_e \mathbf{e}_{ij}.$$

Note that we do not condition $\mathbf{W}_e$ on attention heads or on edge types, so that only one transformation is learned and its representation is shared across heads and types. We leave corresponding extensions as future work.

B.3 Experiment details

We implement all methods in Pytorch Geometric [41] and perform model selection with the parameters in Tab. 5. We report the runtime of the considered models in Tab. 6.

Table 4: Description of the confusion matrix for the anomaly detection task. $\epsilon = (i, j, t)$ are all edges in $\mathcal{G}$, $f_\theta(\epsilon)$ is the probability score returned by the model for edge ϵ; $\tau \in [0, 1]$ is the decision threshold.

		Ground Truth	
		Malicious ($\epsilon \in g_\mathcal{A}$)	Benign ($\epsilon \in \mathcal{G} \setminus g_\mathcal{A}$)
Predict	Anomalous ($f_\theta(\epsilon) < \tau$)	True positive (TP)	False positive (FP)
	Normal ($f_\theta(\epsilon) \geq \tau$)	False negative (FN)	True negative (TN)

Table 5: Grid of parameters for model selection, best parameters in **bold**. For **TGN**, we also search through different aggregation functions (agg in Eq. 5) [**last**, mean, max].

Architecture	**Learning rate**	**# of layers**	**Emb. dim.**	**Neigh. sampler size**
MLP	[$\mathbf{10^{-2}}$, 10^{-3}]	[1, **2**, 3]	100	n.a.
GAT	[10^{-3}, $\mathbf{10^{-4}}$]	[1, **2**]	100	[**10**, 20, 30]
RGCN	[10^{-3}, $\mathbf{10^{-4}}$]	[**1**, 2]	100	[**10**, 20, 30]
TGN	[10^{-4}, $\mathbf{10^{-5}}$]	[**1**, 2]	100	[**10**, 20]

Table 6: Runtime (TRACE dataset) and model size.

Metric	**# of parameters**	**Time per epoch (s)**
MLP	14 901	886
GAT	85 413	5 314
RGCN	626 313	6 656
TGN†§ (**TGAT**†)	114 657	2 107
TGN†	244 113	5 260
TGN§ (**TGAT**)	125 757	2 331
TGN	255 213	5 530
OHD-TGN	255 267	5 532
DIR-TGN	348 514	6 249
Hetero-TGN	3 141 313	28 804
HGT-TGN	1 056 148	12 065

C Related Work

Temporal graph learning. While early models for learning on dynamic graphs focused on the discrete-time case (DTDG) [42], where the graph evolution is represented by snapshots taken at different times, many approaches later focused on the continuous case (CTDG). A number of approaches for CTDG use sequence-based approaches, where for an edge appearing, a sequence model updates node representations of the source and destination node [43, 44, 45, 46]. These methods can be limited, as information is not propagated past the neighborhood of the nodes connected by the new edges, leading to embeddings becoming outdated. To address this problem, another category of approaches propagates information across neighborhoods of the nodes involved. A first set of methods in this category uses random walks on the temporal graph to propagate information [5, 10]. Other approaches explicitly use GNNs in the architecture to deal with information propagation; notable examples in this category include TGAT [4], which uses the graph attention layer introduced in [35], and TGN [11], which combines the TGAT graph convolution to propagate information, and a sequential model to store node-wise memories. Recent advances in the CTDG domain focus on improving the expressiveness, performance and evaluation of methods. APAN [47] and SDGNN [48] suggest to trigger node memories updates to further neighborhoods of the nodes involved in the new edge. Temporal models with better expressiveness are proposed in [49] and [50], respectively with the addition of injective updates and by showing the advantages of encoding sequential information

before graph propagation. On the evaluation side, better temporal model validation guidelines have been suggested in [19], we use these in Sec. 3 of this paper and contribute to better evaluation with our datasets.

Heterogeneous temporal models. In heterogeneous graphs, entities belong to different types which provide rich semantic information. *Knowledge graphs* [51] are the most notable example of heterogeneous graphs. Early methods tackling the heterogeneous-case generalized known GNN architectures to account for entity types, e.g., RGCN [33] and RGAT [52]. Motivated by the limitations of these straightforward approaches [53], many later methods leveraged *meta paths* [54, 55, 56], more expressive node/relation encodings [57] or even graph transformers [58]. Currently, few applications of temporal learning to heterogeneous graphs exists, with one of few examples being temporal knowledge graph learning [59]. Many open challenges remain in the heterogeneous settings [53], including the applications of heterogeneous methods to the temporal domain. To the best of our knowledge, ours is one of the first works in graph learning for the CTDG domain and in the heterogeneous settings, where both nodes and edges are associated with types with the goal of anomaly detection.

Graph Anomaly Detection. While in the past a number of graph anomaly detection approaches were tailored for the streaming events case [24, 60, 25, 23, 61], they do not take into account node features or make strong assumptions about the anomalies. In recent years a number of new anomaly detection methods appeared to address detection of outliers in graphs, many leveraging deep learning [62, 63, 21, 64]. The survey in [12] categorizes graph anomaly detection methods in a task-oriented way, distinguishing approaches based on which anomalous graph objects they can detect (i.e., anomalous nodes, edges, subgraphs or entire graphs) and whether they can operate on dynamic graphs. Most deep learning methods work in an unsupervised fashion relying on AutoEncoder backbones and optimizing a reconstruction error [62, 64], while other use negative sampling of graph objects (nodes, edges, neighborhoods) to be trained in a self-supervised manner [63, 65]. A large benchmark of many Graph Anomaly Detection approaches for the *static* or Discrete-TDG use-case is provided in [66] with the help of the PyGOD framework [67]. We hope our paper can foster additional anomaly detection research in the Continuous-TDG scenario.

D Notes on Threat Detection Baselines Comparison

We were not able to compare our method to other threat detection baselines as most related work treats the detection problem under different assumptions. StreamSpot [24] and Unicorn [23] perform graph classification, Holmes [25] path classification and Threatrace [22] node classification — in comparison, we deal with fine-grained link prediction. ShadeWatcher [21] is the only work resorting to link prediction on provenance graphs and therefore the closest candidate to comparison to our approach. Unfortunately, contrary to what stated in the authors' repository, and even after contacting the authors, we could not reproduce their results.