EULER: Detecting Network Lateral Movement via Scalable Temporal Link Prediction

ISAIAH J. KING and H. HOWIE HUANG, The George Washington University, USA

Lateral movement is a key stage of system compromise used by advanced persistent threats. Detecting it is no simple task. When network host logs are abstracted into discrete temporal graphs, the problem can be reframed as anomalous edge detection in an evolving network. Research in modern deep graph learning techniques has produced many creative and complicated models for this task. However, as is the case in many machine learning fields, the generality of models is of paramount importance for accuracy and scalability during training and inference. In this article, we propose a formalized approach to this problem with a framework we call EULER. It consists of a model-agnostic graph neural network stacked upon a model-agnostic sequence encoding layer such as a recurrent neural network. Models built according to the EULER framework can easily distribute their graph convolutional layers across multiple machines for large performance improvements. Additionally, we demonstrate that EULER-based models are as good, or better, than every state-of-the-art approach to anomalous link detection and prediction that we tested. As anomaly-based intrusion detection systems, our models efficiently identified anomalous connections between entities with high precision and outperformed all other unsupervised techniques for anomalous lateral movement detection. Additionally, we show that as a piece of a larger anomaly detection pipeline, EULER models perform well enough for use in real-world systems. With more advanced, yet still lightweight, alerting mechanisms ingesting the embeddings produced by EULER models, precision is boosted from 0.243, to 0.986 on real-world network traffic.

CCS Concepts: • **Security and privacy** → **Network security**; *Intrusion detection systems;*; • **Computing methodologies** → Neural networks; **Anomaly detection**

Additional Key Words and Phrases: Lateral movement detection, graph neural network, temporal graph

ACM Reference format:
Isaiah J. King and H. Howie Huang. 2023. EULER: Detecting Network Lateral Movement via Scalable Temporal Link Prediction. *ACM Trans. Priv. Sec.* 26, 3, Article 35 (June 2023), 36 pages.
https://doi.org/10.1145/3588771

This article is an extension of a NDSS'22 conference paper by King & Huang [44]. There has been significant addition to the conference paper, including new experiments augmenting the output of the original models and measuring their scalability. We show that by continuing the work done in the conference paper, and applying the framework to a more complex pipeline, we can attain precision high enough for use in real-world systems. There is also additional analysis of our initial results and extended discussions using new diagrams and tables to improve readability throughout the article.

This work was supported in part by DARPA under agreement number N66001-18-C-4033 and National Science Foundation grants 1618706, 1717774, and 2127207. The views, opinions, and/or findings expressed in this material are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense, National Science Foundation, or the U.S. Government.

Authors' address: I. J. King and H. H. Huang, Science & Engineering Hall, Dept. of Electrical & Computer Engineering, 800 22nd St NW Washington, DC 20052; emails: {iking5, howie}@gwu.edu.

2471-2566/2023/06-ART35 $15.00
https://doi.org/10.1145/3588771

1 INTRODUCTION & MOTIVATION

Lateral movement is a key stage of the MITRE ATT&CK framework [72] describing the behavior of **advanced persistent threats (APTs)**. At its core, lateral movement is malware propagating through a network to spread onto new computers in an attempt to find the adversary's target. This may involve pivoting through multiple systems and accounts in a network using either legitimate credentials or malware to accomplish the task [9]. As it is one of the final steps in the kill chain before complete compromise, detecting it early is of critical importance.

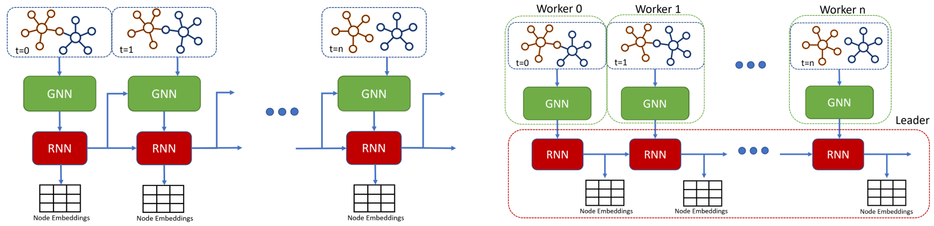
A plethora of machine learning approaches to intrusion detection exist, both signature-based models [18, 45, 56, 57, 79, 80] and anomaly-based [13, 16, 28, 37, 58, 67, 81]. These latter techniques are especially well-suited for lateral movement detection, as APT techniques such as "Pass the Ticket" [7], or even just using stolen credentials [8], are very difficult to formalize into signatures for signature-based intrusion detection systems [9].

The most robust way to detect malware propagation is not to exhaustively list every known malicious signature correlating with it; rather, it is to train a model to learn what normal activity looks like and to alert when it detects behavior that deviates from it. However, detecting anomalous activity in an enterprise network presents unique challenges. The data involved both during training and the implementation of an anomaly-based intrusion detection system is enormous. Often the log files that such a system would require as input are terabytes large. To be useful, a lateral movement detection model must be highly scalable to accommodate such large data. Additionally, when viewed as a classification problem, any such system would have to be highly precise. Millions of events occur in an enterprise network on any given day, and only a fraction of a percent of all interactions are ever anomalous [17]. Therefore, a model must have an extremely low rate of false alerts so as not to overwhelm its users.

In this work, we formulate anomalous lateral movement detection as a temporal graph link prediction problem. Interactions occurring in discrete units of time on a network can be abstracted into a series of graphs $\mathcal{G}_t = \{\mathcal{V}, \mathcal{E}_t\}$ called snapshots, where $\mathcal{V}$ is the set of entities in the network that had interactions $\mathcal{E}_t = \{(u, v) \in \mathcal{V}\}$ during a set period of time, t. A temporal link prediction model will learn normal patterns of behavior from previous snapshots and assign likelihood scores to edges that occur in the future. Edges with low likelihood scores correlate to anomalous connections within the network. As Reference [16] points out, these anomalous connections are often indicative of lateral movement. As we will later show, this reframing of the problem improves precision over standard anomaly-based intrusion detection techniques.

Recent approaches to temporal link prediction combine a **graph neural network (GNN)** with a sequence encoder such as a **recurrent neural network (RNN)** to capture topological and temporal features of an evolving network. However, these approaches are either reliant on RNN output during the GNN stage of embedding [35] or merely incorporate GNNs into the RNN architecture [19, 63, 68]. As Figure 1(a) illustrates, these models are necessarily sequential and, unfortunately, cannot scale to the large datasets that they would need to process to be useful lateral movement detectors.

Proposed solution. To address this problem, we have observed that the most memory-intensive part of existing architectures occurs during the message-passing stage within the GNN. Furthermore, there exists an imbalance between the massive size of node input features and the comparatively minuscule topological node embeddings. This means the most work and the most memory usage occurs in the GNN before the simpler forward pass of the RNN is calculated, necessarily in serial. If several replicated GNNs operate on snapshots independently, then they can execute concurrently, as shown in Figure 1(b). Amdahl's Law [11] would suggest that by distributing such a large portion of work, performance improvements will ensue.



(a) Sequential temporal link predictors

(b) Euler parallel framework

Fig. 1. (a) Prior approaches rely on RNN output during the GNN stage of embedding or merely incorporate GNNs into the RNN architecture, which forces models to work in serial, one snapshot at a time. In contrast, (b) the Euler framework can utilize several worker machines to hold consecutive snapshots of a discrete temporal graph. These workers process snapshots in parallel through a replicated GNN shared across each machine. The output of these GNNs is returned to the leader machine, which runs them through a recurrent neural network to create temporal node embeddings that may be used for link prediction. The framework is explained in detail in Section 5.

In this work, we have developed Euler,[1] a formalized approach for scalable dynamic link prediction and anomalous edge detection. The framework involves stacking a model-agnostic GNN upon a sequence encoder such as an RNN. In this way, a network's topology at discrete moments in time is encoded by the GNN, and the dynamic changes in connections are encoded by the RNN. The embeddings produced by this model provide prior probabilities for future states of the network given what is embedded about the past structure. Most importantly, the framework is designed such that GNNs may be replicated across several worker machines, and execute independently, allowing disjoint sets of snapshots to be processed concurrently. When the GNNs' work occurs in parallel for each snapshot, the topological data for an entire series of graphs can theoretically be encoded in the time it takes to encode the snapshot with the most edges. With these immense performance enhancements, detecting anomalous user activity in industry-scale, real-world networks using powerful GNN models becomes tractable.

Experimental evaluation. There are two things this work aims to show: that despite its simplicity, Euler can outperform slower, more complicated temporal link prediction methods, and that link prediction is an effective method for anomalous lateral movement detection. To evaluate the first contention, we compare the effectiveness of a model following the Euler framework against several state-of-the-art temporal link prediction models on three general-purpose temporal graph datasets. The results of these experiments show that despite its simplicity, the Euler framework performs as well or better than the state-of-the-art. With Euler proven as an effective link prediction model, the next step is to show link prediction can be used for lateral movement detection in larger, real-world event logs.

Implementation as an intrusion detection system. To evaluate the second contention, we test several models following the Euler framework on the LANL comprehensive multi-source cyber-security events dataset [41]. This dataset includes labeled event data from 58 consecutive days of real-world computer network use interspersed with red team activity. There are approximately 1.6 billion events in total. This is a test not only of the Euler framework's precision, but also its ability to scale. Our tests show that Euler-based models outperform prior works in both precision and compute time.

We further augment these results by using Euler as a stage in a longer anomaly detection pipeline. The embeddings produced by link prediction models contain vital information about the

[1] Source code available at https://github.com/iHeartGraph/Euler.

structure of dynamic graphs, but Reference [62] points out that anomaly detection requires more data than just graph structure. By using the outputs of Euler models as inputs to an additional distributed anomaly detection network, we can extract even more information. These experiments showed that this strategy significantly improves precision and remains scalable to large datasets.

In summary, our research contributions are:

- We present, to the best of our knowledge, the first use of temporal graph link prediction for anomaly-based intrusion detection. Other research in applying graph analytics to anomaly detection either does not consider the temporal nature of the data or does not use a powerful GNN model. By incorporating both elements into Euler, models built on this framework outperform other unsupervised anomaly-based intrusion detection systems and yield more informative alerts.
- We demonstrate that for temporal link prediction and detection, the simple framework we propose is equally or more accurate and precise than state-of-the-art temporal graph autoencoder models. Euler models attain higher area under the curve and average precision scores by 4% in inductive link detection tests, and about equal metrics to the state-of-the-art in transductive link prediction tests.
- We propose a scalable framework for distributed temporal link prediction for use on big data. The Euler framework is simple and makes message-passing lightweight even over large graphs. By breaking edge lists into temporal chunks and processing them in parallel, the computational complexity of the message-passing stage of the model is theoretically bound by only the snapshot with the most edges. Other optimizations allow the RNN to operate in parallel with some of the GNN workers, further improving performance.

Please note that this work is an extension of our previously published conference paper [44]. Novel contributions of this work are the additional experiments in Sections 9 and 10 as well as a more complete literature review in Section 2, and a discussion of the assumptions we make about the threat model in Section 7. Additionally, Section 8 contains extended discussion and analysis of the results from our previous work.

2 RELATED WORK

2.1 Intrusion Detection Systems

Many techniques exist to use machine learning on cyber-security data for intrusion detection. They can be broadly categorized as **signature-based intrusion detection systems (SIDS)** and **anomaly-based intrusion detection systems (AIDS)**.

Signature-based intrusion detection systems are used by many popular monitoring systems [18]. They have a distinct advantage in speed, simplicity, and effectiveness so long as the attack has been previously observed [43]. The primary focus of research involving graphs for signature-based intrusion detection, is on the analysis of provenance graphs [45]. References [56, 57] accomplish this through subgraph matching. They generate provenance graphs from kernel log files that can be quickly identified as malicious if they contain a previously identified subgraph correlating to compromise. References [79, 80] follow a similar approach, however, they use vector embeddings of the provenance graphs to measure similarity between observed and known malicious graphs. This approach to subgraph matching, while sometimes less sound than exhaustive search, is much faster. We use a similar embedding approach in this work. Despite these methods' effectiveness at identifying known threats, advanced attackers will endeavor not to use a method with a previously discovered signature. For this reason, we turn to anomaly-based intrusion detection methods.

Anomaly-based intrusion detection systems must first define a baseline for normal behavior, then generate alerts when events occur that significantly deviate from this baseline. The definition for normalcy is highly contingent on the abstraction used to represent the system. Frequency-based AIDS define normalcy through observed distributions of events' frequencies or other stochastic processes present in the network [37, 81]. Midas [14] incorporates structural data with frequency counts; however, it can only detect bursts of anomalous events and would struggle to identify individually anomalous connections. Similarly, Whisper [29] analyzes ongoing packet features in the frequency domain; they find anomalous traffic incurs phase anomalies in sliding windows that are difficult to hide. Supervised learning approaches such as References [10, 24, 55] analyze features of individual events with data-mining approaches. Like the EULER approach, many unsupervised systems use deep autoencoders. However, they, too, embed event features rather than network interactions [13, 28]. Kitsune [58] uses temporal patterns in addition to event features; however, temporal information is used as an input feature rather than for sequence encoding.

Network-level anomaly-based intrusion detection systems in the field of graph analytics are lacking. Only Reference [16] has proposed a method for detecting anomalous events in host logs that leverages the rich graph structure inherent to the medium. This approach only considers the network as a static graph, so edge embeddings that occurred at different points in time always have the same anomaly score.

2.2 Temporal Link Predictors

Temporal link prediction is defined as finding a function that takes two nodes in a graph at a point in time, and outputs the probability of an edge existing between them. This problem has applications in fraud detection [35], contact tracing [19], traffic prediction [84], and as we will show, anomalous lateral movement detection.

A great deal of research has been done in this field, however, there are very few approaches that use graph neural networks. Despite the massive improvements in accuracy they offer [47], many papers still opt to use traditional deep learning approaches. Models such as Triadic Closure [85], DyLink2Vec [65], and DynGEM [33] use n-tuples of nodes known to be connected as inputs to generate embeddings using **multilayer perceptrons (MLPs)**, rather than taking a more global relational approach. Their loss functions account for this somewhat through neighborhood sampling, but MLPs cannot leverage the rich topological information the same way a graph convolutional operator can. References [32, 53] also use MLPs to operate on adjacency vectors, however, they also incorporate a recurrent neural network. This way, both topological and temporal information is present in the embeddings. Both models take nodes' full adjacency vectors as inputs, and their embeddings represent the dynamics of nodes' neighbors over time. Our work extends this idea and incorporates better structural information by instead using a graph convolutional operator.

Additional studies have been done on the effectiveness of random walk-based embeddings over time. Reference [61] found that enforcing temporal order in random walks, with no other changes, outperforms LINE [74], DeepWalk [64], and node2vec [34]. However, as Reference [70] points out, random walk-based approaches are very slow and memory-intensive. Furthermore, these approaches cannot take into account node features and lose out on the rich information they can provide. Additionally, Reference [61] only provides one embedding of each node; that is to say, the embeddings of nodes do not change over time. This may not be the best approach, as nodes' functions in a network may be dynamic. They can, however, consider edge weights, which is something the MLP approaches lack. These are also dynamic and provide important information about the future of the graph's structure. This is something we keep in mind with the framework we propose and is easily incorporated into EULER models depending on the choice of GNN.

Many temporal link prediction approaches that do use graph neural networks are preoccupied with incorporating them into recurrent neural networks in lieu of linear layers. References [19, 35, 68] all modify existing RNN architectures to incorporate graph convolutions. VGRNN [35], however, uses a more modular approach, much like we are proposing. The inputs passed to their graph RNN are embeddings generated with a variational graph autoencoder [48]. In this way, the information propagated through time by the RNN is strongly influenced by the changing structure of the graph. However, the topological encoders are dependent on the output of the recurrent layer at the previous timestep, so they cannot be implemented with our framework. EvolveGCN [63] takes an interesting approach where the parameters of the GCN themselves are passed through a recurrent unit. Their technique shows vast improvements over the state-of-the-art in link prediction tests where edges are unknown for long periods of time. However, we are primarily interested in anomaly detection, where the structure of a graph at any given time slice is likely available. We will show that their model does not perform as adequately as others in this domain.

There are several models that could readily fit into the framework we propose. However, none take advantage of the potential parallelism present in their own models. The first of these was Reference [68], which mentions and tests stacking Chebyshev filter convolutions on an LSTM, however, much of this work is focused on modifying LSTMs to incorporate graph convolutions. DySAT [66] uses a graph attention network fed into temporal self-attention layer; it follows the independent topological encoder to serial temporal encoder framework we propose. DyGGNN [73] stacks gated graph neural networks on an LSTM, however, their model is only tested on full graph classification. T-GCN [84] consists of a GCN stacked on a GRU following the framework, but it was used for node regression, not temporal link prediction. However, we will show their model is an excellent link predictor as well. It is evident that many state-of-the-art models already use this approach in multiple problem domains. This means by formalizing and optimizing it, it has the potential to improve several existing works. These papers are the main motivation for our work.

2.3 Distributed Machine Learning

Techniques for distributed machine learning can generally be separated into two broad categories: data parallel and model parallel [77, 82]. Under model parallelism, portions of a model's parameters are split across multiple workers and updated by those same workers. This is how we implement the separation between the GNN and RNN layers. With data parallelism, one assumes that training and real-world data are independent and identically distributed; thus, it is perfectly acceptable to partition training data and use disjoint subsets to train replicas of one model [82]. The forward pass and gradient calculations are done in parallel by multiple workers, and the parameter update stage is done by some centralized or otherwise coordinated means.

One such method to coordinate parameter updates of a replicated model is the parameter server [50]. Here, one machine, the parameter server, holds a master-copy of a model's parameters. Several workers hold replicas of the model and a subset of the training data. These workers then calculate gradients from the loss generated by the data they each hold and send these gradients to the parameter server. The parameter server aggregates the gradients, updates the model's parameters accordingly, and sends the updated parameters back to each replica. The PyTorch **`DistributedDataParallel`** **(DDP)** further extends this idea [52] using collective communication. This allows workers to broadcast partial gradients as they are backpropagated, allowing for communication during computation. Because their method enjoys near-linear scalability, we use it to implement the data parallel within the GNN layer of Euler.

3 BACKGROUND

Our approach to anomaly-based intrusion detection represents network activity as *discrete temporal graphs*. From there, detecting evidence of lateral movement betrayed by anomalous network activity is equivalent to anomalous edge detection, which we accomplish via *temporal link prediction*. In this section, we will briefly define these terms and define the symbolic notation we will use throughout this work.

Lateral Movement Detection

Lateral movement is the stage in an attack where an adversary roams the network in an attempt to find high-value hosts, services, and data in an attempt to spread from a compromised node throughout the network [51]. It is very difficult to detect in enterprise systems, because it is a slow process [26] that utilizes normal looking tools [71] such that individual malicious system events do not appear out of the ordinary. However, it is not completely invisible: As the objective of lateral movement is to propagate to more valuable nodes, there are often communications between entities that are not normally related to each other.

References [15, 16, 42, 54] have shown that this stage of the attack is most easily detected in unusual communications, and in particular authentications, between users or machines within a network. When data are represented as a graph, unusual authentications are understood as paths or edges that connect nodes with low link prediction probabilities. For this reason, we are motivated to represent cyber security data as a temporal graph.

Discrete Temporal Graphs

A discrete temporal graph[2] $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, \dots \mathcal{G}_T\}$ is defined as a series of graphs $\mathcal{G}_t = \{\mathcal{V}, \mathcal{E}_t, \mathbf{X}_t\}$, which we refer to as snapshots, representing the state of a network at time t. Here, $\mathcal{V}$ denotes a set of all nodes that appear in the network, $\mathcal{E}_t$ denotes relationships between nodes at time t, and $\mathbf{X}_t$ represents any features associated with the nodes at time t. In this work, all graphs are directed, and some have weighed edges, $W : \mathcal{E} \to \mathbb{R}$ representing edge frequencies during the time period each snapshot encompasses.

Let the interactions $\mathcal{I}$ between users and machines in a network at specific times be represented as a multiset of tuples <*src,dst,ts*>. Here, *src* is an entity interacting with entity *dst* at time *ts*. From this multiset, we can build the temporal graph $\mathcal{G} = \{\mathcal{G}_0, \dots, G_T\}$ with time window δ. The set of all nodes $\mathcal{V}$ is the set of every *src* and *dst* entity that appears in $\mathcal{I}$. The set of edges at time t, $\mathcal{E}_t$ is constrained such that for all edges $(u, v) \in \mathcal{E}_t$ there exists an interaction $< u, v, w > \in \mathcal{I}$ where $t \leq w < t + \delta$.

Temporal Link Prediction

Temporal link prediction is defined as finding a function that describes the likelihood that an edge exists in a temporal graph at some point in time, given the previously observed snapshots of a network. By representing an enterprise network as a temporal graph, we can further extend this definition to encompass anomaly detection. This follows from the assumption that anomalous edges in a temporal graph will have a low probability of occurring given what is known about the network's behavior in the past.

Lateral movement detection with temporal link prediction is then defined as finding a function learned from the temporal graph $\mathcal{G}$ of network activity that predicts the likelihood of future interactions occurring in unseen snapshots. An observed interaction between entities with a likelihood score below a certain threshold is said to be anomalous. These anomalous edges, in the context of network monitoring, are often indicative of lateral movement [16].

[2] For simplicity, for the remainder of this work, we refer to these simply as "temporal graphs".

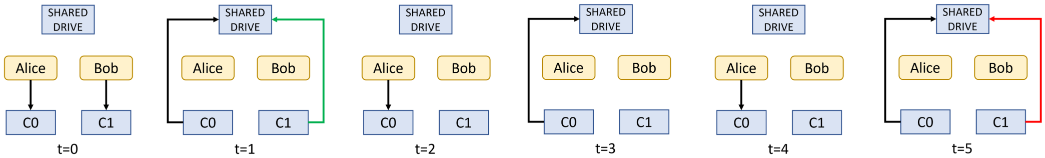


Fig. 2. A simple example of hard-to-detect temporally anomalous activity in a network with three machines and two users. The normal sequence of events is that a user authenticates with a computer, then that computer makes a request to the shared drive. However, suppose C1 is compromised with Mimikatz or a similar malicious process with access to LSASS memory [6]. After Bob's initial connection to C1, his credentials are harvested, and at time $t = 5$, Computer 1 makes a request to the shared drive without Bob's involvement. Though this exact edge has been seen before, in this temporal context it is now indicative of C1's compromise manifested as anomalous network activity.

4 MOTIVATION

Anomaly detection is strongly dependent on how data are represented. The way in which we model the network determines what is considered anomalous activity. An anomaly is simply a deviation from an observed pattern of behaviors; for attacks to appear anomalous, they must cause some unexpected representation of the system to emerge. The way in which the underlying data are represented will ultimately shape and constrain how this is defined. Historically, there have been two main ways of abstracting data for anomaly detection: frequency-based and events-based [17]. Frequency-based methods, such as References [14, 29, 37, 81], define anomalous behavior as activity that significantly deviates from observed temporal patterns. Events-based methods, such as References [10, 13, 24, 28, 55, 58], identify commonalities in features, such as packet count, protocols, and so on, associated with individual events within a system. Anomalies are then defined as events with unexpected features.

The problem with these two approaches is that they ignore the inherently structural nature of network data. This is readily apparent by observing that one of the most influential datasets for intrusion detection [4, 60] has no features for source or destination entities. All that matters with these models are the details of the events themselves in isolation, not the machines between which they occur. It is impossible to capture anomalous interactions between entities in a network using the traditional data representation. Attacks that use stolen credentials, for example, may have features that look entirely normal, and may occur at rates that are non-anomalous as seen in the DARPA Transparent Computing dataset [75]. Indeed, the lateral movement stage of a compromise is often "low-and-slow" and intentionally similar to normal network traffic both in frequency and features, making them indistinguishable from normal activity at the network level when represented with traditional means [26]. During the lateral movement stage of an attack, anomalies instead manifest as unusual *connections* between entities [20]. Networks are webs of relational data, fluctuating over time; the most natural way to represent and analyze relational data is as a graph.

There has been growing interest in static graph-based methods for intrusion detection, such as the work of Bowman et al. 2020 [16]. Here, normalcy is defined in terms of interactions between entities within systems, but time is not considered. Anomalies are defined as edges with low probability given what is known about the graph's structure.

To demonstrate the advantages and disadvantages of the above intrusion detection systems, consider the example shown in Figure 2. The first two time slices show normal activity in the network: First, at $t0$, Alice and Bob authenticate with their computers A and B, then at $t1$ computers A and B make a request to the shared drive. At times $t2$ and $t3$, we see that when Bob does not first authenticate with computer B, it does not communicate with the shared drive. A simple probability distribution is apparent:

$$\begin{aligned} P((C1, SD) \in \mathcal{E}_{t+1} \mid (B, C1) \in \mathcal{E}_t) = 1 \\ P((C1, SD) \in \mathcal{E}_{t+1} \mid (B, C1) \notin \mathcal{E}_t) = 0. \end{aligned} \quad (1)$$

However, in $t4$ and $t5$, something unusual occurs: Computer B requests data from the shared drive without Bob authenticating with it first! Suppose C1 was compromised with Mimikatz or a similar malicious process with access to LSASS memory [6]. After Bob's initial connection to C1, his credentials have been harvested. After lying in wait, with the newfound stolen credentials, the APT could be using attack techniques (as catalogued by the MITRE ATT&CK framework) T1563, remote service hijacking, or attempting T1080, tainting shared content [9].

To detect attacks such as the one in the example, a model would need to consider events with reference to those that occurred previously and with reference to the other interactions within the network. An event between two entities that happens at one point in time cannot be considered identical to the same event occurring in the future under a different global context. Unfortunately, existing graph-based approaches, which do not consider time, and many event-based approaches that look at each event in isolation such as References [16, 28, 58] would see no difference between $(C1, SD)$ at time $t1$ and $(C1, SD)$ at time $t4$. What makes it stand out as anomalous is something lacking in the previous state of the graph—something that would only be detected by considering prior probabilities for the next state of the system.

Frequency-based and event-based approaches such as References [10, 24, 37, 55, 81] would have similar trouble, as they lack the relational data present in the network; Bob's activities a timestep earlier would have little import on the interpretation of the shared drive's activity later on. If the interaction between $(C1, SD)$ at time $t4$ was sufficiently similar to the interaction at time $t1$, then these approaches would see no difference between the two. They lack the ability to capture the importance of interactions occurring between other entities in the network and how they may relate to a separate event.

Our proposed solution, to represent the network as a temporal graph, ensures the global structure of a network at individual points in time is captured without losing the temporal dependencies of the changing connections. We make the more difficult assumption that malicious events can have the same event features as normal ones; if this is the case, then traditional event-based approaches will not work. We also assume attackers can make similar connections, or even the same connections as observed in normal activity, meaning available graph-based approaches and statistical approaches are insufficient. With temporal graphs, assuming they have enough granularity, these problems, as well as those tackled by prior works, are solvable.

5 EULER

In this section, we describe our proposed framework, which we call Euler. This framework aims to learn a probability function conditioned on previous states of a temporal graph to determine the likelihood of an edge occurring at a later state. Furthermore, it is the goal of this work to offer an approach that is not just precise, but also highly scalable. We first describe the basic components of the system, then how they are distributed across multiple machines and how these components interact. Finally, we describe how different training objectives are implemented on the Euler interface.

5.1 The Encoder-Decoder

The Euler framework is a generic extension of the traditional graph autoencoder model [48] to temporal graphs. It consists of a model-agnostic **graph neural network (GNN)** stacked upon a model-agnostic **recurrent neural network (RNN)**. Together, these models aim to find an encoding function $f(\cdot)$ and a decoding function $g(\cdot)$. The encoding function maps nodes in a temporal graph with T snapshots to T low-dimensional embedding vectors. The decoding function ensures minimal information is lost during the encoding process and aims to reconstruct the input snapshots from the latent $\mathbf{Z}$ vectors. More formally, we can describe the behavior of the encoder as

$$\begin{aligned} \mathbf{Z} &= f(\{\mathcal{G}_0, \dots, \mathcal{G}_T\}) \\ &= \text{RNN}(\ [\text{GNN}(\mathbf{X}_0, \mathbf{A}_0), \dots, \text{GNN}(\mathbf{X}_T, \mathbf{A}_T)]\ ), \end{aligned} \tag{2}$$

where $\mathbf{A}_t$ is the $|\mathcal{V}| \times |\mathcal{V}|$ adjacency matrix representation of the snapshot at time t. This $T \times |\mathcal{V}| \times d$ dimensional tensor $\mathbf{Z}$ is optimized to contain information about both the structure of the graph and the dynamics of how it changes over time.

This is enforced by a decoder function, $g(\cdot)$, which attempts to reconstruct the original graph structure given the embeddings. More formally,

$$g(\mathbf{Z}_t) = Pr(\mathbf{A}_{t+n} = 1 \mid \mathbf{Z}_t), \tag{3}$$

where $\mathbf{Z}_t = \mathbf{Z}[t]$ is the embedding of graph $\mathcal{G}_t$ and $n \geq 0$. As was done by References [16, 34, 35, 48], we use the inner product decoding as this $g(\cdot)$ function:

$$g(\mathbf{Z}_t) = \sigma(\mathbf{Z}_t \mathbf{Z}_t^\intercal) = \tilde{\mathbf{A}}_{t+n}, \tag{4}$$

where $\sigma(\cdot)$ denotes the logistic sigmoid function, and $\tilde{\mathbf{A}}_{t+n}$ represents the reconstructed adjacency matrix at time $t+n$. As a consequence of using inner product decoding, the dot product of vectors $\mathbf{Z}_t[u]$ and $\mathbf{Z}_t[v]$ represents the log-odds that an edge, (u, v) exists at time $t + n$. In this way, the $g(\cdot)$ function is used to detect anomalous edges.

5.2 Workflow

The core of the Euler framework is a simple design. It simply stacks the replicas of a model-agnostic GNN that we refer to as the *topological encoder* upon a model-agnostic *recurrent layer* with a few simple constraints. When fit into the leader/worker paradigm[3] with one recurrent layer as the leader, and multiple topological encoders as workers, it has the potential for massive parallelism.

The overall workflow for Euler is shown in Figure 3. It occurs in 5 basic stages: (1) The leader spawns the workers and instructs them on which snapshots to load; (2) the leader initiates the training loop, and workers generate topological embeddings; (3) as the topological embeddings are received, the leader processes them through an RNN; (4) the output of the RNN is sent back to the workers to calculate loss or for scoring; (5) in training mode, loss is returned to the leader to be backpropagated. During evaluation, anomaly scores are returned. In this section, we describe how these 5 steps are implemented in greater detail. For more detailed technical information about the Euler interface, we direct the reader to Table 7 in the conference version of this work [44].

Loading Data

When the program first starts, several processes are spawned. The PyTorch multiprocessing library automatically assigns each spawned process a unique ID from 0 to $k+1$; for convenience the machine with id:0 becomes the leader, and all others are workers. The leader machine, which holds remote references to all workers in addition to the recurrent layer, issues commands to the other processes to spin up all worker instances.

Upon their creation, workers connect to the leader and await instructions. After the initialization of all workers, the leader assigns them contiguous subsets of temporal graph data to load, shown in step 1 of Figure 3. For example, given a set $W = \{w_1, \dots, w_k\}$ of k workers, and a temporal graph split into T snapshots, the leader assigns each worker $s = \left\lfloor \frac{T}{k} \right\rfloor$ snapshots. Then worker

[3] Historically called the master/slave paradigm.

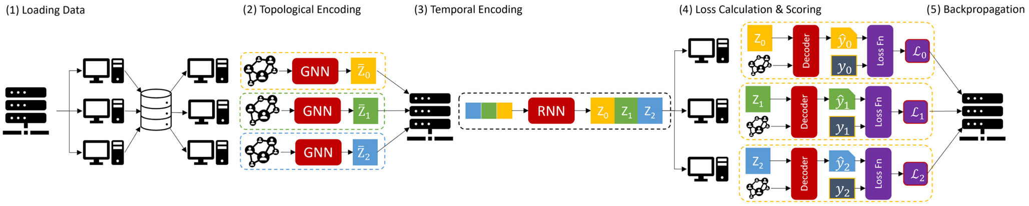


Fig. 3. The complete series of interactions between the leader and worker machines during one training step. In step (1), the leader machine initiates each worker and issues a command for each of them to load a disjoint subset of the snapshots from memory. When this has occurred, the training loop can begin. At step (2), the leader issues a command to each worker to perform a forward pass on their graphs through their GNNs. As they complete this, workers send the topological embeddings, $\bar{\mathbf{Z}}_t$, to an ordered queue in the leader. Upon receiving the embedding for $t = 0$, step (3) begins. The leader runs embeddings from the queue through its RNN as they come in to produce the final $\mathbf{Z}$ embeddings. Then, in step (4), the leader sends the embeddings back to the workers to decode and calculate loss. Finally, (5), the leader and workers perform a backward pass on the aggregated loss functions according to the DDP gradient bucketing algorithm [52]. During evaluation, the steps are nearly identical, but on step (4) workers return their scores for each edge $\hat{y}_t$ instead of loss.

w_i holds $\{\mathcal{G}_{si}, \ldots, \mathcal{G}_{si+(s-1)}\}$. In the likely case where k does not divide T, extra units of work are assigned to workers holding snapshots later in time. This is because the recurrent layer of the model processes the topological encoders' output in order, as it comes in. Thus, the RNN can perform the necessarily sequential forward pass on earlier embeddings while future snapshots are still being processed by workers.

After the workers have been assigned their snapshots, they concurrently read them in. The leader waits for each worker to signal that all data are loaded, then moves on to the next phase of the workflow.

Topological Encoding

During the forward pass—shown in step 2 of Figure 3—the recurrent layer issues asynchronous calls to `forward` on each worker machine. To minimize network traffic, the only thing the leader sends to the workers is an enum representing which partition of edges to process, as some are held out for validation and testing. Workers then process every snapshot they hold. Further reducing network traffic, the matrices returned by the workers are far smaller than those used as inputs, as each worker is essentially a graph autoencoder [48].

For performance optimization, and to ensure consistency, we impose just one constraint for this stage: topological encoders must not be dependent on any temporal information. They provide a purely spatial encoding of the state of each snapshot they hold, using only features observable at a single point in time. With this constraint satisfied, all encoders can operate in parallel, as no one worker is dependent on the output of another. Theoretically, with as many workers as snapshots, the time complexity of a forward pass is constrained only by the snapshot with the greatest number of edges.

Temporal Encoding

The leader maintains an ordered list of future objects that point to the eventual output of the workers and waits for the future pointing to the first embeddings to finish executing. When the leader receives this tensor, it is immediately processed by its RNN, shown in step 3 of Figure 3. Note that so long as tasks are slightly imbalanced such that workers holding later snapshots contain more work units, the leader's recurrent layer can execute concurrently with at most $k-1$ workers' topological encoders. Workers with earlier snapshots hold fewer work units, and therefore finish

ALGORITHM 1: Leader machine forward method

```
def forward(self, workers, partition):
    /* Leader tells each worker to begin executing                              */
    futures = [];
    for $w \in workers$ do
        future = asynchronously execute w.forward(partition);
        futures.append(future);
    /* As workers return their embeddings, the leader processes them in order, as they arrive */
    h=NULL;
    zs = [];
    for $f \in futures$ do
        z, h = self.RNN(f.wait(), h);
        zs.append(z);
    return concat(zs)
```

executing earlier, so the leader can process their outputs while workers holding greater quantities of snapshots further in time are still processing.

When the recurrent layer has finished processing the output of one worker, the hidden state and outputs from the RNN are saved. The leader waits for the next topological embedding to finish processing, then uses the saved RNN hidden state and the next embedding to repeat the process until all workers have finished executing. This procedure is described in more detail in Algorithm 1.

Decoding

When the leader finishes generating the final embeddings, they are sent back to the workers to decode, and if the model is training, to calculate loss. This process occurs in parallel on the worker machines. In general, graph functions such as those used to find edge likelihoods are more compute- and memory-intensive, so we endeavor to run them in parallel whenever possible. This stage is shown in step 4 of Figure 3.

During evaluation, instead of returning loss, the workers return edge likelihoods $\hat{y}_t$ and the ground-truth edge labels y_t. The process for decoding the embeddings is the same, however, loss is not calculated.

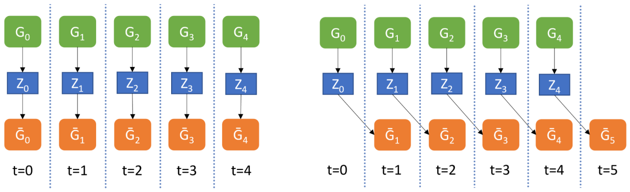
Backpropagation & Evaluation

When loss has been calculated and returned to the leader machine, gradients are calculated via backpropagation, first through the recurrent layer, then components of the loss function generated by each topological encoder are backpropagated in parallel and broadcast between workers in accordance with the bucketing algorithm described in Reference [52]. After the backpropagation step, and collective communication between workers, gradients across all workers' model replicas are equal. Finally, the recurrent layer and the topological encoders all update their parameters using an Adam optimizer [46], and the leader repeats steps 2–5 until convergence.

If the model is in evaluation mode, then the leader machine instead uses the $\hat{y}_t$ likelihoods the workers generate, and the known labels y_t, also returned by the workers, to calculate precision and accuracy metrics, which are saved. In a real-world implementation without labels, it would instead raise alerts on observed edges with likelihoods below a certain threshold.

5.3 Training

There are two modes of training Euler models: as a link *detector* or a link *predictor*. These two modes are distinguished by which $\mathbf{Z}_t$ embeddings are sent to the workers at step 4 to calculate loss.



(a) Link Detection (b) Link Prediction

Fig. 4. How embeddings are generated and the interpretation of their decoded values for static and dynamic models. In link detection, embeddings $\mathbf{Z}_t$ generated from partial snapshot $\hat{\mathcal{G}}_t$ are used for evaluating the likelihood of unseen edges in snapshot $\mathcal{G}_t$. For link prediction, embeddings $\mathbf{Z}_t$ generated from snapshot $\mathcal{G}_t$ are used to predict the likelihood of edges in a future snapshot $\mathcal{G}_{t+n}$ (in this figure, $n = 1$).

Link detectors are inductive; they generate $\mathbf{Z}_t$ using partially observed snapshots $\{\hat{\mathcal{G}}_0, \ldots, \hat{\mathcal{G}}_t\}$ and attempt to reconstruct the full adjacency matrix $\mathbf{A}_t$ with $g(\mathbf{Z}_t)$. This process is illustrated in Figure 4(a). In practice, link detectors would be used for forensic tasks, where one is performing an audit to identify anomalous connections that have already occurred.

Link predictors are transductive; they generate $\mathbf{Z}_t$ using snapshots $\{\mathcal{G}_0, \ldots, \mathcal{G}_t\}$ to predict the future state, $\mathbf{A}_{t+n}$ where $n > 0$. Figure 4(b) illustrates this process when $n = 1$. In practice, they could be used as a live intrusion detection tool, as predictive models can score edges as they are observed—before they have been processed into full snapshots. For example, when $n = 1$, given what has been observed about the network up until time $t - 1$, it is the goal of predictive implementation of Euler to score edges observed at time t. Such a model can use embeddings learned from previous states of the network to process connections as they occur.

To ensure these objectives, the reconstruction loss function aims to minimize the negative log likelihood of Equation (3), where $n = 0$ when training detectors, and $n > 0$ for predictors. For larger graphs, operating over the entire adjacency matrix quickly becomes intractable. Instead, as is done by graph autoencoders [48], we approximate this value by minimizing binary cross-entropy on the likelihood scores for known edges, and a random sample of non-edges at time $t+n$: P_{t+n} and N_{t+n}, respectively. Consider Equation (4) on a per-edge level:

$$P((u,v) \in \mathbf{A_{t+n}} \mid \mathbf{Z}_t) = \sigma(\mathbf{Z}_t^{(u)} \mathbf{Z}_t^{(v)\intercal}), \tag{5}$$

where $\mathbf{Z}_t^{(v)}$ represents the embedding of node v at snapshot t. Then, the reconstruction loss function for snapshot $t + n$

$$\mathcal{L}_t = \mathbb{E}[-\log(Pr(\mathbf{A}_{t+n} \mid \mathbf{Z}_t))] \tag{6}$$

can be approximated by taking the sample mean.

$$\mathcal{L}_t \approx -\frac{-1}{|P_{t+n}|} \sum_{(u,v) \in P_{t+n}} \log P((u,v) \in \mathbf{A_{t+n}} \mid \mathbf{Z}_t) - \frac{-1}{|N_{t+n}|} \sum_{n \in N_{t+1}} \log(1 - P((u,v) \in \mathbf{A_{t+n}} \mid \mathbf{Z}_t)) \tag{7}$$

New negative edges are randomly sampled on each epoch by workers for each snapshot they hold. The leader machine coordinates which slices of the $\mathbf{Z}$ tensor are sent to each worker to generate the probabilities, and each worker independently calculates loss on the training data they hold.

On predictive models, $\mathbf{Z}_t$ represents the latent space of nodes n snapshots in the future. Consequently, predictive models cannot calculate loss on the first n snapshots of the graph. To account for this, the leader pads $\mathbf{Z}$ with a n $|\mathcal{V}| \times d$ zero matrices at the beginning, and the final n matrices are removed before returning the embeddings to the workers. This way, $\mathbf{Z}[t]$ predicts the snapshot

ALGORITHM 2: Calculate loss

```
def worker_loss(zs, n_jobs, workers, n=1):
    /* Pad the Z tensor if predictive */
    zeros = $n \times |\mathcal{V}| \times d$ 0 matrix;
    zs = concat(zeros, zs[:-n]);
    futures = [];
    start=0; end=n_jobs[0]; loss = 0; i=0;
    /* Send the correctly offset embeddings to the workers to calculate loss */
    for $w \in$ workers do
        f = asynchronously execute w.loss(zs[start:end]);
        futures.append(f);
        start=end; end=start+n_jobs[i++];
    return sum([f.wait() for f in futures]) / len(workers)
```

indexed at t on all workers except the one holding the initial snapshots, which ignores embeddings that equal the zero matrix. This process is shown more clearly in Algorithm 2.

5.4 Classification

Though for much of our evaluation, we rely on regression metrics relating to the fitness of scores assigned to edges, it is useful to automate the process of deciding the threshold for what counts as anomalous to obtain classification scores. To this end, when training the model, we hold out one or more full snapshots to act as an extra validation set. Using the final hidden state **h** of the RNN from the training snapshots as input for the validation snapshot, a training partition of edges is passed through the model. From there, finding an optimal cutoff threshold for edge likelihood scores becomes a simple optimization problem.

Given a set of scores for edges that exist in the validation snapshots, but were held out of the training set, and a set of scores for non-edges, the optimal cutoff threshold τ is the one that satisfies

$$\underset{\tau}{argmin} \quad \|(1-\lambda)TPR(\tau) - \lambda FPR(\tau)\|, \tag{8}$$

where $TPR(\tau)$ and $FPR(\tau)$ refer to the true and false positive rate of classification given cutoff threshold τ, and λ is a hyperparameter in $[0-1]$ biasing the model to optimize for either a high true positive rate or a low false positive rate. Experiments have shown that for anomaly detection, where low false positive rates are critical, $\lambda = 0.6$ is very effective. For any metric involving classification for the remainder of the article, this is how classes were determined from the edge likelihood scores, and unless otherwise specified $\lambda = 0.6$.

5.5 High-performance Temporal Encoding

When distributing workloads to workers, it is assumed that each worker has equivalent processing power and memory. We also assume that each snapshot constitutes roughly the same amount of work. This is not always the case, especially with natural graphs that follow cyclical patterns (e.g., the ebb and flow of network activity at night and during the day), however, this provides a rough upper bound on the maximum amount of work one worker is expected to accomplish. For this reason, we use snapshots as the unit of work assigned to each worker. As previously stated, units of work are evenly distributed between workers, with any extra units of work given to workers processing later snapshots; this allows the RNN to process earlier, completed graph embeddings while later embeddings are still being computed. If workers cannot be assumed to be equivalent in

Table 1. Dataset Metadata

Dataset	Nodes	Edges	Avg. Density	Timestamps
FB	663	23,394	0.00591	9
colab	315	5,104	0.01284	10
Enron10	184	4,784	0.00514	11

processing power, or memory, then jobs could be assigned according to a greedy interval partition algorithm [49] or similar, but we leave this as a topic for future work.

Synchronization between workers is handled by PyTorch. Communication between workers to randomly initialize model weights and share gradients is accomplished through the Distributed Data Parallel module [52]. Communication between the workers and the leader is implemented using the PyTorch remote procedure call framework [2], which communicates over TCP via TensorPipes [3]. The leader issues asynchronous calls to workers, who send their output to a socket when their work is concluded. Thus, as the workers' output is in the form of a list of promises pointing to memory addresses, their order is guaranteed. In the event that a worker faults in some way, the leader process will shut down if some timeout period has passed without receiving the worker's result. To prepare for such a situation (though it never arose in our experiments) the leader saves a copy of the model's weight dictionary every training step that can be reloaded into the workers and leader so training may resume if it were to crash.

6 BENCHMARK EVALUATIONS

To determine if temporal link prediction is a viable method for lateral movement detection, it is necessary to show that Euler is a viable temporal link predictor. Thus, before transitioning to larger cyber security datasets, we evaluate the effectiveness of Euler on link prediction in general. In this section, we briefly present the results of our experiments on smaller, general purpose temporal graph datasets from our prior work. Details of these datasets are reported in Table 1. For a more detailed explanation of the experimental setup, we direct the reader to the conference version of this work [44].

Prior works VGRNN [35] and **Evolving GCN (EGCN)** [63] both assert that simply embedding graph snapshots with a GNN and running these embeddings through an RNN, as was done by References [68, 84], does not adequately capture the shifting distribution of nodes in a dynamic network. Reference [63] demonstrates that this is the case for inferring complete graph structure several timesteps in the future, but Reference [35] does not evaluate their model against the very model they so thoroughly disregard. To remedy this, we present a comparison between several existing temporal autoencoder models and a simple stacked GCN [47] and GRU [21] following the Euler framework. We select these two layers because of their lack of parameters—they are the most generalized of the models in their respective domains [22, 69]. For all experiments, we set all models evaluated to have the same dimensionality and number of layers. As each model evaluated uses a single weight matrix per encoding layer, and three per RNN layer (apart from DynAE, which does not use an RNN, and DynRNN, which does not use an encoding layer), this caps the parameter count to a uniform level across all tests. For reference, a brief summary of the other methods tested is presented in Table 2.

6.1 Results

As was done by Reference [35], we conducted three different benchmarking tests to compare Euler to other temporal link prediction methods: inductive dynamic link detection, transductive dynamic link prediction, and transductive dynamic new link prediction. Link detection and link

Table 2. Prior Works Evaluated

Model		GNN	RNN	Description
(SI-)VGRNN	[35]	✓	✓	At the time of writing, the most effective temporal link prediction model in the literature. A GCN stacked upon a GC-LSTM [19].
E-GCN	[63]	✓	✓	A powerful model that uses an RNN to update the parameters of a GNN at each timestep. Especially good at link prediction far into the future, but untested in short-term predictions.
DynAERNN		✗	✓	A family of models that pass adjacency vectors into a multilayer perceptron and/or an RNN to capture graph dynamics using traditional deep learning techniques.
DynRNN	[65]	✗	✓	
DynAE		✗	✗	
VGAE	[48]	✓	✗	A simple two-layer variational graph autoencoder. This model does not consider time at all. Instead, it views every interaction as one large graph.

Table 3. Comparison of Euler to Related Work on Dynamic Link Detection

Metrics	**Methods**	Enron	COLAB	Facebook
AUC	VGAE	88.26 ± 1.33	70.49 ± 6.46	80.37 ± 0.12
	DynAE	84.06 ± 3.30	66.83 ± 2.62	60.71 ± 1.05
	DynRNN	77.74 ± 5.31	68.01 ± 5.50	69.77 ± 2.01
	DynAERNN	91.71 ± 0.94	77.38 ± 3.84	81.71 ± 1.51
	EGCN-O	93.07 ± 0.77	**90.77 ± 0.39**	86.91 ± 0.51
	EGCN-H	92.29 ± 0.66	87.47 ± 0.91	85.95 ± 0.95
	VGRNN	94.41 ± 0.73	88.67 ± 1.57	88.00 ± 0.57
	SI-VGRNN	95.03 ± 1.07	89.15 ± 1.31	88.12 ± 0.83
	Euler	**97.34 ± 0.41**	**91.89 ± 0.76**	**92.20 ± 0.56**
AP	VGAE	89.95 ± 1.45	73.08 ± 5.70	79.80 ± 0.22
	DynAE	86.30 ± 2.43	67.92 ± 2.43	60.83 ± 0.94
	DynRNN	81.85 ± 4.44	73.12 ± 3.15	70.63 ± 1.75
	DynAERNN	93.16 ± 0.88	83.02 ± 2.59	83.36 ± 1.83
	EGCN-O	92.56 ± 0.99	**91.41 ± 0.33**	84.88 ± 0.52
	EGCN-H	92.56 ± 0.72	88.00 ± 0.85	82.56 ± 0.91
	VGRNN	95.17 ± 0.41	89.74 ± 1.31	87.32 ± 0.60
	SI-VGRNN	**96.31 ± 0.72**	89.90 ± 1.06	87.69 ± 0.92
	Euler	**97.06 ± 0.48**	**92.85 ± 0.88**	**91.74 ± 0.71**

prediction are implemented as described in Section 5.3. On link detection tests the objective function is Equation (3) with $n = 0$. On (new) link prediction tests, $n = 1$. New link prediction is an extra evaluation of predictive models. This test is identical to link prediction, but the set of true positives only includes edges that were not in the snapshot immediately before.

Dynamic Link Detection

As shown in Table 3, the simplistic Euler model outperforms the more modern ones in almost every test. In tests where it does not outperform the state-of-the-art methods, it is equivalent, despite its simplicity. Compared to the static VGAE, which does not consider time at all, the benefit

Table 4. Comparison of Euler to Related Work on Dynamic Link Prediction

Metrics	Methods	Enron	colab	Facebook
AUC	DynAE	74.22 ± 0.74	63.14 ± 1.30	56.06 ± 0.29
	DynRNN	86.41 ± 1.36	75.7 ± 1.09	73.18 ± 0.60
	DynAERNN	87.43 ± 1.19	76.06 ± 1.08	76.02 ± 0.88
	EGCN-O	84.28 ± 0.87	78.63 ± 2.14	77.31 ± 0.58
	EGCN-H	88.29 ± 0.87	80.80 ± 0.95	75.88 ± 0.32
	VGRNN	93.10 ± 0.57	**85.95 ± 0.49**	89.47 ± 0.37
	SI-VGRNN	**93.93 ± 1.03**	85.45 ± 0.91	**90.94 ± 0.37**
	Euler	**93.15 ± 0.42**	**86.54 ± 0.20**	**90.88 ± 0.12**
AP	DynAE	76.00 ± 0.77	64.02 ± 1.08	56.04 ± 0.37
	DynRNN	85.61 ± 1.46	78.95 ± 1.55	75.88 ± 0.42
	DynAERNN	89.37 ± 1.17	81.84 ± 0.89	78.55 ± 0.73
	EGCN-O	86.55 ± 1.57	81.43 ± 1.69	76.13 ± 0.52
	EGCN-H	89.33 ± 1.25	83.87 ± 0.83	74.34 ± 0.53
	VGRNN	93.29 ± 0.69	87.77 ± 0.79	89.04 ± 0.33
	SI-VGRNN	**94.44 ± 0.85**	**88.36 ± 0.73**	**90.19 ± 0.27**
	Euler	**94.10 ± 0.32**	**89.03 ± 0.08**	**89.98 ± 0.19**

Table 5. Comparison of Euler to Related Work on Dynamic New Link Prediction

Metrics	Methods	Enron	colab	Facebook
AUC	DynAE	66.10 ± 0.71	58.14 ± 1.16	54.62 ± 0.22
	DynRNN	83.20 ± 1.01	71.71± 0.73	73.32 ± 0.60
	DynAERNN	83.77 ± 1.65	71.99 ± 1.04	76.35 ± 0.50
	EGCN-O	84.42 ± 0.82	79.06 ± 1.60	75.95 ± 1.15
	EGCN-H	87.00 ± 0.85	78.47 ± 1.27	74.85 ± 0.98
	VGRNN	**88.43 ± 0.75**	77.09 ± 0.23	87.20 ± 0.43
	SI-VGRNN	**88.60 ± 0.95**	**77.95 ± 0.41**	87.74 ± 0.53
	Euler	87.92 ± 0.64	**78.39 ± 0.68**	**89.02 ± 0.09**
AP	DynAE	66.50 ± 1.12	58.82 ± 1.06	54.57 ± 0.20
	DynRNN	80.96 ± 1.37	75.34 ± 0.67	75.52 ± 0.50
	DynAERNN	85.16 ± 1.04	77.68 ± 0.66	78.70 ± 0.44
	EGCN-O	86.92 ± 0.39	81.36 ± 0.85	73.66 ± 1.25
	EGCN-H	86.46 ± 1.42	79.11 ± 2.26	73.43 ± 1.38
	VGRNN	87.57 ± 0.57	79.63 ± 0.94	86.30 ± 0.29
	SI-VGRNN	**87.88 ± 0.84**	**81.26 ± 0.38**	**86.72 ± 0.54**
	Euler	**88.49 ± 0.55**	**81.34 ± 0.62**	**87.54 ± 0.11**

of the additional RNN layer is clear. We further observe the benefit of graph neural networks over MLPs when Euler and the state-of-the-art methods are compared to the DynGraph2Vec methods. However, the experiments do not support claims that much is gained by the complex models beyond what is afforded simply by using a GNN and RNN. Both highly engineered models, the EGCN and VGRNN variants, do not perform significantly better than the simplistic stacked GCN on a GRU, and in some cases perform worse.

Significantly, the dataset where Euler performed better than prior works with $p < 0.05$ was the Facebook dataset. This dataset contains the most nodes and edges and has the fewest snapshots in the training set. Despite these difficulties, our simple Euler model achieves a 4% improvement over prior work in both AUC and AP, signifying its ability to learn very complex spatio-temporal patterns even on larger datasets. We observe that the model is generalized enough not to become overfit on the smallest datasets, but not so simple it cannot handle larger ones. This supports our claim that this model design, despite its simplicity, is highly precise.

Dynamic (New) Link Prediction

In these cases where temporal data are more significant, the results are less clear. As shown in Tables 4 and 5, between our method and (SI-)VGRNN models, the results are almost all within each methods' margin of error. We note that on the dynamic new link prediction test for Enron10, though our method's observed mean AUC and AP were lower than both VGRNN methods, a t-test showed that this disparity was not statistically significant. We can conclude that neither method is significantly better than the other. Furthermore, we observe that on the Facebook dataset, which has roughly the same edge density as Enron, but 3.5× as many nodes, Euler performs significantly better than other methods in new link prediction. As variance and complexity increases in the data, Euler adapts better than the other methods while retaining precision on simpler data without becoming overfit.

In both tests, models that process graph embeddings using an RNN were significantly better than DynAE, which does not. This component enables temporal attributes of the data to be carried over from previous timesteps. If a new edge has been seen in the distant past, or a pattern that indicates a new edge is likely to appear has previously been observed, then this history is carried over by the RNN into future embeddings. From this, we can again infer that the benefit derived from the (SI-)VGRNN models has more to do with the components of those models, which are also GCNs connected with an RNN. The espoused benefit of the topological embedders ingesting temporal information does not appear to be as great as simply using those components; by removing the GNNs' reliance on temporal information, our models can embed at least the same quality of information in a more efficient manner.

With these data, it is clear that the simplicity of models following the Euler framework is not a hindrance, and in many cases is actually advantageous. The purpose of Euler is chiefly to improve efficiency and scalability, so the fact that it is only a small improvement, or about equal to state-of-the-art models, is adequate for our purposes. The real benefit of building models within the Euler framework is their ability to scale. With larger datasets, this advantage is more evident.

7 THREAT MODEL & ASSUMPTIONS

As we have shown, the Euler framework is capable of outperforming prior works on temporal link prediction generally. The next step is to show that temporal link prediction is an effective method of lateral movement detection. In the following sections, we will show that Euler can efficiently detect anomalies indicative of lateral movement under the minimal constraints defined in this section.

Like many other anomaly detection models [15, 16, 42, 54], we make the assumption that a large sample of benign data is available from which to learn a baseline for normal behavior offline. This means the framework has certain limitations: If there is not enough normal behavior to learn from, then the false positive rate may be higher. Additionally, in a real-world setting, if there was malicious activity in what is thought to be the clean training data, then the model could learn that certain patterns in connections are normal when they are not. Unique to temporal anomaly detection is the difficulty in knowing how fine-grained the graph snapshots ought to be. Certain events may be anomalous simply because they happened at a certain time of day [31], which would evade our system if snapshots spanned a full day of activity. Finally, we note that, depending on how Euler is implemented, there may be some delay in detection time. With a predictive model, new edges can be scored as they appear, which takes a negligible amount of time. However, if Euler uses the detection mode, then it must wait for an entire snapshot to be generated before it may score the events that occurred. However, experiments showed smaller snapshots resulted in better performance for the model; the delay incurred waiting for a snapshot would be between 6 and 30 minutes using our hyperparameters. As lateral movement is a "low-and-slow" process, this would likely not be a problem.

To alleviate some of these difficulties, we focus on datasets with large bases of training data from which to learn these patterns. In a real-world setting, there are countless commercial and academic systems for network security monitoring that could be employed to build a large database for training [1, 23, 40]. This assumes that the logged data does not contain anomalous activity, but this can be verified using a signature-based system during its collection, or other expert analysis. Our focus on lateral movement detection means only internal communications are used for training, further reducing the risk of anomalous behavior in the training set (e.g., users visiting websites for the first time, novel events that are "anomalous" but not malicious). This means that our models will be unable to detect the initial compromise but are trained to find irregularities in inter-host communications, a signature indicative of lateral movement [20].

To capture these suspect communications, at a minimum, we need the information about the sources and destinations of traffic sources within an organization. In our experiments, we used authentication records and data flow objects as the means of tracking these things. Both data sources largely represent the same thing: a user or system attempting to use a service on a remote device. As a threat-actor attempts to traverse a large network of computers, they will endeavor to do so slowly and sneakily, so it is unlikely they would use a service that is not monitored by these tracking services in the datasets [71]. However, we concede that if there are holes in a systems auditing that ignore certain services, then an attacker could utilize them for lateral movement and

Table 6. LANL Dataset Metadata

Nodes	17,685
Events	45,871,390
Anomalous Edges	518
Duration (Days)	58

evade our (and other similar) systems' alerts. Thus, we assume that all relevant remote tools and services are monitored by a logging system. Beyond unique identifiers for entities within a system to track their communications, Euler needs very little. As a temporal graph analyzer, timestamps are required or may be inferred from the ordering of the logging events. Features for each entity, while they may aid in false positive reduction, are not strictly needed. In this work, we use only features that can be inferred from the unique names of system entities (whether they are users or computers, if they are administrators, etc.). We leave additional research in feature extraction for the nodes as a topic of future work.

8 EXPERIMENTS ON LANL

In the previous section, all datasets tested were rather small. It is not until a real-world application of the Euler framework is tested that the true performance improvements are evident.

To demonstrate the impressive speedup achieved by this framework when compared to related work, we evaluate several Euler models on the LANL 2015 Comprehensive Multi-Source Cyber Security Events dataset [41]. The dataset consists of 57 days of log files from five different sources within the Los Alamos National Laboratory's internal corporate network as it underwent both normal activity and a red team campaign. Specific details about this dataset are reported in Table 6. The edge count in the table represents the number of weighted edges; multiple events between the same entities in the same time period may be compressed into a single weighted edge. Because events from the authentication logs have been labeled as normal or anomalous, this dataset has been widely used for cyber security research [13, 16, 37, 81]. The labels make it especially apt for lateral movement detection research. When an APT-level threat is attempting to traverse a system, one possible warning sign will be authentications that should not normally occur, a sign indicative of lateral movement on a network level [38, 42, 54].

In this section, all distributed models were implemented with four worker nodes unless otherwise specified. All experiments are run on a server with two Intel Xeon E5-2683 v3 (2.00 GHz) CPUs, each of which has 14 cores with 28 threads, and 512 GB of memory [5].

We will first present the utility of models following the Euler framework as an anomaly-based intrusion detection system on the LANL dataset, then an analysis of the immense scalability afforded by splitting models in this way.

8.1 Graph Construction

We construct a weighted, directed graph from the authentication logs by mapping which entities authenticate with one another. As nodes, we use the entities denoted source and destination computers in the LANL documentation. For all authentications that occur from time t to $t + \delta$, an edge is created between the source computer and destination computer. If an edge already exists, then a tally keeping track of the number of authentications between the two machines is updated. Experiments have shown that the most effective method to normalize these tallies into usable edge weights is to take the logistic sigmoid of the edges' standardized values. Mathematically, it can be represented as

$$W((u,v) \in \mathcal{E}) = \sigma\left(\frac{C(u,v) - \mu_{\mathcal{E}}}{\Sigma_{\mathcal{E}}}\right), \tag{9}$$

where $\sigma(\cdot)$ represents the sigmoid function, $C(u, v)$ represents the frequency of an authentication between u and v in the time window, and $\mu_{\mathcal{E}}$ and $\Sigma_{\mathcal{E}}$ represent the mean and standard deviation of all edge frequencies in the time window. In this way, edges that occur very infrequently are given lower weight during training to appear less "normal," and edges that occur with high frequency, such as edges from computers to domain controllers or ticket-granting servers, have high weight and appear routine.

The LANL dataset has no node features by default, however, some information can be gleaned from the naming convention used in the log files. Entities have unique, anonymized identifiers that start with either a `U` or a `C` denoting users and computers, respectively. There are also nodes with non-anonymized names that have important roles in the system such as `TGT`, the Kerberos key distribution center, `DC`, the domain controller, and so on. To leverage this additional data, we concatenate a 1-hot vector denoting user, computer, or special administrative machine to each node's one-hot ID vector.

For quicker file scanning, and data-loading times, the full 69 GB `auth.txt` file is split into chunks, which each hold 10,000 seconds (approximately 3 hours) of logs. Worker machines are issued instructions to read in certain ranges of the log files and build the temporal graphs. Workers accomplish this by spawning several child processes to load multiple snapshots in parallel. The associated edge lists and edge weight lists from each child process are combined to form the final `TGraph` object, which holds a list of all edge lists, edge weights, node features, and tensor masks to take partitions of each edge list for training.

For all experiments, the training set consists of all snapshots that occur before the first anomalous edge appears in the authentication logs. This allows models to learn what normal activity looks like. From this set, we remove the final 5% of snapshots for tuning the classifier and mask 5% of edges from each snapshot for validation.

8.2 Experimental Setup

We test three encoders in conjunction with two recurrent neural networks as well as models with no recurrent layer to measure how much value temporal data adds to the overall embeddings. The encoder models are GCN [47], GAT [76], and GraphSAGE [36]. The recurrent models are GRU [21] and LSTM [39]. The models are trained in the same manner as the link detection and link prediction models in Section 6. However, experiments showed that once a local optimum was found and validation scores ceased improving, it rarely improved after further iterations. As such, early stopping occurs after only 10 epochs of no improvement.

Experiments showed for every model that using smaller time windows always lead to better results. As such, we only present the output of tests on temporal graphs with time window $\delta =$ 1,800 seconds (30 minutes).

The GAT encoder uses three attention heads, which was found to be optimal via hyperparameter tuning. The SAGE encoder uses maxpooling as its aggregation function, as this was found to be optimal in their paper, and this makes it capable of discerning between certain graphs GCN cannot [83].

Unfortunately, many other works that experiment with the LANL dataset either do not use the dataset in full, as is the case with References [37, 81], or conduct tests on portions of the data other than purely the authentication logs, as was done by Bai et al. [13], so it would not be fair or meaningful to compare our results to theirs. Bowman et al. [16] does use the full authentication log as its dataset, however, it trains on a larger set of data, using all days that contained no anomalous activity as the training set, rather than just the days before the attack campaign. We include their results, nonetheless. We also include the TPR and FPR of a rules-based **Unknown Authentication (UA)** model reported by Reference [16]. This rule simply marks any edge that did not exist in the

Table 7. Performance of Euler Models on the LANL Dataset when $\delta = 0.5$

Dynamic Link Detection

Encoder	RNN	AUC	AP	TPR	FPR	P
GCN	GRU	0.9912	**0.0523**	86.10	0.5698	0.0054
	LSTM	0.9913	0.0169	89.65	0.5723	0.0056
	None	**0.9916**	0.0116	88.57	**0.4798**	**0.0066**
SAGE	GRU	0.9872	0.0307	84.71	0.6874	0.0044
	LSTM	0.9887	0.0389	83.55	0.6591	0.0045
	None	0.8652	0.0052	79.58	24.5669	0.0001
GAT	GRU	0.9094	0.0076	85.21	21.533	0.0001
	LSTM	0.8713	0.0022	96.83	19.873	0.0002
	None	0.9867	0.0079	**99.88**	23.174	0.0002
VGRNN		0.9315	0.0000	59.69	4.938	0.0000
UA		–	–	72.00	4.400	0.0010
GL-LV [16]		–	–	67.00	1.200	0.0034
GL-GV [16]		–	–	85.00	0.900	0.0051

Dynamic Link Prediction

Encoder	RNN	AUC	AP	TPR	FPR	P
GCN	GRU	**0.9906**	0.0155	85.49	0.6088	0.0050
	LSTM	0.9885	0.0166	78.91	0.5987	0.0047
	None	0.9902	0.0092	86.42	0.5425	**0.0057**
SAGE	GRU	0.9847	0.0200	86.30	1.6542	0.0019
	LSTM	0.9865	**0.0228**	85.29	0.8037	0.0038
	None	0.9284	0.0020	86.23	16.525	0.0002
GAT	GRU	0.8826	0.0020	87.82	21.971	0.0001
	LSTM	0.8383	0.0002	83.42	29.297	0.0001
	None	0.9352	0.0079	**88.83**	20.093	0.0002
VGRNN		0.9503	0.0004	70.00	**0.280**	0.0004

training data as anomalous. This system acts as a baseline to which we compare all other models. Models that outperform UA's FPR show they are capable of anticipating probable edges despite no prior knowledge of them; models that outperform its TPR show an understanding of edge context within the greater neighborhood, both temporal and spatial. They can detect edges that, though previously observed, in a new context, are malicious.

By default, VGRNN operates on full adjacency matrices, however, we modified it to use sparse edge lists for our experiments. This way it was able to scale to the large size of the LANL dataset. Unfortunately, the E-GCN and DynGraph2Vec models could not scale to the LANL dataset. DynGraph2Vec relies on dense adjacency matrices, and the size of the 1-hot vectors used as inputs was too large for E-GCN to process. As a result, our hardware was unable to fully evaluate these methods, and their results are not compared to those of Euler.

All models evaluated use 32-dimensional hidden layers, and 16-dimensional embeddings. All Euler models use a tanh activation function between the encoder and the recurrent layers and an edge dropout layer before the GNNs. They all determine the classification threshold according to Equation (8) with $\lambda = 0.6$, except the GraphSAGE models. For this encoder, experiments showed $\lambda = 0.5$ was more appropriate. All reported results are average scores from five independent tests on link detection and link prediction.

8.3 Anomalous Edge Detection

It is difficult to properly evaluate methods for classifying imbalanced data, especially anomaly detection, where small false positive rates are so critical. For this reason, in addition to the raw true positive and false positive rates, we report **precision (P), area under the curve (AUC)**, and **average precision (AP)**. This latter method is recommended for anomaly detection by Reference [25] as especially adept for imbalanced datasets. The AUC and AP metrics evaluate the overall quality of scores given to edges, as opposed to the quality of classification, and provide better measurements of the model if the anomalous score threshold was to be changed. The precision metric provides further context to the quality of classification at the specific threshold. The average results of five experiments are shown in Table 7.

As a baseline, consider the TPR of the UA rules-based approach. This implies that 28% of anomalous connections are those that have occurred before in the network. This supports our claim that temporal information about the context of connections is just as important as the entities that are authenticating. This system is an excellent baseline model to compare to, as any model that has a higher TPR than UA must be using a more advanced metric than simply memorizing every legitimate connection observed in normal activity. If a model has a TPR above

72% with an FPR lower than 4.4%, then it must be leveraging topological or temporal context judge the validity of connections.

The results show that the GCN is the most effective encoder for link detection, and SAGE is most effective for link prediction; this supports our claim and those of Halilaj et al. [69] that more generalized models are more effective. The GAT models, which have 3× as many parameters as GCN and SAGE performed quite poorly both in quality of scores and quality of classification.

Also worth noting is the way using an RNN affects the output. Surprisingly, the best AUC in the link detection tests were from a GCN with no temporal encoder. However, this metric is not a good indicator of model quality on datasets with imbalance as extreme as LANL. The dramatically higher AP score of all models that use RNNs suggests temporal data strongly affects FPR and cannot be ignored. Similarly, the models without an RNN have high precision on the GCN models. However, again, the AP scores would indicate that while at this specific threshold omitting the RNN is beneficial, over all thresholds, models that take time into account perform better.

In the more realistic transductive link prediction tests, though the difference between the two RNNs tested is small in every case, the benefit they add is unquestionable. The best-performing encoder, GraphSAGE, enjoyed a 10× improvement in AP when used in conjunction with any RNN. The next best-performing encoder, GCN, achieved a 1.6× improvement in AP. This is evidence that temporal information carries important context for the topological state of the network, particularly for filtering false positives. Where one authentication may appear anomalous in isolation, when viewed in the context of previous authentications, it can be correctly identified as benign.

The GL-LV and GL-GV methods do not consider time at all; the network is viewed as a static graph. Here, we again see the benefit of using a sequence encoder in conjunction with a pure topological embedder. The best Euler methods outperform their random walk-based approach in terms of both TPR and FPR. Also worth noting is that because our model uses temporal graphs, the alerts from Euler-based models come with a timestamp, making them more informative and valuable in a real-world scenario. The prior work ranks any duplicate edges, regardless of their temporal context, as equally anomalous.

Like Euler, VGRNN combines a sequence encoder with a temporal one. They claim that by using temporal information as input during the topological encoding phase, complex temporal dependencies are better encoded than without it. However, this method performs no better than the purely statistical UA method. Even still, the false positive rate is excellent in the predictive test, outperforming every Euler model. It is worth noting, however, the quality of likelihood scores is very poor with this method, which implies that had the threshold for the Euler models been set lower, their FPR would be lower with equivalent TPRs. This is readily apparent in the GCN-based models where the FPR is only a few 10ths of a percent larger than the dynamic VGRNN, but their TPRs are almost 10% higher.

Finally, we must concede that while the Euler models do outperform prior works, their FPRs are still too high to be useful as an intrusion detection system on their own. Some of this can be attributed to the dataset itself; labeled anomalous events are very coarse-grained. There are likely many events the compromised entities engaged in that should be considered anomalous, and may have even been detected by our models, but that are treated as false positives due to the lack of fine-grained label information. Indeed, the red log only tracks “compromise events” and not the further malicious activity that ensues [41].

However, even if this is not fully the case, this method has great potential as a filtering device for further analysis tools. The low cost of processing time, which we will demonstrate later on, makes this an efficient way to minimize the number of interactions that need to be analyzed by a signature-based technique, for example. As we will show in Section 9, this approach can be further improved as a step in a longer pipeline. However, even without additional analysis, we have demonstrated

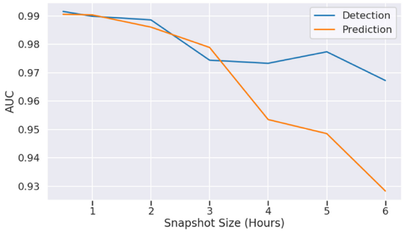


Fig. 5. Change in AUC score as δ increases for link prediction and detection with GCN+GRU models. Scores are the average of five tests on the LANL dataset.

that compared to all other anomaly-based works that analyze this dataset, Euler models are the most effective.

8.4 Parameter Analysis

Time window size, the hyperparameter δ, has significant tradeoffs associated with it. As this value decreases, the number of edges increases, requiring more processing time. As this value increases, more repeat edges are condensed into single, weighted edges; additionally, there will be fewer discrete timesteps for the recurrent networks to process, all contributing to faster processing time at the expense of less precise data. With more edges, and more temporal granularity, models are more capable of learning useful patterns across time. Figure 5 illustrates this phenomenon.

As is evident from the figure, small window sizes are more optimal for quality of scores; however, due to the longer processing time required, and the relatively small performance improvements as window size decreases after a certain point, it may be adequate to leave the snapshot duration a little higher than is optimal for faster training and evaluation. Additionally, we observe that changes in δ seem to affect the link prediction model at a higher rate than the link detection one. We suspect this is caused by the model's inability to predict short-term temporal patterns like the one described in Section 4 as δ grows, and the predictive model is especially apt to detect this type of anomaly.

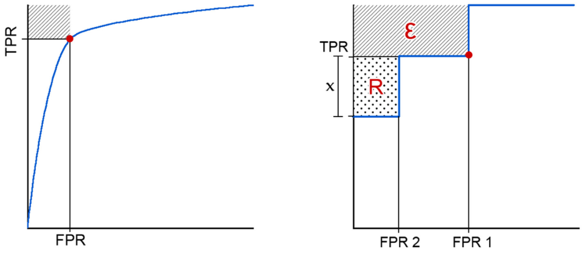
Nonetheless, in both cases, more granular graphs lead to more informative edge scores. We speculate that at some point, having time slices too granular would have diminishing returns, as graphs will have too few edges to be useful. But due to the severe training time as δ decreases, we have never managed to reach this point. We leave finding this boundary as an area for future work.

8.5 Interpreting the Results

Table 7 shows the results we found when testing several Euler models, the VGRNN model [35], and the results reported by Bowman et al. [16] on the LANL dataset. In Reference [44], we concluded that our method outperformed the others. However, as anomalous link detection is so heavily class imbalanced, it was at times difficult to properly interpret these results.

A low false positive rate is of paramount importance, but this is very contingent on where the cutoff threshold is set. But even assuming this threshold is set optimally, how do we factor in the true positive rate? For our experiments, we were fortunate that Euler models outperformed prior works in both true positive and false positive rates in many cases but consider VGRNN's performance on link prediction. It has the best FPR of any model, but we argue that it still is not the best model. Looking at any one metric is deceiving. The AUC scores, which give us a glimpse of how moving the threshold would affect things, are excellent for this.

One way we attempted to quantify this was to generate a lower bound on how much changing the threshold would affect the TPR. Figure 6(a) demonstrates how to geometrically find the error



(a) Observed error

(b) Worst-case residual error

Fig. 6. Calculating the upper bound of error caused by increasing or decreasing the classification threshold. The observed error, ε, in the AUC is the shaded region in Figure 6(a). In the worst case, the residual error, R, takes up the dotted region in Figure 6(b). This value is bound by $AUC + \varepsilon + R = 1$, so the TPR can decrease by at most x.

guaranteed from the single known TPR and FPR metric. The AUC plus this known amount of error, and the residual error, must sum to 1. In the worst case, the entire residual error would be to the left side of the curve. Figure 6(b) places a lower bound on the TPR if the threshold were changed to match the FPR attained by another model. The full lower bound equation is

$$\hat{t} \geq t - \frac{1 - A - (1 - t)f}{\hat{f}}, \tag{10}$$

where $\hat{f}$ is the desired FPR, A is the AUC score, and t and f are the observed TPR and FPR scores.

Unfortunately, for our tests, this approach did not generate useful results. In most cases, assuming the residual error falls entirely on the left of the known point results in a lower bound less than zero. However, we believe this approach to model comparison has value and present the equation for future use.

It is challenging to properly compare models, but we were fortunate that our results were so unambiguous. Though the TPR and FPR can provide more concrete descriptions of how these models may perform in the real world, from a more analytic standpoint, the regression metrics carry more information. These metrics tell us *if* a good threshold exists, rather than if one was found. As models' classification thresholds can always be tuned, metrics that measure the actual distribution of threat scores are far more telling.

8.6 Challenges

Evaluating these temporal link prediction models came with several challenges: setting up the experiment in a fair manner, accounting for the differences between models, and the nature of the data itself. In this subsection, we will describe these challenges and by what means we overcame them.

Validation and Tuning

When tuning a model's hyperparameters, it is important to only measure their effect on a validation set. Tuning them with respect to the test data is a form of p-score hacking [78] and generally in poor form. However, for our experiments, we did not have an especially representative validation set. As our model was meant to be entirely unsupervised, in lieu of holding out some of the labeled data for validation, we used a held-out set of unlabeled, benign data. In doing so, the validation did not reflect the eventual class imbalance that would be present in the test data. Nor did it reflect the actual nature of the test data.

The test data contained anomalous edges that affected the overall graph structure. As a result, these edges would be considered in the GNN during node embedding. The negative edges in the validation set, however, were not. This raises questions about the value of this method of validation. Because of this, there was a large discrepancy between the validation results and the test results. Nonetheless, it was all we could do. This may explain why more complex models such as the GAT had such poor performance. It was very difficult to properly tune their hyperparameters on a validation task that was dissimilar to the final test objective.

Fair Comparison

Another difficulty we faced was fair comparison of Euler to the other models. For example, the models from Reference [16] use entirely different training parameters and experimental setups. These approaches view the underlying graph structure as static. This entails that every interaction between entities through time is analyzed at once. This makes it so their model is considering fewer edges overall in its evaluation. Because Euler considers a temporal graph, duplicate edges count more than once; false positives can occur more than once while the number of true positives remains the same. Even if the two models classified edges in exactly the same manner, just the fact that there are more edges in the temporal graph makes it harder to have a false positive rate lower than the prior works.

Additionally, their models were trained on more data than ours was. Every day that does not contain an attack was used in the training set, while only attack days were used for evaluation. Nevertheless, even with these setbacks, Euler outperforms these approaches. Though the comparison is not fair, it is biased against our model, not theirs. We believe this strengthens our argument for Euler. We feel that unfair comparison is acceptable so long as it works in the competing methods' favor.

Labeling the Data

The LANL dataset came with certain events that were labeled as anomalous. However, they give very little context to what these anomalous events were. It may be the case that the labeled authorizations only represent initial compromises or successful compromises. There could be many other unlabeled anomalous authorization events that follow from these labeled events.

Furthermore, as the dataset is from a real-world system, there could still be more anomalous events just from the users doing unusual, potentially malicious things unrelated to the documented campaign. As these events are unlabeled, we cannot know for sure without expert assistance. However, as all methods were evaluated using the same labels, we still feel this is a fair assessment of each of their abilities.

We did run tests under the assumption that compromised hosts compromise all other hosts they authenticate with in the future, but the results were inconclusive. The first compromise occurs only 2 days into the 57 days of logs. Using this new labeling system, very quickly, almost every authentication is marked as anomalous. This is obviously not realistic or adding any value. More likely, after a day or so the campaign ended, and the compromised hosts were reset, but again, we did not have enough information to make such an assessment. We did have fair results by resetting the list of compromised hosts every 24 hours, but this makes assumptions that are not backed by the data. Ultimately, we just used the labels they provided and accepted the lower evaluation scores. So long as we evaluate every model with the same labels, the hindrance is distributed equally.

9 EXPERIMENTS ON OPTC

In this section, we apply the lessons learned from the initial experiments on the LANL and benchmarking datasets to a new experiment. Additionally, we apply a new augmentation to Euler to further extract value from its embeddings.

Table 8. OpTC Dataset Metadata

Nodes	1,114
Events	7,773,514
Anomalous Edges	21,872
Duration (Days)	7

9.1 The OpTC Dataset

The DARPA **Operationally Transparent Cyber Data Release (OpTC)** dataset consists of host logs from 1,000 machines interacting with each other and external hosts over a one-week period. These logs span four days of benign activity and three days of differing attack campaigns. In total, the 1.1 TB dataset contains 17.4 billion events [12] stored in a modified CAR format [59]. As we are primarily interested in lateral movement detection over the network, we only use a small subset of these events: all `FLOW` objects undertaking a `START` action. In this way, we create a temporal graph of hosts on the network initiating communication with other machines. The decision to base the graph on successful `FLOW-START` objects is two-fold: first, because it is a more compressed way to represent the entirety of several `FLOW-MESSAGE` objects, while retaining the same data; second, because this is the most direct way in which inter-host communication is represented in the OpTC dataset. While `PROCESS-OPEN` events for remote processes also capture this, they will also always produce a `FLOW-START` event. Thus, it is the most efficient way to capture all communication between hosts and where lateral movement will necessarily manifest [51, 71]. The full metadata of this dataset is shown in Table 8.

To convert from log files to graphs, we preprocess the data to extract all `FLOW-START` objects and parse out the source and destination IPs as well as the timestamp. This dataset is not labeled by default. Rather, the curators provide a red log written by humans describing the malware campaign and any events of note. As a result, when labeling the data, more context is available. So, unlike in the original LANL experiments, there is no ambiguity: If an event is labeled as anomalous, then it is because of lateral movement.

The data are labeled according to the following criteria: If the source IP was involved in a red team event that happened that day, and if that red team event occurred before or during the time of the logged event, then it is marked as anomalous. This decision is motivated by the assumption that when a host is compromised, it remains compromised for the duration of the red team campaign—and these campaigns each last a single day. Rather than guessing if the compromised state of an entity should be propagated, as we attempted to in Section 8.6, the red log explicitly states if an attempt at lateral movement was successful. So, only hosts that have been successfully compromised produce anomalous edges. Each campaign only lasts a single day, so at the end of each day, the state of each entity in the network returns to normal.

When the event logs have been parsed and labeled, the temporal graph is constructed in parallel by each worker. Each worker loads a disjoint range of the labeled events, partitions them into snapshots, and constructs edge lists from the observed interactions between IPs and their labels. Any self-loops in the data are ignored. Though there is much data describing each node, in our experiments, nodes have no features, so one-hot encodings of the node IDs (the identity matrix of size $|\mathcal{V}|$) are used as input features to the GNNs.

9.2 Model Augmentations

As Ouyang et al. [62] point out, the objectives of networks that embed nodes for high-fidelity reconstruction and those that identify anomalies are often different. There may be irrelevant information encoded in the $\mathbf{Z}$ vectors necessary for reconstruction, but not necessarily required for anomaly detection.

Therefore, we also implement their approach to anomaly detection, which we denote Euler-SM. After embeddings for reconstruction have been generated, they are used as the inputs for a final anomaly detection layer $a(\cdot)$. This layer aggregates a sample of each node's neighbors' embeddings and applies a final transformation upon $\mathbf{Z}_t$ into a $|\mathcal{V}|$ dimensional vector that acts as a probability distribution that an edge exists between any two nodes at time $t + n$. This final layer is then defined as

$$\begin{aligned} P[\mathbf{A}_{t+n} = 1 \mid \mathbf{H}_t] &= a(\mathbf{H}_t) \\ a(\mathbf{H}_t) &= \text{Softmax}(\mathbf{W}\mathbf{H}_t) \\ \mathbf{H}_t &= \frac{1}{s}\mathrm{S}_t\mathbf{Z}_t \end{aligned}, \tag{11}$$

where $\mathbf{W}$ is a trainable parameter, and S_t is the identity matrix plus s randomly sampled elements of $\mathbf{A}_t$ per row. In practice, this is accomplished with message-passing [30], not matrix multiplication. As was done by Reference [62], to create the S_t matrix, we sample an equal number of neighbors with replacement for each node during training and use the full, normalized adjacency matrix during evaluation. The number of samples drawn, s, is a user-defined hyperparameter. Experiments showed that $s = 5$ works well.

As distributing the work of graph processing is still of paramount importance, this is implemented as another distributed model. The anomaly detection layer is replicated across all worker machines after they have completed training and generated the $\mathbf{Z}$ embeddings. The training cycle repeats with the $\mathbf{Z}$ embeddings being used as input to optimize the $\mathbf{W}$ parameter in the $a(\cdot)$ function on each worker.

This technique extracts even more data from the latent $\mathbf{Z}$ embeddings. Unfortunately, due to the necessity of the softmax function, this decoding strategy does not scale well with additional hosts—though it does scale with large time periods and large edge counts—nor can it accommodate networks that add nodes over time, as the size of the final $a(\mathbf{H}_t)$ vector must be static. However, we will demonstrate that there are situations where this decoding strategy is extremely beneficial.

9.3 Experimental Setup

In this section, we compare the effectiveness of several link prediction models on the OpTC dataset. Models are trained on the first four benign days to learn the pattern of normal interactions within the network; all snapshots after the first four baseline days are considered the test set. Because each day in the OpTC test set represents a different attack campaign that is reset the following day, the final hidden state of the RNNs after they process the training set is saved to be used as the initial hidden state for the beginning of each campaign. We evaluate models using several different sizes of snapshots and report test results of models that had the best validation AP.

For Euler(-SM) models, we tested two different RNNs: GRU [21] and LSTM [39]. For the graph neural network, we only tested GCN [47], as the number of nodes is rather small, and prior works have shown GCN and other models perform about equally on such datasets [69]. Unfortunately, because the heuristic we used to label the edges is our own, there are no prior works with which to compare our method. However, we attempt to present a fair comparison between the Euler(-SM) models and our own implementations of EGCN and VGRNN [35, 63]. We note that these have been modified slightly such that edge lists are used as inputs rather than full adjacency matrices, and the loss functions use negative sampling instead of full adjacency matrix reconstruction.

9.4 Results

As is evident from the results in Table 9, Euler models that make use of the additional softmax layer achieve the greatest results on the test data. Though the difference between an LSTM and

Table 9. Effectiveness of Link Prediction Models on the OpTC Dataset

Dynamic Link Detection						Dynamic Link Prediction					
Model	δ (h)	AUC	AP	TPR (%)	FPR (%)	Model	δ (h)	AUC	AP	TPR (%)	FPR (%)
EGCN-O	5	0.554	0.003	67.5	58.7	EGCN-O	5	0.563	0.003	72.7	63.2
EGCN-H	3.5	0.484	0.002	83.9	85.4	EGCN-H	3.5	0.507	0.003	80.0	80.2
VGRNN	5	0.988	0.367	**99.3**	15.0	VGRNN	0.125	0.692	0.008	73.1	42.1
Euler GRU	2.5	0.888	0.088	17.8	0.473	Euler GRU	3	0.785	0.180	37.6	10.4
Euler LSTM	2.5	0.882	0.118	17.8	0.168	Euler LSTM	3	0.779	0.243	42.7	6.75
Euler-SM GRU	0.125	**0.995**	0.973	97.0	0.021	Euler-SM GRU	0.125	**0.995**	0.969	93.8	0.017
Euler-SM LSTM	0.125	**0.995**	**0.984**	96.7	**0.012**	Euler-SM LSTM	0.5	0.994	**0.986**	**94.9**	**0.013**

a GRU temporal embedding unit is minimal, the LSTM attains slightly higher scores overall. The large increase in AUC and AP between the inner product decoding models and the softmax decoders shows that Euler is not losing information in its encodings; rather, it is being used for reconstruction moreso than anomaly detection. The dedicated network for anomaly detection adds great value to the overall pipeline.

The competing methods performed quite poorly in comparison, likely because they, too, are encoding for reconstruction rather than anomaly detection. Though Euler embeddings alone still have higher metrics than the prior works in link prediction, in link detection, VGRNN is the next best-performing algorithm behind the softmax ones. We note that VGRNN attained the highest TPR of all models, but at the expense of a very high FPR.

Using the lower bound formula from Equation (10), we find the residual error is 109.5. Thus, if we allow the lower bound on the TPR to be 67.5%, that of the lowest ranking model, EGCN-O, then the FPR drops to 3.45%. VGRNN clearly outperforms the EGCN methods, but we cannot draw any conclusions about how VGRNN would compare to the Euler-SM models. Setting the desired FPR to such a low value results in a negative lower bound, which yields very little information. But from the pure AUC and FPR scores alone, we can say with high confidence, the VGRNN likely would not be able to attain such a low FPR with a similarly high TPR.

These results show that the decoder algorithm is a good direction for future work, as small adjustments can produce great improvements without compromising the distributed architecture of Euler.

10 SCALABILITY

The main benefit of using models that fit into the Euler framework is their scalability. While evaluation metrics on benchmarks were generally better than prior work, there were certainly some categories where the advanced models are comparable to our simple ones. However, as we will show, distributed topological encoding has tremendous performance benefits.

We will first identify theoretical complexities of each model tested. For simplicity, we assume the models are implemented using a GCN and a GRU, however, the asymptotic analysis for other GNNs and RNNs are similar, so the following values are general. Using the characteristic equations of GCN [47] and GRU [21], these components' individual computational complexities are $O(|\mathcal{E}|d^2L)$ and $O(Td^2L)$, respectively, where d is the dimensionality of the hidden layer, L is the number of layers, and there are T elements in each sequence. For temporal models that use the snapshotting technique for the embedding portion, the GNN computational complexity becomes $O(T|\mathcal{E}|d^2L)$. However, if we assume that for Euler models there are T workers, then the complexity is the same as a traditional GNN, with the caveat that there will be some communication overhead to synchronize the workers. To account for this, we have added an additional term $C(\omega)$ to represent how communication time scales as more workers are added.

Table 10. Theoretical Runtime Analysis

Framework	$\mathcal{O}$
Euler	$\mathcal{O}(\lvert\mathcal{E}\rvert d_e^2 L_e + T d_r^2 L_r + C(\omega))$
EGCN	$\mathcal{O}(T(\lvert\mathcal{E}\rvert d_e^2 L_e + d_r^2 L_r))$
VGRNN	$\mathcal{O}(T(\lvert\mathcal{E}\rvert d_e^2 L_e + d_r^2 L_r))$

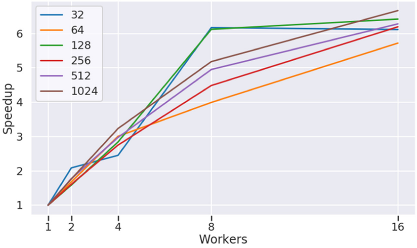
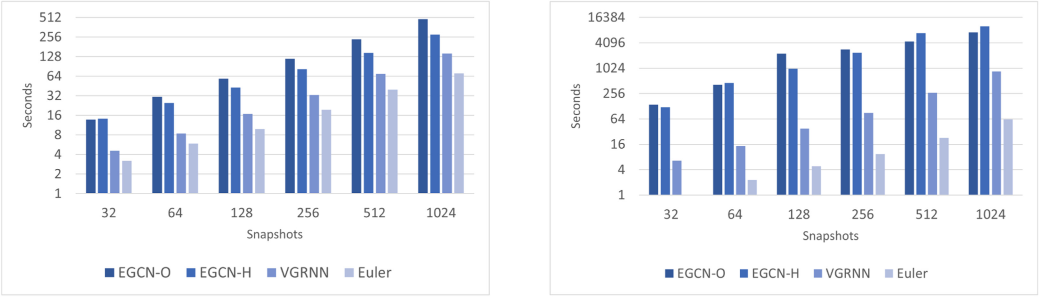


Fig. 7. Performance improvements as more workers are added for varying numbers of snapshots from the LANL dataset. The model used was a GCN stacked on a GRU. All time windows, δ, are 0.5 hours.



(a) Forward propagation time

(b) Backward propagation time

Fig. 8. Performance comparison between the distributed Euler model and competing methods on varying numbers of snapshots from the LANL dataset. All time windows, δ, are 0.5 hours.

For ease of reading, we show time complexities of each model tested in Table 10. As is evident from this comparison, Euler is the only framework that can remove the T term from the expensive GNN portion of the forward pass. As the GNN component scales with the number of edges in the graph, it is clearly the most expensive step. By doing this step just once, rather than T times, as we will show, there are significant performance improvements that outweigh the cost of ω workers.

Figure 7 shows the speedup of a GCN stacked upon a GRU built within the framework of Euler as more workers are added. For these experiments, we evaluate only workers that are powers of two to ensure each worker holds the same number of snapshots. As each worker requires two threads to run—one for the model replica and one for collective communication—our equipment can only accommodate up to 16 workers. The framework allows for the number of workers to be a user-defined hyperparameter, so it is effortless to distribute work across a network with enough nodes to support it. It is evident from the chart that performance improvements are immediate. As more workers are added, runtime improves rapidly, with negligible improvement after about 8 workers for smaller amounts of data. However, as the amount of data processed increases, performance improvements diminish at a much slower rate. This is because the topological encoding task, the bulk of which must occur on CPU due to the high number of random accesses, scales perfectly with additional workers.

In Figure 8, we compare the best runtimes of our method to the runtimes of the serial GNN models we used for benchmarking in Section 6. The serial methods were allowed 16 threads for intraprocess communication for a fair comparison to our 16 worker processes. However, even with this advantage, these methods are forced to process timesteps one at a time; they simply cannot

Table 11. Training Time for the Full Pipelines with $\delta = 0.5$-hour Snapshots

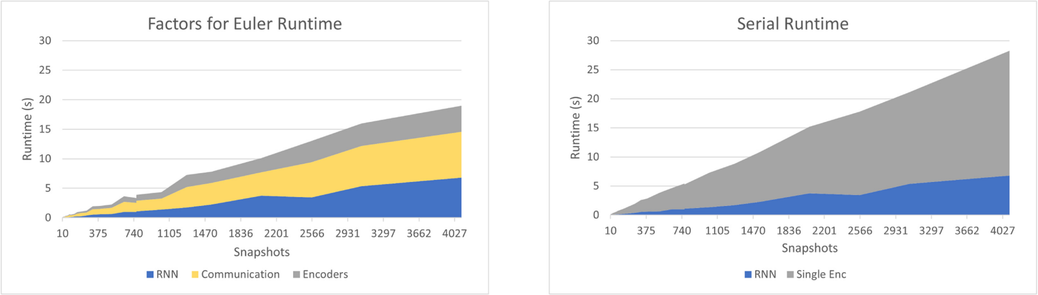
		LANL		OpTC	
		Training Time (s)	TPE (s)	Training Time (s)	TPE (s)
	VGRNN	649.48	11.31	767.53	25.62
	EGCN-H	1,831.46	98.53	488.08	8.47
	EGCN-O	5,047.25	120.87	284.57	6.98
Euler	GCN-GRU	**129.51**	**6.48**	184.11	1.48
	SAGE-GRU	145.99	10.73	**165.14**	1.67
	GAT-GRU	272.79	9.64	190.37	2.30
	GCN-LSTM	115.82	6.90	170.15	**1.44**
	SAGE-LSTM	149.19	11.06	154.69	1.72
	GAT-LSTM	302.94	10.02	237.75	2.57
Euler-SM	GCN-GRU	5,186.87	25.06	1,085.43	20.93
	SAGE-GRU	4,910.13	25.40	1,024.84	20.95
	GAT-GRU	4,835.19	24.14	753.32	21.52
	GCN-LSTM	4,499.26	23.89	714.52	19.78
	SAGE-LSTM	4,808.62	25.65	728.90	19.28
	GAT-LSTM	4,766.39	26.30	762.22	19.33

compete with the efficiency of Euler, especially as the size of data increases. Figure 8(a) shows that as the size of the data being processed increases, Euler's forward propagation speed is 2× that of the fastest competing algorithm. Additionally, by implementing Euler using `DDP` [52], backpropagation is sped up dramatically. Figure 8(b) shows Euler has almost a 16× improvement in backpropagation speed, suggesting backpropagation has near linear scaling as workers are added.

More concretely, we report the exact training time each model took in the previous experiments in Table 11.

As is evident from these results, Euler is noticeably faster than the serial methods. On the LANL data, the next-best method, VGRNN achieves an average **time per epoch (TPE)** of 11.31 s, while Euler with GCN+GRU averages almost half that time with the same number of parameters. On the OpTC data, as there are fewer nodes, and thus a smaller adjacency matrix, the E-GCN models are the best-performing serial models, attaining a TPE of 6.98 s. But still the Euler model improves on this, showing a 4.84× speedup with a 1.44 s TPE. Other combinations of models, especially those using GraphSAGE, which use the more expensive maxpooling aggregation function as well as an additional parameter matrix, are slower but still outperform the serial models. We also note that the full training time is faster for every Euler model, likely due to their simplicity. This means that in addition to individual epochs running faster both forward and backward passes, fewer epochs are required before convergence to, as we have shown, more accurate models. With the softmax augmentation, the neighbor sampling and $|\mathcal{V}|$-dimensional output increase latency significantly. However, we note that their average TPE is still lower than the majority of serial methods. Considering the evaluation improvement they show on smaller datasets, we feel this is an acceptable tradeoff.

Of course, as more workers are added, there will eventually be some diminishing returns. Though we were not able to reach this level with our hardware, it is worth attempting to quantify a theoretical bound on the end of these benefits. To do this, we run a number of experiments. In these experiments, we time individual components of the Euler model in an attempt to break down the exact runtime of each component so we can determine the exact latencies of different phases of the model.



(a) Components of distributed runtime

(b) Components of serial runtime

Fig. 9. Itemized runtimes of a Euler model performing forward propagation on differently sized data with 16 workers.

Figure 9 shows the latency of each phase of the Euler model, broken into runtimes for the MPNN embedders, communication time to synchronize the workers' output, and the RNN's time to process this output. As is evident from these figures, the time it takes to synchronize the communication from each worker is far less than the time it would take to run each worker in serial, the difference between Figures 9(a) and 9(b), respectively. Unsurprisingly, the amount of time it takes to calculate the embeddings in serial is approximately 16× that of what it takes to calculate them in parallel. However, the communication time spent waiting for the tensors to be transmitted over the network, and for the notification that the asynchronous promise has been completed, does take a significant amount of time: approximately three times as long as the embedding phase itself. And this latency varies as more workers are added. To quantify a theoretical bound on the benefit enjoyed as we add additional workers, we must determine how this communication time scales with additional workers, as well as if larger quantities of data has adverse effects on the waiting time.

In Figure 10, we report the communication latency as more workers are added for varying numbers of snapshots the workers process. These values are the average of 10 independent runs across randomly generated Erdös Rényi graphs [27] to ensure realistic load imbalances with similar node counts and branch factors as those found in LANL (1,000 nodes, and $p = 0.2$). Each snapshot generated has between 10,000 and 20,000 edges, the exact value sampled from the uniform distribution, and assigned according to the Erdös Rényi formula. We find that when these latencies are normalized, the size of the data being transmitted has very little effect on performance. However, latency does scale linearly as more workers are added. It approximately follows the slope $L(\omega) = 0.017(\frac{\omega}{15} + \frac{14}{15})$ for $\omega \geq 1$. However, this hardly seems realistic given enough processors, so we will also consider the worst case where $L(\omega) = O(2^{\omega})$. If this is the case on our hardware as more workers are added, then communication overhead roughly doubles every 16 workers. However, this value is highly contingent on the interprocess communication method used and will vary across different implementations of Euler on different hardware.

With this doubling factor found, we present an equation to bound the effectiveness of adding additional workers:

$$Es \leq \frac{s}{\omega}(L(\omega) + E), \tag{12}$$

where E represents the time to encode a single snapshot, $L(\omega)$ represents the communication latency for ω workers, and s is the number of snapshots. The largest positive value ω for which this inequality holds is the maximum number of workers that will still be beneficial to add.

Through the results found in Figure 10, we see that the number of snapshots per worker has little effect on its latency. Thus, we can simplify the inequality by assuming, in a perfect scenario,

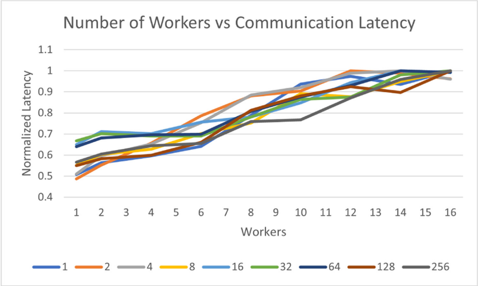


Fig. 10. Effect of new workers on communication latency with varying load sizes, normalized.

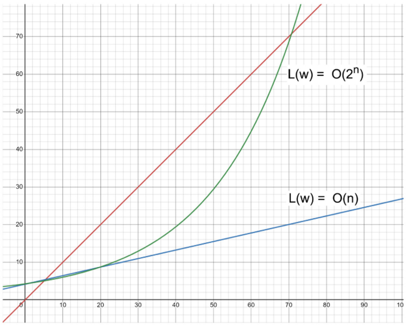


Fig. 11. Approximate theoretical bounds on the benefit of adding additional workers contingent on how $L(\omega)$ scales.

there will be the same number of workers as snapshots. This allows us to rearrange the inequality as

$$\omega \leq \frac{L(\omega)}{E} + 1. \tag{13}$$

Under the assumption that $L(\omega)$ is linear, this inequality holds for all $\omega > 5$, which is dubious. So, we will instead assume $L(\omega) = 0.0172^{\frac{\omega-1}{16}}$ and find the bound in the worst case. Finally, we input the values we found for the various variables to find

$$\omega \leq 0.017 \times 2^{\frac{\omega-1}{16}} \frac{1}{0.005} + 1, \tag{14}$$

which we plot in Figure 11, along with the linear case. Note that the two functions do not significantly diverge until after 20 workers: beyond our hardware's ability to evaluate. This is why we must consider both cases. Thus, after 70 workers, there will be diminishing returns due to the communication overhead in the worst case.[4]

11 CONCLUSION

In this work, we presented the Euler framework: a method to exploit the previously untapped potential for distributing the work done to train and execute temporal graph link predictors. When each topological encoder can operate independently, the most compute-heavy task of generating node embeddings can be scaled in a highly efficient manner. Though models following this framework are necessarily simple, we have shown that for anomalous link detection and prediction, models following the Euler framework perform as well, or better, than their complex contemporaries.

Finally, we showed how this framework can be used to train highly precise anomaly-based intrusion detection systems when network activity is viewed through the abstraction of a temporal graph. These intrusion detection systems are scalable and more sound than other unsupervised techniques despite being trained with less data. Additionally, we found that further improvements could be made to the classifier by implementing a softmax decoder, rather than the simple inner product logistic regression on the raw embeddings. We showed that while the embeddings produced by these models have value on their own, we can extract even more information with additional processing; possibly enough to use this system in a real-world setting.

[4] If we instead assume $L(\omega)$ is polynomial, using least squares regression to find the parabola of best fit, then this value increases to 2,625. The true value is likely somewhere in between.

Future work may include testing other topological or temporal encoders that we did not. As this framework allows for scalable message-passing graph neural networks, future work could even include testing this technique on any large temporal graph dataset previously thought intractable for GNNs.

REFERENCES

[1] 2023. About zeek – Book of zeek (v5.1.0). *Zeek Documentation* (2023). https://docs.zeek.org/en/current/about.html.

[2] 2019. Distributed RPC framework. *PyTorch Master Documentation* (2019). https://pytorch.org/docs/master/rpc.html.

[3] 2022. Pytorch/tensorpipe: A tensor-aware point-to-point communication primitive for machine learning. *Pytorch/tensorpipe* (2022). Retrieved from https://github.com/pytorch/tensorpipe.

[4] Stephen D. Bay, Dennis Kibler, Michael J. Pazzani, and Padhraic Smyth.2000. The UCI KDD archive of large data sets for data mining research and experimentation. *ACM SIGKDD Explorations Newsletter* 2, 2 (2000), 81–85.

[5] 2014. Intel xeon processor E5-2683 v3 (35M Cache, 2.00 GHz) product specifications. *Intel Product Specifications: Processors* (2014). Retrieved from https://ark.intel.com/content/www/us/en/ark/products/81055/intel-xeon-processor-e5-2683-v3-35m-cache-2-00-ghz.html.

[6] Ed Williams, SpiderLab Trustwave, and Edward Millington. 2020. OS credential dumping: LSASS memory, sub-technique T1003.001. *Mitre Att&ck* (2020). https://attack.mitre.org/techniques/T1003/001/.

[7] Ryan Becwar and Vincent Le Toux. 2020. Use alternate authentication material: Pass the ticket, Sub-technique T1550.003. *Mitre Att&ck* (2020). Retrieved from https://attack.mitre.org/techniques/1156T1550/003/.

[8] Jon Sternstein, Mark Wee, Praetorian Netskope, Prasad Somasamduram, Sekhar Sarukkai, Syed Ummar Farooqh, and Yossi Weizman. 2017. Valid accounts. *MITRE ATT&CK* (2017). Retrieved from https://attack.mitre.org/techniques/T1078/.

[9] 2018. Lateral movement, tactic TA0008. *MITRE ATT&CK* (2018). https://attack.mitre.org/tactics/TA0008/.

[10] Ammar Alazab, Michael Hobbs, Jemal Abawajy, and Moutaz Alazab. 2012. Using feature selection for intrusion detection system. In *Proceedings of the International Symposium on Communications and Information Technologies (ISCIT)*. IEEE, 296–301.

[11] Gene M. Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the Spring Joint Computer Conference*. 483–485.

[12] Md Monowar Anjum, Shahrear Iqbal, and Benoit Hamelin. 2021. 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*. 27–32.

[13] Tim Bai, Haibo Bian, Abbas Abou Daya, Mohammad A. Salahuddin, Noura Limam, and Raouf Boutaba. 2019. A machine learning approach for RDP-based lateral movement detection. In *Proceedings of the IEEE 44th Conference on Local Computer Networks (LCN)*. IEEE, 242–245.

[14] Siddharth Bhatia, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. 2020. MIDAS: Microcluster-based detector of anomalies in edge streams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3242–3249.

[15] Haibo Bian, Tim Bai, Mohammad A. Salahuddin, Noura Limam, Abbas Abou Daya, and Raouf Boutaba. 2021. Uncovering lateral movement using authentication logs. *IEEE Trans. Netw. Serv. Manag.* 18, 1 (2021), 1049–1063.

[16] Benjamin Bowman, Craig Laprade, Yuede Ji, and H. Howie Huang. 2020. Detecting lateral movement in enterprise computer networks with unsupervised graph AI. In *Proceedings of the 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID'20)*. 257–268.

[17] Anna L. Buczak and Erhan Guven. 2015. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Commun. Surv. Tutor.* 18, 2 (2015), 1153–1176.

[18] Brian Caswell and Jay Beale. 2004. *Snort 2.1 Intrusion Detection*. Elsevier.

[19] Jinyin Chen, Xuanheng Xu, Yangyang Wu, and Haibin Zheng. 2018. GC-LSTM: Graph convolution embedded LSTM for dynamic link prediction. *arXiv preprint arXiv:1812.04206* (2018).

[20] Pin-Yu Chen, Sutanay Choudhuri, Luke Rodriguez, Alfred Hero, and Indrajit Ray. 2019. Enterprise cyber resiliency against lateral movement: A graph theoretic approach. *Industrial Control Systems Security and Resiliency*, Springer Nature. DOI: https://doi.org/10.1007/978-3-030-18214-4_5

[21] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).

[22] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[23] Benoit Claise. 2004. Cisco systems netflow services export version 9. *The Internet Engineering Task Force (IETF)*, Network Working Group, Technical Report. Retrieved from https://www.ietf.org/rfc/rfc3954.txt.

[24] L. Dhanabal and S. P. Shantharajah. 2015. A study on NSL-KDD dataset for intrusion detection system based on classification algorithms. *Int. J. Adv. Res. Comput. Commun. Eng.* 4, 6 (2015), 446–452.

[25] Aswathy Divakaran and Anuraj Mohan. 2019. Temporal link prediction: A survey. *New Gener. Comput.* 38 (2019), 213–258.

[26] Mohamed Gamal El-Hadidi and Marianne A. Azer. 2020. Detecting Mimikatz in lateral movements using Mutex. In *Proceedings of the 15th International Conference on Computer Engineering and Systems (ICCES)*. IEEE, 1–6.

[27] Paul Erdös and Alfréd Rényi. 2011. On the evolution of random graphs. In *The Structure and Dynamics of Networks*. Princeton University Press, 38–82.

[28] Fahimeh Farahnakian and Jukka Heikkonen. 2018. A deep auto-encoder based approach for intrusion detection system. In *Proceedings of the 20th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 178–183.

[29] Chuanpu Fu, Qi Li, Meng Shen, and Ke Xu. 2021. Realtime robust malicious traffic detection via frequency domain analysis. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 3431–3446.

[30] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural message passing for quantum chemistry. In *Proceedings of the International Conference on Machine Learning*. PMLR, 1263–1272.

[31] Joshua Glasser and Brian Lindauer. 2013. Bridging the gap: A pragmatic approach to generating insider threat data. In *Proceedings of the IEEE Security and Privacy Workshops*. IEEE, 98–104.

[32] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. 2020. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowl.-based Syst.* 187 (2020), 104816.

[33] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. DynGEM: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273* (2018).

[34] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 855–864.

[35] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. 2019. Variational graph recurrent neural networks. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc., 10701–10711. Retrieved from https://proceedings.neurips.cc/paper/2019/file/a6b8deb7798e7532ade2a8934477d3ce-Paper.pdf.

[36] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216* (2017).

[37] Nick Heard and Patrick Rubin-Delanchy. 2016. Network-wide anomaly detection via the Dirichlet process. In *Proceedings of the IEEE Conference on Intelligence and Security Informatics (ISI)*. IEEE, 220–224.

[38] Grant Ho, Mayank Dhiman, Devdatta Akhawe, Vern Paxson, Stefan Savage, Geoffrey M. Voelker, and David Wagner. 2021. Hopper: Modeling and detecting lateral movement. In *Proceedings of the 30th USENIX Security Symposium (USENIX Security'21)*. 3093–3110.

[39] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computat.* 9, 8 (1997), 1735–1780.

[40] Hassaan Irshad, Gabriela Ciocarlie, Ashish Gehani, Vinod Yegneswaran, Kyu Hyung Lee, Jignesh Patel, Somesh Jha, Yonghwi Kwon, Dongyan Xu, and Xiangyu Zhang. 2021. TRACE: Enterprise-wide provenance tracking for real-time apt detection. *IEEE Trans. Inf. Forens. Secur.* 16 (2021), 4363–4376.

[41] Alexander D. Kent. 2015. *Comprehensive, Multi-source Cyber-security Events Data Set*. Technical Report. Los Alamos National Lab. (LANL), Los Alamos, NM.

[42] Alexander D. Kent, Lorie M. Liebrock, and Joshua C. Neil. 2015. Authentication graphs: Analyzing user behavior within an enterprise network. *Comput. Secur.* 48 (2015), 150–166.

[43] Adam Khalid, Anazida Zainal, Mohd Aizaini Maarof, and Fuad A. Ghaleb. 2021. Advanced persistent threat detection: A survey. In *Proceedings of the 3rd International Cyber Resilience Conference (CRC)*. IEEE, 1–6.

[44] Isaiah J. King and H. Howie Huang. 2022. Euler: Detecting network lateral movement via scalable temporal graph link prediction. In *Proceedings of the Network and Distributed System Security Symposium*. Internet Society. DOI:https://doi.org/10.14722/ndss.2022.24107

[45] Samuel T. King and Peter M. Chen. 2003. Backtracking intrusions. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*. 223–236.

[46] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[47] Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[48] Thomas N. Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[49] Jon Kleinberg and Éva Tardos. 2006. Interval scheduling: The greedy algorithm stays ahead. In *Algorithm Design*. Addison Wesley.

[50] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)*. 583–598.

[51] Meicong Li, Wei Huang, Yongbin Wang, Wenqing Fan, and Jianfang Li. 2016. The study of APT attack stage model. In *Proceedings of the IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*. IEEE, 1–5.

[52] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. 2020. PyTorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704* (2020).

[53] Taisong Li, Jiawei Zhang, S. Yu Philip, Yan Zhang, and Yonghong Yan. 2018. Deep dynamic network embedding for link prediction. *IEEE Access* 6 (2018), 29219–29230.

[54] Qingyun Liu, Jack W. Stokes, Rob Mead, Tim Burrell, Ian Hellen, John Lambert, Andrey Marochko, and Weidong Cui. 2018. Latte: Large-scale lateral movement detection. In *Proceedings of the IEEE Military Communications Conference (MILCOM)*. IEEE, 1–6.

[55] Ritika Lohiya and Ankit Thakkar. 2021. Intrusion detection using deep neural network with antirectifier layer. In *Applied Soft Computing and Communication Networks*. Springer, 89–105.

[56] Sadegh M. Milajerdi, Birhanu Eshete, Rigel Gjomemo, and V. N. Venkatakrishnan. 2019. Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 1795–1812.

[57] Sadegh M. Milajerdi, Rigel Gjomemo, Birhanu Eshete, Ramachandran Sekar, and V. N. Venkatakrishnan. 2019. Holmes: Real-time apt detection through correlation of suspicious information flows. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. IEEE, 1137–1152.

[58] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: An ensemble of autoencoders for online network intrusion detection. *Mach. Learn.* 5 (2018), 2.

[59] MITRE. 2022. Data Model | MITRE Cyber Analytics Repository. Retrieved from https://car.mitre.org/data_model/.

[60] Ghulam Mohi-ud din. 2018. NSL-KDD. DOI: https://doi.org/10.21227/425a-3e55

[61] Giang Hoang Nguyen, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-time dynamic network embeddings. In *Proceedings of the Web Conference*. 969–976.

[62] Linshu Ouyang, Yongzheng Zhang, and Yipeng Wang. 2020. Unified graph embedding-based anomalous edge detection. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[63] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. 2020. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 5363–5370.

[64] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 701–710.

[65] Mahmudur Rahman, Tanay Kumar Saha, Mohammad Al Hasan, Kevin S. Xu, and Chandan K. Reddy. 2018. Dylink2vec: Effective feature representation for link prediction in dynamic networks. *arXiv preprint arXiv:1804.05755* (2018).

[66] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2018. Dynamic graph representation learning via self-attention networks. *arXiv preprint arXiv:1812.09430* (2018).

[67] Iqbal H. Sarker. 2021. CyberLearning: Effectiveness analysis of machine learning security modeling to detect cyber-anomalies and multi-attacks. *Internet Things* 14 (2021), 100393.

[68] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. 2018. Structured sequence modeling with graph convolutional recurrent networks. In *Proceedings of the International Conference on Neural Information Processing*. Springer, 362–373.

[69] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868* (2018).

[70] Michele Starnini, Andrea Baronchelli, Alain Barrat, and Romualdo Pastor-Satorras. 2012. Random walks on temporal networks. *Phys. Rev. E* 85, 5 (2012), 056115.

[71] Branka Stojanović, Katharina Hofer-Schmitz, and Ulrike Kleb. 2020. APT datasets and attack modeling for automated detection methods: A review. *Comput. Secur.* 92 (2020), 101734.

[72] Blake E. Strom, Andy Applebaum, Doug P. Miller, Kathryn C. Nickels, Adam G. Pennington, and Cody B. Thomas. 2018. *Mitre attack: Design and philosophy.* The MITRE Corporation, Technical Report.

[73] Aynaz Taheri, Kevin Gimpel, and Tanya Berger-Wolf. 2019. Learning to represent the evolution of dynamic graphs with recurrent models. In *Proceedings of the World Wide Web Conference*. 301–307.

[74] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. 1067–1077.

[75] Jacob Torrey. 2020. GitHub - darpa-i2o/Transparent-Computing: Material from the DARPA Transparent Computing Program. Retrieved from https://github.com/darpa-i2o/Transparent-Computing.

[76] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[77] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer. 2020. A survey on distributed machine learning. *ACM Comput. Surv.* 53, 2 (2020), 1–33.

[78] Anthony Yu-Tung Wang, Ryan J. Murdock, Steven K. Kauwe, Anton O. Oliynyk, Aleksander Gurlo, Jakoah Brgoch, Kristin A. Persson, and Taylor D. Sparks. 2020. Machine learning for materials scientists: An introductory guide toward best practices. *Chem. Mater.* 32, 12 (2020), 4954–4965.

[79] Shen Wang and S. Yu Philip. 2019. Heterogeneous graph matching networks: Application to unknown malware detection. *IEEE International Conference on Big Data* (2019), 5401–5408. https://doi.org/10.1109/BigData47090.2019.9006464

[80] Renzheng Wei, Lijun Cai, Aimin Yu, and Dan Meng. 2021. DeepHunter: A graph neural network based approach for robust cyber threat hunting. *arXiv preprint arXiv:2104.09806* (2021).

[81] Mark Whitehouse, Marina Evangelou, and Niall M. Adams. 2016. Activity-based temporal anomaly detection in enterprise-cyber security. In *Proceedings of the IEEE Conference on Intelligence and Security Informatics (ISI)*. IEEE, 248–250.

[82] Eric P. Xing, Qirong Ho, Pengtao Xie, and Dai Wei. 2016. Strategies and principles of distributed machine learning on big data. *Engineering* 2, 2 (2016), 179–195.

[83] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[84] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. 2019. T-GCN: A temporal graph convolutional network for traffic prediction. *IEEE Trans. Intell. Transport. Syst.* 21, 9 (2019), 3848–3858.

[85] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic network embedding by modeling triadic closure process. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

Received 19 August 2022; revised 3 February 2023; accepted 13 March 2023