LINK-BASED ANOMALY DETECTION IN SYSMON LOGS USING GRAPH NEURAL NETWORKS

DÉTECTION D'ANOMALIES BASÉE SUR LES LIENS DANS LES JOURNAUX SYSMON À L'AIDE DE RÉSEAUX DE NEURONES GRAPHIQUES

A Thesis Submitted to the Division of Graduate Studies
of the Royal Military College of Canada

by

Charles Grimshaw, BEng, rmc
Lieutenant(Navy), MASc Candidate

In Partial Fulfillment of the Requirement for the Degree of
Master of Applied Science in Electrical and Computer Engineering

June 7, 2023

Acknowledgments

My colleagues, Taylor Perkins and Emilie Coote, for their boundless guidance unfettered by normal working hours.

To Chandra Majumdar, whose passion for the field of information security was the primary enabled for this study.

My supervisor, Mr. Brian Lachine, for his constant support throughout this research.

Finally, to my patient partner, Erin Teschuk. If my code ran as fast as she did, this thesis would have been over a long time ago.

Abstract

Anomaly detection is a challenge well-suited to machine learning. In the context of information security, the benefits of unsupervised solutions show significant promise when used in conjunction with traditional signature-based detection techniques. By deriving a context for normal network behaviour, anomaly-based detection systems provide an adaptive alerting engine that can keep pace with the growing volume of novel threats in the field of cybersecurity.

The development of behavioural anomaly detection systems is not new. However, recent attention to Graph Neural Networks (GNNs) has provided an innovative approach to learn from attributed graphs, leveraging both node and edge attributes alongside network structure. While the focus of graph anomaly detection research has been devoted to anomalous node identification, the edges between nodes can be similarly detected during the reconstruction phase of a GNN encoder-decoder architecture. Given that hosts on a computer network can be naturally modeled as attributed nodes on a graph, the network connections between hosts can be used to represent relational edges. This topology is a natural fit for applying link prediction techniques to predict deviating behaviours.

The aim of this research is to determine whether an unsupervised GNN model can detect anomalous network connections in a static, attributed network. In this context, a static network refers to a fixed window of analysis, while attributed denotes the presence of node and edge features. Corporate network logs were collected using endpoint monitoring tools and analyzed to discern the underlying host communication patterns. A GNN-based anomaly detection system was designed and employed to score and rank anomalous connections between hosts. The model was validated against a range of realistic experimental scenarios, incorporating real anomalous data from the large corporate networks as well as an assessment against a smaller artificial network environment. Although quantitative metrics were affected by factors such as the scale of the network, qualitative assessments indicated that anomalies from all scenarios were detected. This exploratory research serves as a promising step for advancing this methodology in detecting anomalous network connections.

Index Terms - Anomaly Detection, Graph Neural Networks, Unsupervised Learning, Link Prediction

Resume

La détection d'anomalies est un défi parfaitement adapté à l'apprentissage automatique. Dans le contexte de la sécurité de l'information, les avantages des solutions non supervisées présentent une promesse significative lorsqu'elles sont utilisées en conjonction avec les techniques de détection basées sur des signatures traditionnelles. En dérivant un contexte pour le comportement normal du réseau, les systèmes de détection basés sur les anomalies fournissent un moteur d'alerte adaptatif capable de suivre le volume croissant de menaces nouvelles dans le domaine de la cybersécurité.

Le développement de systèmes de détection d'anomalies comportementales n'est pas nouveau. Cependant, l'attention récente portée aux réseaux neuronaux graphiques (GNN) a permis une approche novatrice pour apprendre à partir de graphiques attribués, en tirant parti à la fois des attributs des nœuds et des arêtes ainsi que de la structure du réseau. Alors que la recherche sur la détection d'anomalies dans les graphiques s'est concentrée sur l'identification des nœuds anormaux, les arêtes entre les nœuds peuvent également être détectées lors de la phase de reconstruction d'une architecture encodeur-décodeur GNN. Étant donné que les hôtes d'un réseau informatique peuvent être naturellement modélisés comme des nœuds attribués sur un graphique, les connexions réseau entre les hôtes peuvent être utilisées pour représenter des arêtes relationnelles. Cette topologie est parfaitement adaptée à l'application de techniques de prédiction de liens pour prédire les comportements déviants.

L'objectif de cette recherche est de déterminer si un modèle GNN non supervisé peut détecter des connexions réseau anormales dans un réseau attribué statique. Dans ce contexte, un réseau statique fait référence à une fenêtre d'analyse fixe, tandis qu'attribué indique la présence de caractéristiques des nœuds et des arêtes. Les journaux de réseau d'entreprise ont été collectés à l'aide d'outils de surveillance des points de terminaison et analysés pour déterminer les modèles de communication sous-jacents des hôtes. Un système de détection d'anomalies basé sur le GNN a été conçu et utilisé pour évaluer et classer les connexions anormales entre les hôtes. Le modèle a été validé sur une gamme de scénarios expérimentaux réalistes, en utilisant des données anormales réelles provenant de grands réseaux d'entreprise ainsi qu'une évaluation contre un environnement réseau artificiel plus petit. Bien que les mesures quantitatives aient été affectées par des facteurs tels que l'échelle du réseau, les évaluations qualitatives ont indiqué que les anomalies de tous les scénarios étaient détectées. Cette recherche exploratoire constitue une étape prometteuse pour faire progresser cette méthodologie dans la détection des connexions réseau anormales.

Mots-clés - Détection d'anomalies, Réseaux de neurones graphiques, Apprentissage non supervisé, Prédiction de liens

List of Figures

List of Tables

Listings

List of Acronyms

ANE Artificial Network Environment.

APT Advanced Persistent Threat.

ATT&CK Adverserial Tactics, Techniques, and Common Knowledge.

AUC Area Under the Curve.

AWS Amazon Web Services.

BCE Binary Cross-Entropy.

C2 Command and Control.

DDoS Distributed Denial of Service.

EDA Exploratory Data Analysis.

FN False Negative.

FP False Positive.

GAE Graph Autoencoder.

GAT Graph Attention Network.

GNN Graph Neural Network.

GUID Globally Unique Identifier.

IDS Intrusion Detection System.

IHGAT Intention-Aware Heterogeneous Graph Attention Network.

IP Internet Protocol.

ISP Internet Service Provider.

JSON JavaScript Object Notation.

LANL Los Alamos National Laboratory.

LSTM	Long Short-Term Memory Network.
NDJSON	Newline-Delimited JavaScript Object Notation.
NMAP	Network Mapper.
NMF	Non-negative Matrix Factorization.
OUI	Organizationally Unique Identifier.
PID	Process Identifier.
REST	Representational State Transfer.
RFC	Request for Comments.
RNN	Recurrent Neural Network.
S3	Simple Storage Service.
SIEM	Security Information and Event Management.
TCP	Transmission Control Protocol.
TN	True Negative.
TP	True Positive.
UDP	User Datagram Protocol.

Contents

1 Introduction

Anomaly-based detection systems offer a substantial benefit over signature-based methods as they can identify new and previously unknown threats. However, these systems often generate a high number of false positives due to challenges in accurately learning typical system behaviour. To improve their performance, it is helpful to process the data using representations that better reflect the inherent structures present in the data. Graph Neural Networks (GNN) provide a potential solution for processing graph data and learning from both features and relational information within the data. This thesis explores the application of graph anomaly detection, specifically by modelling anomalous connections as edges between hosts on a graph. This experiment involves an in-depth examination of host data gathered from four enterprise networks and a simulated network environment, encompassing exploratory data analysis, model architecture, and performance evaluation metrics.

1.1 Motivation

The number of threats in the information security environment is proliferating. Hundreds of thousands of new malware samples are detected daily [1], and a recent survey of Canadian industry determined the average costs of a data breach exceeded CAD 7 million [2]. These costly figures highlight the demand for intelligent defences to mitigate increasingly numerous threats.

Intrusion Detection Systems (IDS) are a standard layer in the network security stack. These systems monitor host logs and network traffic, raising alerts based on detected policy violations. The most common variant of IDS software uses signature-based detection [3], scanning for unique patterns extracted from malware. The straightforward nature of this technique allows signature-based detection systems to operate effectively within a broad and well-defined range of attacks. However, the effectiveness of signature-based detection is confined to known threats, thereby making it impossible to identify new and emerging attacks.

Empowered by significant advances in Machine Learning, anomaly-based detection systems have become a popular area of research. Anomaly-based detection systems model the behaviour of the host and/or network, reporting deviations from a learned context. This framework enables the detection of novel threats, bridging shortcomings from signature-based techniques. However, anomaly-based systems can suffer from false positives: by raising alerts on benign data, resources can be wasted. The challenge of accurately detecting anomalies in network systems is further compounded by the massive volume of log data generated by modern, large-scale enterprise networks [4]. The sheer amount of data available in these networks adds to the complexity of learning normal system behaviour and distinguishing between genuine threats and benign activities, making it even more crucial to develop effective methods for processing and analyzing this data.

To improve the performance of anomaly-based detection systems, the underlying representations within logged data can be exploited. While convolutional neural networks are a robust architecture for computer vision tasks, and recurrent networks effectively process sequential data like natural language, computer networks are a strong fit for GNNs due to relationships present in the underlying data [5]. Graph anomaly detection is a comparatively new field of machine learning, and most research into the use of GNNs has focused on identifying anomalous nodes in present in networks [5], [6]. However, edge-based anomaly detection techniques have effectively identified fraudulent transactions in e-commerce networks [6], [7]. This link detection approach can also be applied to local computer networks with relatively static hosts, where the behavior of incoming and outgoing connections may vary significantly during anomalous activity.

1.2 Statement of Deficiency

Anomalous edge detection remains a relatively under-explored area in the information security domain. Investigating edge-based anomaly detection systems for modelling suspicious connections within computer networks presents an opportunity to enhance threat detection. By evaluating connections based on their abnormality in the context of the larger network, it is possible to develop a valuable heuristic for identifying potential threats and improving the overall effectiveness of anomaly detection systems in the field of information security.

1.3 Aim

The aim of this research is to determine whether an unsupervised GNN model can detect anomalous network connections in a static, attributed network. The benefit of using attributed network models lies in leveraging node and edge features in conjunction with relational information to produce a more representative model. The data attributed to the model is derived from host-based sources related to network connections, and to narrow the scope of analysis, browsers will be excluded.

Excluding browser traffic is useful for constraining the scope of the analysis because browser traffic is typically quite high in volume and often involves numerous connections to various external servers. By removing browser traffic, the research can focus on more specific types of network connections and help reduce noise in the data to make it easier to identify meaningful patterns and relationships.

A static network refers to a network that is analyzed based on a fixed snapshot in time, rather than studying the network as it evolves over time. The network structure and its features are considered at a particular point in time, and any changes or updates to the network are not taken into account during the analysis. This facilitates a more manageable and focused analysis, as it constrains the scope of the study to a specific, unchanging network state.

The use of host data for anomaly detection provides a granular window into the state of the host execution environment during a period of analysis. For this research, host data from endpoint monitoring systems *Sysmon* [8] and *Osquery* [9] is used to characterize communications on the network via features queried from host runtimes.

The lateral movement phase of an adversarial campaign is often characterized by new, both failed and successful, connections within a computer network, which can be indicative of anomalous activity when compared to a derived behavioral baseline [10]. However, the training and evaluation of an unsupervised anomaly detection system in real-life corporate networks across various business sectors presents significant challenges. The presence of a diverse range of users, activities, and networked devices complicates the task of defining a normal context, making the detection of anomalies a challenging endeavor.

Furthermore, the relatively small quantities of labelled data presents challenges when evaluating unsupervised models [11], [12]. To assess the effectiveness of an edge-based anomaly approach, synthetic data injection with known anomalies is used. Synthetic data injection involves creating artificial data points or modifying existing data points to represent anomalous behavior in a controlled manner. By introducing known true positives into the dataset, the performance of the anomaly detection model can be assessed. It is important to ensure that the synthetic data is representative of real-world data and scenarios, as the performance associated with the anomaly detection model on synthetic data may not always generalize to real-world situations [11].

1.4 Research Activities

The following research activities are conducted to evaluate the performance of edge-based anomaly detection systems when applied to large computer networks:

1. **Data Extraction, Exploratory Data Analysis, and Feature Engineering**. This

consists of host data collected from endpoints using *Sysmon* and *osquery.* Following data extraction, exploratory data analysis (EDA) is performed. EDA includes basic statistical and comparative analysis of the five different datasets and graph visualizations to characterize communications on the different networks in order to inform feature engineering and model development. Following EDA, core feature engineering is conducted, leading into model development.

2. **Model Development**. This includes the development and training of an unsupervised detector where node and relational features, and GNN hyper-parameters are investigated and adjusted to produce a functional experimental model.

3. **Model Evaluation**. In this phase, model performance is measured by detecting a known quantity of synthetic data into the corporate network capture. Four different scenarios capture a variety of anomalous network behaviours, yielding metrics that are used to assess the performance of the model under different conditions.

1.5 Results

A log extraction tool was developed to query vast amounts of unstructured logs, significantly streamlining the data analysis process. A comprehensive exploratory data analysis was conducted on four large corporate networks, highlighting information about their structure and characteristics. Quantitative metrics obtained from scenarios indicated a relationship between the performance of the anomaly detection system and the scale of the network. Qualitatively, the model demonstrated its capability to detect relational anomalies within large graph structures. In a separate validation effort, the model was applied to a smaller, simulated attack network, resulting in considerably enhanced performance. Notably, the model successfully detected key features of the simulated attack campaign, highlighting its potential utility in real-world situations. These findings provide a stepping stone in advancing the field of graph anomaly detection, particularly in the context of anomalous edge detection, and pave the way for further development and refinement of techniques that can robustly identify and respond to network anomalies in diverse network data.

1.6 Organization

The remainder of this thesis provides a deeper context for the research and how it was conducted. Chapter 2 presents background information on endpoint monitoring tooling, anomaly detection, graph neural networks, and specific challenges associated with the analysis of large networks. This chapter also introduces the Graph Autoencoder (GAE) model and its application in detecting anomalous edges within complex graphs. In Chapter 3, the methodology and design of the experiment is described, detailing the data collection and preprocessing, feature selection and engineering, GAE model development, and evaluation metrics. Chapter 4 presents the results of the experiment, highlighting the model's performance in a selection of scenarios and against a simulated attack campaign in an artificial network environment. Finally, Chapter 5 offers a summary and enumerates the contributions of the research, as well as outlining potential areas for future work in the domain of this research.

1.7 Summary

This chapter has offered a brief introduction to the proposed research on anomalous edge detection using graph neural networks. The adoption of graph representation to model host data is expected to generate a system that accurately mirrors the inherent features and relationships of the underlying communications. The set objectives will be pursued through three primary research activities: Data Extraction, Exploratory Data Analysis, and Feature Engineering, Model Development, and Model Evaluation. These activities will involve the extraction of log data, exploratory data analysis (EDA), and feature engineering and selection, followed by the development and training of an unsupervised detection model. The model's effectiveness will then be gauged through its performance against corporate network data, in which synthetic labelled data has been injected. This research explores the potential of identifying suspicious connections in a local network through anomalous edges.

2 Background

This chapter presents the theoretical foundations and practical applications that underpin the research. The discussion begins with an exploration of two essential endpoint monitoring tools, System Monitor and Osquery. The focus then shifts to the Simple Storage Service (S3) and the concept of word embeddings, followed by an examination of graphs as an abstract data type, with a specific look at the PageRank algorithm. The discussion also incorporates the MITRE ATT&CK Framework, a critical resource in the field of cybersecurity.

The latter part of the chapter delves into relevant machine learning concepts. It begins with an overview of unsupervised learning, focusing on anomaly detection and its evaluation metrics. The discussion then transitions to Graph Neural Networks (GNNs) and their mechanism of Neural Message Passing. Autoencoders are next in line, specifically their application in a Graph Autoencoder Architecture, and Graph Attention Networks. An illustration of GNN Encoding provides practical insights into its operation.

The chapter also explores the concept of link prediction as a method for detecting anomalous edges. Finally, it reviews related works in the field, including key studies on network lateral movement detection, anomalous edge detection, and fraud transactions detection using graph-based techniques. These topics build a comprehensive foundation for the research methodology and design outlined in Chapter 3.

2.1 Endpoint Monitoring Tools

Monitoring the state of devices on a network is a critical requirement for information security personnel. Endpoint monitoring tooling is used to continuously record these metrics as structured log data, expediting analysis of host and network-level behaviours. These logs are typically aggregated in Security Information and Event Management (SIEM) systems, which can provide a single interface to analyze events. The experiment uses *Sysmon* and *Osquery* to retrieve host data from a corporate network, which provides a combination of host and network-level data for further processing

2.1.1 System Monitor

System Monitor (*Sysmon*) is a Windows system driver used to monitor system behaviour [8]. *Sysmon* produces structured log data which can be collected and analyzed for anomalies. To integrate with log management tooling, records are logged in a newline-delimited JSON (NDJSON) format, visualized below in Figure 1.

```
{
    "event_id": 3,
    "hostname": "cgrimshaw",
    "utc_timestamp": "2022-09-26T22:45:25Z",
    "event_data": {
      "source_ip": "10.10.0.1",
      "dst_ip": "8.8.8.8",
      "src_port": 52091,
      "dst_port": 52,
    },
    ...
}
```

Figure 1: Abbreviated event ID 3 record with logging metadata

The events monitored by *Sysmon* provide a comprehensive overview of the host execution environment. *Sysmon* events are enumerated by a numeric identifier (ID). For example, *Sysmon* Event ID 1 captures all process creation events, Event ID 22 is recorded whenever a process executes a DNS query, and *Sysmon* Event ID 3 is a *Network Connection Event.* An exhaustive list of *Sysmon* Event IDs is provided in Appendix A, Section 1.1.

Network Connection Events occur in response to any TCP/UDP connections made on the host machine. Each connection is linked to a sending/receiving process via Process ID and Process Globally Unique Identifier (GUID). *Sysmon* also provides information on source/destination Internet Protocol (IP) addresses and ports. In total, 17 features are recorded in Event ID 3, listed in Table 1, providing an abundance of context per network connection [13].

2.1.2 Osquery

Osquery is an open-source infrastructure monitoring tool created in 2014 at Facebook by Mike Arpaia, Ted Reed, and Javier Marcos de Prado [9]. It is composed of two separate tools: an administrative command shell, and daemon processes running on remotely configured endpoints. The *Osquery* daemon turns a host operating system into a queryable relational database, allowing users to explore operating system data. When deployed on multiple hosts across a corporate network, *Osquery* provides a granular window into the state of network behaviour.

Table 1: Fields present in *Sysmon* Event ID 3

Field	Description
DestinationIsIpv6	Indicates if the destination is an IPv6 address
DestinationIp	Destination IP address
DestinationHostname	DNS name of the contacted host
DestinationPort	Destination port number
DestinationPortName	Name of the destination port in use
Image	The file path of the process that made the network connection
Initiated	Indicates whether the process initiated the TCP connection
ProcessGuid	The GUID of the process that made the network connection
ProcessId	PID of the process that made the connection
Protocol	Protocol used for the network connection (e.g. TCP, UDP)
SourceIsIpv6	Indicates if the source IP is an IPv6 address
SourceIp	Originating IP address
SourceHostname	DNS name of the host that made the network connection
SourcePort	Source port number
SourcePortName	Name of the source port in use
User	Name of the account that made the network connection
UtcTime	Time in UTC when event was created

The current version of *Osqeury* realizes host operating system information as 274 distinct tables. The code listing in Figure 2 demonstrates a simple query over multiple schema (the *process* and *listening* tables) to report unique listening processes and related features. This research leverages schema related to Internet Protocol (IP) addresses and their associated hardware interfaces. The usage of *Osqeury* is considered due to the rich feature selection available, and leveraging *Sysmon* in tandem allows node and edge data to be strengthened with more host data.

2.2 Simple Storage Service (S3)

The raw data collected from the corporate networks for this research was initially pushed to an Amazon S3 (Simple Storage Service) instance. Amazon S3 is a scalable cloud object storage service offered by Amazon Web Services (AWS) [14]. S3 is designed to store and retrieve large quantities of data from anywhere on the internet, making it a common choice for various applications, such as content storage and delivery, big data analytics, and log archival.

Numerous open-source systems can be S3-compliant by implementing the S3

```
$ osqueryi
osquery> SELECT DISTINCT
    ...>   process.name,
    ...>   listening.port,
    ...>   process.pid
    ...> FROM processes AS process
    ...> JOIN listening_ports AS listening
    ...> ON process.pid = listening.pid
    ...> WHERE listening.address = '0.0.0.0';

+----------+-------+-------+
| name     | port  | pid   |
+----------+-------+-------+
| Spotify  | 57621 | 18666 |
| ARDAgent | 3283  | 482   |
+----------+-------+-------+

```

Figure 2: A snippet of Osquery usage with resulting output

RESTful API standards, which enables integration and interaction with the Amazon S3 service. This compliance allows developers to utilize the robust capabilities of Amazon S3, while maintaining the flexibility and extensibility of an open-source solution.

2.3 Word Embeddings

Word embedding algorithms transform a vocabulary into dense, fixed-dimension vector representations. These embeddings offer several advantages over one-hot vector encoding strategies, where each word in the vocabulary is represented by a unique binary feature. First, the reduced dimensionality of word embeddings is computationally efficient. Second, embeddings help mitigate the "Curse of Dimensionality", a challenge associated with learning from sparse, high-dimensional data [15]. Lastly, learned representations capture more meaningful and nuanced relationships between words than encoding strategies that treat each word as an independent feature. This improved understanding of language can enhance the performance of machine learning tasks that rely on natural language processing [15].

Several popular word embedding strategies, such as word2vec, GloVe, and fastText, are commonly used to represent discrete sets as vectors. These approaches have gained widespread adoption for their effectiveness in capturing linguistic relationships in vector form [15] - [16].

2.4 Graph (Abstract Data Type)

Graphs are a data structure used to represent entities and the relationships between them. This data type is formally represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, denoting a set of vertices $\mathcal{V}$ connected by a set of edges $\mathcal{E}$. Edges are used to associate vertices. Adjacent vertices are called neighbours. The *degree* of a vertex refers to the number of neighbours that it possesses. Many relations can be modeled using graphs, including molecular structures, social media relationships, and computer networks. An example of such a structure can be found below in Figure 3.

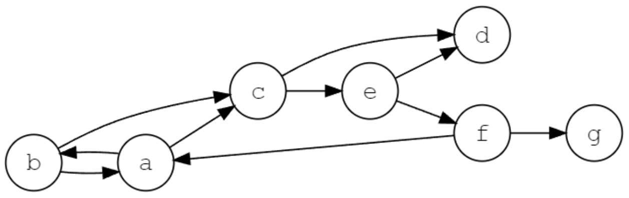


Figure 3: An unweighted directed graph

Graphs can be enhanced with a variety of data describing the entities and the types of relations between them. Nodes may possess a set of *attributes*, or features. Edges may possess direction, which impose constraints on the relationships between vertices, and weight, which adds a value to the link.

Graphs can also be *static* or *dynamic* in nature. A static graph is a fixed sequence of nodes and relations. In contrast, a dynamic graph may change over a period of analysis. Static graph approaches facilitate offline optimization and processing, but do not scale well to variable dimensional sizes [5]. Dynamic graphs are flexible, but do not allow for the same level of graph optimization as their static counterparts [5]. Lastly, graphs may also be *multi-relational*, where two nodes may possess multiple edges.

Internally, graphs are most commonly represented using an adjacency matrix $A \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$. An unweighted edge exists between two nodes i and j if $A[i, j] = 1$, and 0 otherwise. If nodes possess features, they can be represented as a supplementary matrix $X \in \mathbb{R}^{|\mathcal{V}| \times m}$, where m is the number of features per-node. These feature matrices can be processed alongside the adjacency matrix during graph-level encoding discussed later in this chapter.

2.4.1 PageRank

PageRank is an algorithm used to determine the relative importance of nodes in a directed graph [17]. It was originally developed for ranking hyperlinked web pages, but can be applied to any directed graph, where nodes represent entities and edges represent the relationships between them.

In a directed graph, each node can have multiple incoming and outgoing links. PageRank assigns a score to each node based on the importance of the nodes that link to it. Nodes that have many high-scoring incoming links from other important nodes will have a higher PageRank score than nodes with fewer or lower-scoring incoming links.

The algorithm begins by iteratively calculating the PageRank score of each node based on the scores of the nodes that link to it. The PageRank score of a node is determined by the sum of the PageRank scores of its incoming nodes, divided by the total number of outgoing edges from those nodes. This calculation is repeated until the PageRank scores converge to a stable value.

The usefulness of the PageRank algorithm lies in its ability to leverage the relationships between nodes to provide a quantitative measure of their relative importance, enabling more efficient and effective navigation and analysis of complex directed graphs.

2.5 MITRE Attack Framework

The MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) Framework is a comprehensive compendium of knowledge concerning the diverse tactics, techniques, and procedures (TTPs) leveraged by cyber adversaries at various stages of a cyber attack [18]. Created in 2013 by MITRE, a non-profit organization that manages federally-funded research and development centers in the US, the framework undergoes continuous updates with the latest threat intelligence to stay current with emerging threats and the evolving threat landscape.

The framework presents a standardized language for describing the different stages of a cyber attack and the tactics and techniques employed by attackers in each stage. It is represented in the form of a matrix, where each technique is mapped to one or more tactics.

Organizations can utilize the MITRE ATT&CK Framework to assess their cybersecurity defenses by matching their existing security controls to the various techniques used by attackers. By doing so, they can identify gaps in their security posture and take necessary steps to enhance their defenses, thereby reducing the likelihood of a successful cyber attack.

2.6 Unsupervised Learning

Unsupervised learning refers to algorithms used to identify patterns in data sets containing data points that are neither classified nor labelled [19]. These algorithms are able to classify, label and group the data points with no supervision for the overall task.

As no labels are provided to the learning algorithm, the model attempts to find structure in the input data set using only features present in the observations. Consequently, the main goal of unsupervised learning is to discover hidden and interesting patterns in unlabeled data [5], [19].

2.6.1 Anomaly Detection

Anomaly detection is a data analysis problem dealing with the identification of events outside of a profiled threshold. Such events may signal potentially interesting or otherwise notable occurrences [4]. Due to the large volume of unlabelled computer networking data, the anomaly detection problem in the information security domain is typically treated as an unsupervised learning task.

In the context of graph anomaly detection, the task can be further subdivided into three different types of anomalies [19] [20]:

1. Node anomalies;
2. Edge anomalies; and
3. Sub-graph anomalies.

Node anomalies may deviate from the remainder of the graph via anomalous features or anomalous structure, which is based on an abnormal state of relations with adjacent nodes [19]. Edge-based anomaly detection is concerned with scoring abnormal or atypical interactions between nodes in a network [6], [21], [12]. Finally, sub-graph level anomaly detection is an area of research concerned with identifying subsets of the overall graph that do not conform with the overall structure of the graph [7].

In exceptionally large data sets sourced from computer networks, anomaly detection can pose challenges that affect the overall performance and accuracy of detection algorithms [19], [20]. Three of these challenges include:

1. **Scalability**: Large networks typically consist of a massive number of nodes and edges, which can lead to an exponential growth in the amount of data to be processed. Anomaly detection systems need to be designed with scalability in mind to efficiently handle the volume of data.

2. **High dimensionality**: Data collected from networks often encompasses numerous features from a diverse range of sources, which may result in high-dimensional data. To ensure manageable computational complexity, it is essential to employ effective feature selection techniques in the anomaly detection pipeline. This facilitates the identification of the most relevant features while reducing the overall dimensionality of the dataset.

3. **Dynamic nature**: Live networks are constantly changing as new devices are added, existing devices are removed, and network configurations change. Anomaly detection algorithms should provide some mechanism of either fixing the window of analysis or provide an adaptive solution.

Addressing these challenges is crucial for the development of an effective anomaly detection solutions that can provide reliable results in large networks.

2.6.2 Evaluation Metrics

In supervised learning, a model's performance can be evaluated by evaluating predictions against a set of known labels. However, in unsupervised anomaly detection, the goal is to identify patterns or anomalies in data without any prior knowledge of what those anomalies might be.

Anomaly detection is often realized as a binary classification task: a pattern of data either conforms to an expected behaviour, or it does not [20]. However, anomalies are typically scored along a continuous range $\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$, with higher scores denoting greater abnormality [6], [21]. A user-defined threshold is used to realize the final classification of anomalies. This classification process can have four distinct outcomes:

1. **True Positive (TP)**: An anomalous data point is correctly labelled;

2. **True Negative (TN)**: A benign data point is correctly labelled;

3. **False Positive (FP) / Type I error**: A benign data point is incorrectly labelled anomalous; and

4. **False Negative (FN) / Type II error**: An anomalous data point is incorrectly labelled benign.

From these four metrics, further performance characteristics can be derived. A model's *recall*, also known as *true positive rate*, is the model's ability to correctly label outlying data from the selected anomalies.

$$recall = \frac{TP}{TP + FN} \tag{1}$$

Similarly, *precision* is the probability of correctly labeling data as anomalous when considering only the instances that the model has predicted as anomalous.

$$precision = \frac{TP}{TP + FP} \tag{2}$$

Recall and precision metrics can be combined to give an overall measure of predictive performance, realized as a model's F_1 score. The F_1 score is calculated as the harmonic mean of *precision* and *recall*.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3}$$

Holistically, these metrics can be limited in their ability to provide a comprehensive evaluation of the model's performance, as they may not take into account the specific context or application domain [19], [21]. Anomalies in a computer network are not constrained to malicious behaviors, because they can refer to any deviation from normal patterns or expected behavior [20]. This can encompass a wide range of activities, including system errors, hardware failures, configuration changes, unusual user activities, or even legitimate traffic spikes. While some anomalies may indicate security threats or malicious intent, many are simply the result of benign events, user mistakes, or natural fluctuations in network usage. Consequently, defining what constitutes an anomaly is non-trivial, and can vary greatly depending on the application domain [22]. Behaviours that are considered anomalous in one network may not be anomalous in another [20]. Additionally, the concept of an anomaly may evolve over time as new types of anomalies are discovered or as the underlying data changes [19].

As a result of the challenges associated with evaluating unsupervised anomaly detection systems, it is important to consider the qualitative performance of the anomaly detection system and understand the behaviours yielding the binary classification metrics.

2.7 Graph Neural Networks

Graph Neural Networks (GNNs) are models that can process and learn from graph-structured data, enabling them to capture complex relationships and patterns between entities. Simple encoding strategies like 1-hot encoding can represent categorical data locally, but has the unfortunate consequence of producing producing a sparse vector of mostly zeroes. Moreover, this local representation cannot convey relationships between observations. GNNs provide a more generalized processing approach, mapping nodes, relations, and all corresponding feature vectors into a latent space, as visualized in Figure 4 [5]. These mappings, referred to as embeddings, are vectorized representations of the original nodes, and are functionally similar to the word embeddings discussed earlier, in section 2.3.

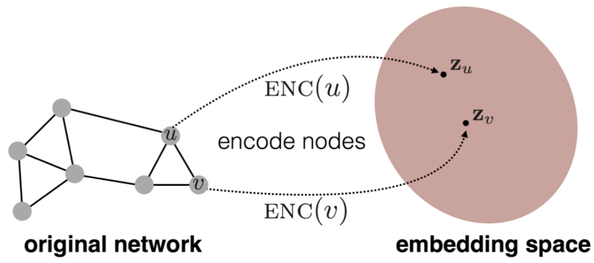


Figure 4: Node encoding into a low-dimension feature space

GNNs have been adopted to efficiently and intuitively detect anomalies from graphs due to their ability to process complex relationships in data [3], [23]. Modern graph anomaly detection approaches combine GNNs with existing deep learning approaches [5] - [6]. GNN models facilitate the detection of anomalies by encoding both graph topology and features simultaneously.

The naive processing scheme for graph data, suggested by Hamilton et. al., involves directly training a neural network on the adjacency matrix A, treating each row in the matrix as distinct node embeddings [5]. However, this approach is not *permutation invariant*—it considers the ordering of nodes as a learned context, even though the order of node/edge occurrences is irrelevant in graph data. GNNs address this issue by accurately representing node embeddings while preserving the vital property of permutation invariance, ensuring that algorithms do not rely on fixed node or relation ordering. This characteristic is achieved through the implementation of Neural Message Passing, which serves as the foundation for GNNs' underlying architecture.

2.7.1 Neural Message Passing

The key framework powering GNN architecture is the concept of Neural Message Passing, which serves as a generalized convolution algorithm over graph data [5]. This algorithm iteratively updates nodal encodings by aggregating neighbouring states along with the current state of the node.

Node u possesses an *encoding* at epoch k, which is equal to $h_u^{(k)}$. The node's updated encoding at the next epoch $k+1$ can be calculated by aggregating state information over its neighbours, a set of vertices produced by $\mathcal{N}(u)$:

$$h_u^{(k+1)} = \mathbf{UPDATE}^{(k)}\left(h_u^{(k)}, \mathbf{AGGREGATE^{(k)}}(\{\mathbf{h_v^{(k)}}, \forall \mathbf{v} \in \mathcal{N}(\mathbf{u})\})\right) \tag{4}$$

The functions *UPDATE* and *AGGREGATE* are trainable neural networks. After K iterations of this algorithm node u's final encoding z_u can be derived:

$$z_u = h_u^{(K)}, \forall u \in \mathcal{V} \tag{5}$$

The key insight in (4) is that the *AGGREGATE* function accepts a set of vertices, making the algorithm permutation invariant. After K layers, the encoding z_u possesses information about nodes from K hops away.

The simplest message-passing implementation sums the incoming messages from neighbouring nodes as an aggregation function [5]. This operation is defined below:

$$h_u^k = \sigma\left(w_{self}^k + w_{neigh}^k \sum_{v \in N(u)} h_v^{(k-1)} + b^k\right) \tag{6}$$

Where w_{self}^k and w_{neigh}^k are trainable parameter matrices, b^k represents a bias term, and σ denotes element-wise non-linearity function such as tanh or $ReLU$. Messages from neighbouring nodes are combined with the node's previous state prior to the non-linear transformation.

Various aggregation and update functions exist: mean aggregation can capture the distribution of nodes, maximum aggregation can be used to identify representative nodes, and sum aggregation facilitates the learning of graph topology [24]. The most popular GNN layer is the graph convolutional network (GCN) layer [5]. However, traditional GCN implementations cannot handle edge features and weigh neighbouring nodes with equal priority [25], [26].

2.8 Autoencoders

Autoencoders are a type of artificial neural network trained to learn efficient low-dimensional representations of input data in an unsupervised manner [20]. Autoencoders are commonly composed in a two-step process: an encoder, responsible for mapping the input data into a compressed representation, and a decoder, which reconstructs the original signal from the low-dimensional representation [20]. As seen in Figure 5, this model forces input features into a lower-dimensional space (referred to as a "bottleneck") [23], before being reconstructed. Ideally, the resultant representations can be used to reconstruct the original data perfectly. In practice, however, the models are lossy; errors accrue during input reconstruction due to information loss during the encoding phase of the model [23].

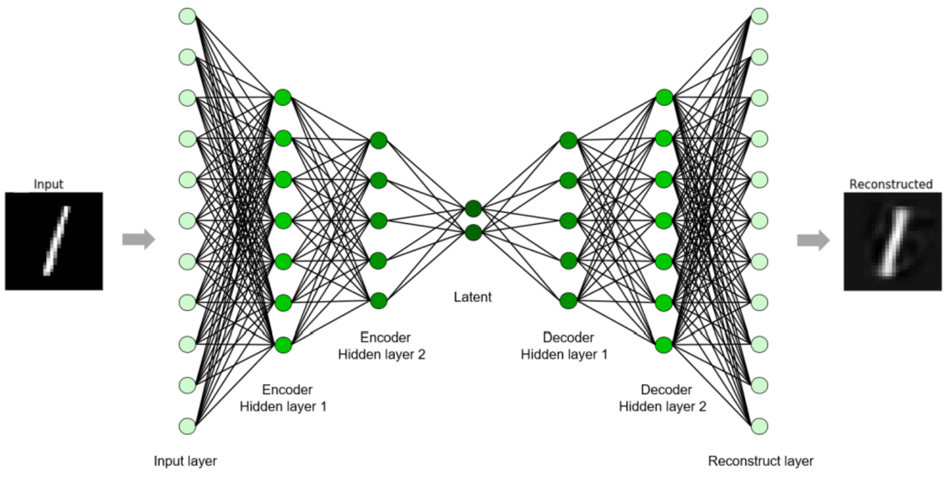


Figure 5: The "bottleneck" architecture of an autoencoder network [23]

When using autoencoders for anomaly detection, there is a trade-off between reconstruction accuracy and anomaly detection sensitivity. If an autoencoder is trained to perfectly reconstruct the input data, it may not be able to detect anomalies that deviate only slightly from the norm. However, if the autoencoder is trained to be highly sensitive to anomalies, it may reconstruct the input data poorly and introduce false positives into the system.

Therefore, the choice of reconstruction accuracy depends on the specific application and the nature of the expected anomalies. In some cases, it may be more important to accurately reconstruct the input data. In contrast, it may be more important to detect anomalies even at the cost of introducing some reconstruction errors.

In practice, a decision threshold is often applied to the reconstruction error, such that input data with a reconstruction error below the threshold are classified as belonging to the normal population, and error values above the threshold are classified as belonging to the anomalous population. The selection of the decision threshold can be optimized according to the desired balance between Type I and

Type II errors.

The two-step process of an autoencoder network is a useful tool for graph anomaly detection. Specifically, an encoder can map nodes $v \in V$ to representational embeddings $z_v \in \mathbb{R}^d$, where z_v corresponds to the embedding for node v. The representation z_v can then be decoded to reconstruct node v's neighbors. This process can be repeated for all nodes V, producing a reconstructed graph [5].

2.9 Graph Autoencoder Architecture

Graph Autoencoders (GAEs) extend autoencoder networks by leveraging graph data. GAEs are a type of deep neural network architecture that map nodes in a graph to a latent feature space and reconstruct graph information from these latent representations. GAEs can be used to generate new graphs or learn network embeddings, but this document will focus on the latter application.

The GAE model created by Kipf et. al. [27] consists of two components: an encoder and decoder. First, the encoder leverages two Graph Convolutional Layers to encode structural and feature information concurrently. The final encoder output is a matrix of node embeddings, where each row represents a node and each column represents a feature in the embedding space. Encoding is described in equation 7:

$$\mathbf{Z} = enc(\mathbf{X}, \mathbf{A}) = f(Gconv(\mathbf{A}, \mathbf{X}; \mathbf{\Theta_1})) \tag{7}$$

where:

- **Z**: Represents the output feature matrix, containing the encoded node features after applying GNN layers;
- **enc(X, A)**: An encoding function that takes the input node feature matrix **X** and the adjacency matrix **A**;
- **Gconv**: A graph convolution operation applied in a GNN layer; and
- Θ_1: A learnable weight matrix associated with the GNN layer.

Relational information between nodes is reconstructed by applying a similarity function to node embeddings, as given in equation 8 [5]. In practice, the most common implementation is the cosine similarity between vectors, or the dot product of pair of nodes u and v [20]. The resulting dot product is input to a sigmoid function to yield a probability value between 0 and 1. When this process is applied to each pair of nodes, the outputs are used to populate the entries of a predicted adjacency matrix.

$$\mathbf{\hat{A}_{u,v}} = dec(\mathbf{z_v}, \mathbf{z_u}) = \sigma(\mathbf{z_v}^T \mathbf{z_u}) \tag{8}$$

where:

- $\mathbf{\hat{A}_{u,v}}$: The predicted or reconstructed edge weight between nodes u and v based on the encoded features;
- **dec(**z_u**,** z_v**)**: A decoding function, which accepts the embeddings of nodes u and v.
- σ: A sigmoid activation function, which maps the input value to a range between 0 and 1; and
- $\mathbf{z_v^T z_u}$: The dot product of the encoded feature vectors of nodes u and v, which measures the similarity between two nodes.

The Graph Autoencoder (GAE) model by Kipf et al. learns to reconstruct the original input graph by comparing the real adjacency matrix A and the predicted adjacency matrix $\hat{A}$. It does so by minimizing the cross-entropy loss between these two matrices [27]. In this context, the presence or absence of an edge in the graph is treated as a binary classification problem. Cross-entropy loss is a suitable choice for this task because it quantifies the dissimilarity between the predicted edge probabilities and the actual presence of edges in the network. By minimizing this loss, the GAE model learns to generate accurate predictions of edge probabilities that closely resemble the true adjacency matrix A [27].

2.9.1 Graph Attention Networks

In many popular GNN architectures, each neighbour has the same importance during the message-passing phase [25]. This can be an undesirable quality; in some tasks, nodes should not all have the same importance. To work around this limitation, Graph Attention Networks (GATs) compute a learned weighted score over the neighbors of each graph node, yielding a score for each edge (u, v) indicating the importance of the features of the neighbor u to the node v [25], [26]. The attention mechanism works by computing a weight for each node in the graph based on its features and the features of its neighbors. These weights are then used to compute a weighted sum of the features of the neighboring nodes, which is combined with the features of the central node to obtain a new representation for that node.

One of the advantages of GATs is that they can learn different weights for each neighbor of a node, depending on its relevance to the task [25]. This allows the model to capture complex patterns in the graph data, such as node importance and community structure.

GATs have demonstrated strong performance in a wide range of graph-based tasks, including node classification, link prediction, and graph classification [21], [23], [25], [26].

2.9.2 GNN Encoding Example

In a traditional GNN, the goal is to generate node embeddings that represent nodes in a graph based on their features. Figure 6 describes a simple two-layer GNN designed to process node features in undirected graphs. With two layers, each node encodes information from neighbours up to two hops away.

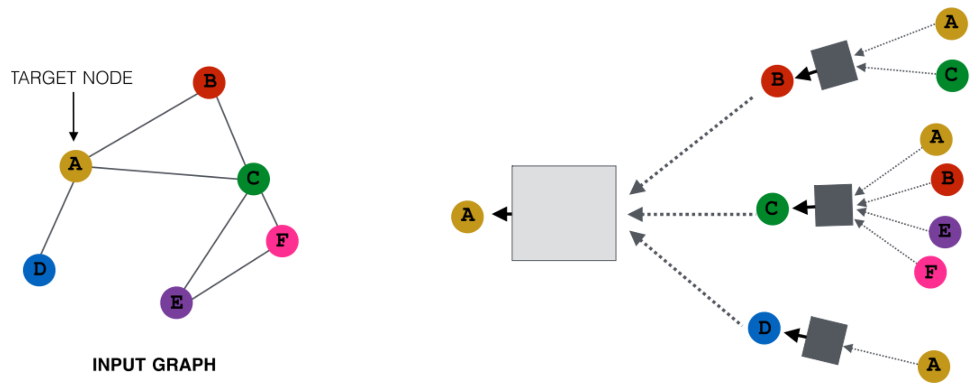


Figure 6: 2-layer GNN applied to an undirected graph, with an unrolled layout.

This model accepts two inputs. First, an array X of node feature vectors. This array has N rows, where N is the number of nodes in the graph. Each row represents the feature vector of a node, capturing its attributes or properties. The second input is usually given as a list of node tuples (pairs of connected nodes). This list represents the connections or relationships between the nodes in the graph and guides the message-passing mechanism within subsequent GNN layers.

The sequence of events can be broken down into four main steps:

1. **Input**: First, a fixed-length feature vector is assigned to each node, based on its attribute data.

2. **First Layer**: The GNN begins the message-passing mechanism in its first layer. Each node sends its feature vector to its neighbors. After all messages are sent, every node aggregates the received messages using an aggregation function. The aggregated data is combined with the node's original features through a neural network layer, followed by a non-linear activation function (e.g., Sigmoid, tanh). The output of this layer is an updated set of node feature vectors.

3. **Second Layer**: The same process is repeated in the second layer of the GNN, using the updated node embeddings from the first layer instead of the original feature vectors.

4. **Output**: After the second layer, the final node embeddings are generated. These embeddings represent nodes in the original graph and encode data from both node features and the local structure around each node.

In summary, a two-layer GNN creates node embeddings by processing and aggregating node features hierarchically. These embeddings capture both feature and relational information, making them valuable for various downstream tasks, such as node classification and link prediction.

2.10 Link Prediction

Link prediction is a task in network analysis that involves predicting the likelihood of a missing link between two nodes in a network based on the network's structure and properties [19]. This approach can be useful in various applications, such as detecting fraudulent transactions in financial networks or identifying potential disease-causing genes in biological networks [21], [28].

Link prediction tasks may be transductive or inductive. Transductive and inductive detection systems differ in their approach to identifying anomalies within graph-structured data. Transductive systems focus on detecting abnormalities within a given graph, and do not generalize to other unseen graphs, making them suitable for single, fixed datasets. In contrast, inductive systems learn general patterns from one or more input graphs, allowing them to identify anomalies in new, unseen graphs, and adapt to dynamic environments.

In the context of link prediction, an autoencoder architecture can be used to detect anomalous links in a network by training the autoencoder on a subset of the network and using it to predict the likelihood of the missing or future links [20]. Anomalous links, which are not well-represented in the normal patterns of the network, will result in high prediction errors [19]. These errors can be used as an indicator of anomalous links in the network. Anomalous links can be identified by comparing the prediction errors with a threshold value. Links with prediction errors above the threshold are classified as anomalous [21].

It is typical to use negative labels in the training data to enable the system to learn the patterns of non-existent edges [5], [21]. Positive labels correspond to pairs of nodes that are connected by an edge in the graph, whereas negative labels correspond to pairs of nodes that are not connected by an edge.

Without the inclusion of negative samples for training, the model will not learn the patterns of non-existent edges, which can lead to overfitting and poor generalization performance [21]. Therefore, it is typical to include negative examples in the training data to enable the model to learn to that the absence of an edge is equally important as the presence of an edge during reconstruction.

2.11 Related Works

2.11.1 EULER: Detecting Network Lateral Movement via Scalable Temporal Graph Link Prediction (2022)

This paper formulates anomalous lateral movement detection as a temporal graph link prediction problem, improving precision over standard anomaly-based intrusion detection techniques [21]. The authors propose EULER, a framework that combines a GNN with a sequence encoder, such as a recurrent neural network (RNN), to capture topological and temporal features of an evolving network.

In this research, a GAE architecture [27] is employed, leveraging a selection of GNN layers, including a GAT [25] encoder. Experimental evaluation on three datasets shows that the EULER framework performs as well or better than state-of-the-art temporal link prediction models and can scale to accommodate larger data. When tested on the Los Alamos National Laboratory (LANL) real-world dataset with approximately 1.6 billion events [29], EULER-based models outperform prior works in both precision and compute time.

Table 2: LANL Dataset Metadata

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

The authors highlight certain challenges in evaluating anomaly detection systems, particularly when real network data lacks fine-grained labeling. One such challenge is that events identified as anomalies might not be part of the labeled dataset, even though they display anomalous behavior. Due to the absence of detailed labeling information, these events are considered false positives. This issue underscores the difficulty in assessing the effectiveness of anomaly detection systems when dealing with real-world network data that lacks comprehensive labeling.

Additionally, although the LANL Dataset is a good representation of real network traffic, the events in the dataset are custom abstractions of authentic corporate network events and only include connections internal to the LANL intranet [29]. Despite these limitations, EULER still sets a strong precedent for unsupervised anomaly detection through link prediction, showcasing its potential in addressing network security challenges.

2.11.2 Unified Graph Embedding-Based Anomalous Edge Detection (2020)

Ouyang introduces a unique end-to-end framework for anomaly detection in graph data, focusing on identifying anomalous edges in static, unattributed graphs [12]. The approach, a "shallow embedding" strategy, considers only relational data and not edge or node features. Conventional shallow embedding methods often depend on hand-crafted features or community-based techniques, which might not be well-suited for anomaly detection tasks as they involve separate models with different objective functions. To address these limitations, the proposed framework jointly learns the graph structural embedding and normal behavior model. The framework consists of three components: a conditional probability distribution used as an objective function to guide normal behavior modeling, an end-to-end neural network architecture parameterizing the probability distribution, and an unsupervised training algorithm. Experiments on six public datasets demonstrate the method's superior accuracy and scalability compared to other shallow graph embedding techniques. The size of these graphs is presented below in Table 3.

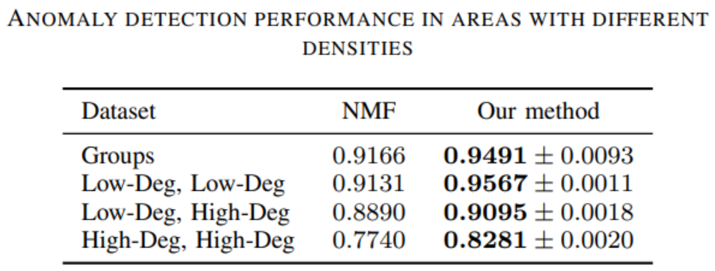
In the experiments, the authors use six datasets, four of which did not have ground truth anomalies present. To quantitatively compare anomaly detection methods against their model, the authors manually injected anomalous edges into these datasets using an injection technique from a similar experiment [30]. The goal of the injection technique was to assess the behaviour of the anomaly detection system in both sparse and dense connection areas in the graph structure

Table 3: Public Datasets for Ouyang et. al.

Dataset	# Vertex	# Edge
UCI Messages	1,899	13,838
arXiv hep-th	9,877	25,998
Digg	30,398	86,312
DBLP	317,080	1,049,866
Email-Eu-core	1,005	16,706
Enron Email	36,692	183,831

[12]. To accomplish the anomaly injection task, the authors sorted nodes by degree and selected the top 25% and bottom 25% nodes as "High-Deg" and "Low-Deg", respectively. Following this random selection, a quantity equal to 1% of the total edges in the network was injected with anomalous edges in various source-destination combinations, yielding three datasets: "Low-Deg, Low-Deg", "Low-Deg, High-Deg", and "High-Deg, High-Deg". Additionally, with ground truth anomalies available in the Email-Eu-core dataset, the authors devised a scenario called "Groups" wherein existing anomalous edges were also randomly inserted between nodes in different groups to simulate malicious emails sent by attackers, as would be typical in a spearphishing campaign [12].

Table 4: Connection Density Comparison Results in the Email-Eu-core Dataset



ANOMALY DETECTION PERFORMANCE IN AREAS WITH DIFFERENT DENSITIES

Dataset	NMF	Our method
Groups	0.9166	**0.9491** $\pm$ 0.0093
Low-Deg, Low-Deg	0.9131	**0.9567** $\pm$ 0.0011
Low-Deg, High-Deg	0.8890	**0.9095** $\pm$ 0.0018
High-Deg, High-Deg	0.7740	**0.8281** $\pm$ 0.0020

Table 4 shows the performance comparison of Ouyang et al.'s approach and Non-negative Matrix Factorization (NMF) using the Area Under the Curve (AUC) classification metric [12]. AUC measures the probability that a randomly chosen positive instance is ranked higher than a randomly chosen negative instance, with higher values indicating better classification performance. NMF is a shallow embedding algorithm that creates a lower-rank approximation of a graph's adjacency matrix to uncover inherent community structures. Anomalies are often associated with predicted edges that differ significantly from the approximated matrix.

The researchers employed anomaly injection techniques to assess the robustness and sensitivity of various methods, including NMF, in detecting anomalous edges. This comparison demonstrated the effectiveness of the strategy proposed by Ouyang et al. in identifying anomalies within static unattributed graphs, outperforming NMF and other techniques. Moreover, the approach reaffirms that a significant amount of classification accuracy can be derived solely from the structure of the graph.

2.11.3 Intention-aware Heterogeneous Graph Attention Networks for Fraud Transactions Detection (2021)

In the commerce domain, deep learning models often employ sequence-based methods that analyze users' behaviors as ordered sequences of actions [31]. However, these existing methods typically treat each transaction as an independent event, overlooking the rich interactions between entities involved in the transactions. To address this limitation, the authors introduce a novel approach called Intention-aware Heterogeneous Graph ATtention networks (IHGAT). This method utilizes a heterogeneous transaction-intention network to capture both transaction and intention-level interactions. Heterogeneous graphs comprise multiple types of nodes and edges, representing various entities and relationships within a complex network. In the context of this study, an intention refers to a sequence of user behaviors that ultimately lead to a transaction.

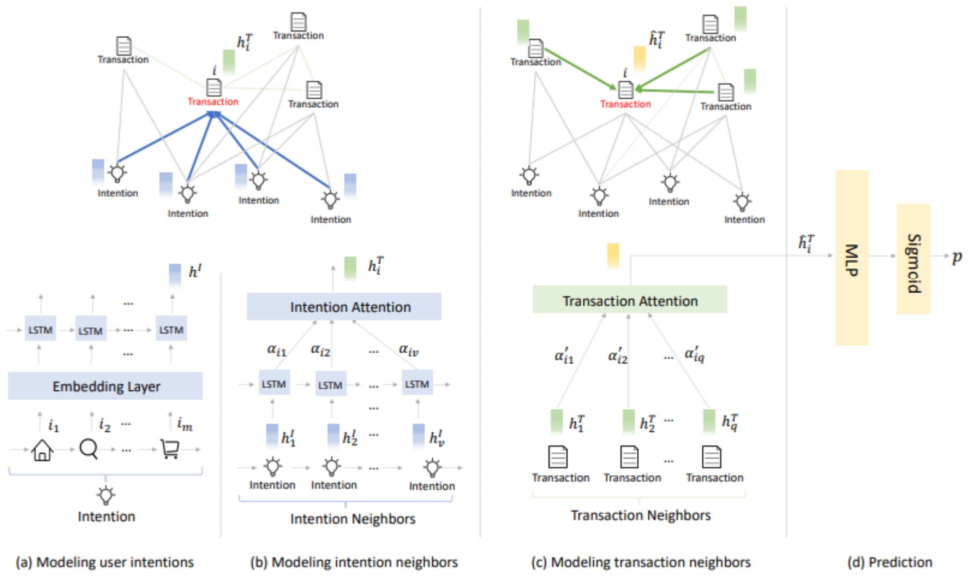
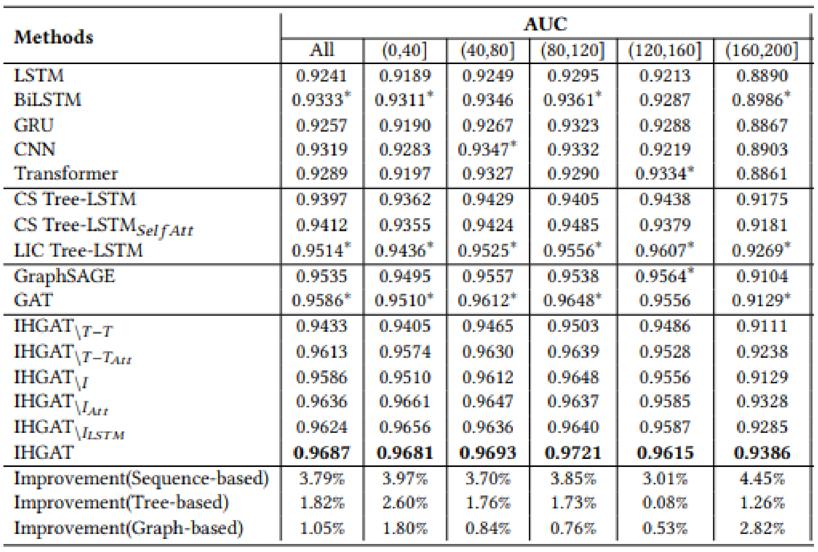


Figure 7: IHGAT model architecture

The authors conceive the input network as transaction nodes and intention nodes. The network also incorporates transaction-transaction edges based on co-occurred attributes and transaction-intention edges created through user behavioural sequence segmentation. The IHGAT model derives a transaction node representation in a two-stage process, visualized in Figure 7 [31]. First, intention neighbors are aggregated using a sequence-based model with an attention mechanism, such as Long Short-Term Memory Networks (LSTM), to capture sequential information. Attention weights are calculated to measure the importance of each intention neighbor and then aggregated to form the current transaction node representation. Next, a multi-head graph attention operation is applied to aggregate transaction neighbors. The final representation of the transaction node is calculated by averaging the attention scores across multiple attention mechanisms. This process enables the model to capture both intention and transaction-level interactions for better fraud transaction detection.

Table 5: IGHAT Methods Comparison Results



Methods	**AUC**					
	All	(0,40]	(40,80]	(80,120]	(120,160]	(160,200]
LSTM	0.9241	0.9189	0.9249	0.9295	0.9213	0.8890
BiLSTM	0.9333*	0.9311*	0.9346	0.9361*	0.9287	0.8986*
GRU	0.9257	0.9190	0.9267	0.9323	0.9288	0.8867
CNN	0.9319	0.9283	0.9347*	0.9332	0.9219	0.8903
Transformer	0.9289	0.9197	0.9327	0.9290	0.9334*	0.8861
CS Tree-LSTM	0.9397	0.9362	0.9429	0.9405	0.9438	0.9175
CS Tree-LSTM$_{SelfAtt}$	0.9412	0.9355	0.9424	0.9485	0.9379	0.9181
LIC Tree-LSTM	0.9514*	0.9436*	0.9525*	0.9556*	0.9607*	0.9269*
GraphSAGE	0.9535	0.9495	0.9557	0.9538	0.9564*	0.9104
GAT	0.9586*	0.9510*	0.9612*	0.9648*	0.9556	0.9129*
IHGAT$_{\backslash T-T}$	0.9433	0.9405	0.9465	0.9503	0.9486	0.9111
IHGAT$_{\backslash T-T_{Att}}$	0.9613	0.9574	0.9630	0.9639	0.9528	0.9238
IHGAT$_{\backslash I}$	0.9586	0.9510	0.9612	0.9648	0.9556	0.9129
IHGAT$_{\backslash I_{Att}}$	0.9636	0.9661	0.9647	0.9637	0.9585	0.9328
IHGAT$_{\backslash I_{LSTM}}$	0.9624	0.9656	0.9636	0.9640	0.9587	0.9285
IHGAT	**0.9687**	**0.9681**	**0.9693**	**0.9721**	**0.9615**	**0.9386**
Improvement(Sequence-based)	3.79%	3.97%	3.70%	3.85%	3.01%	4.45%
Improvement(Tree-based)	1.82%	2.60%	1.76%	1.73%	0.08%	1.26%
Improvement(Graph-based)	1.05%	1.80%	0.84%	0.76%	0.53%	2.82%

The researchers conducted extensive experiments on a large-scale, real-world industrial dataset from an online e-commerce platform to evaluate the effectiveness of the proposed model. The dataset consisted of 1.59 million transactions, with 0.51% identified as fraudulent transactions. Table 5 outlines an evaluation between a variety of sequence, tree, and graph-based methodologies [31]. Entries labeled with an asterisk indicate the best performance out of the three types of baselines. The IHGAT model demonstrated superior performance, with graph-based methods outperforming sequence-based and tree-based models. In a notable follow-up experiment, the model was tested in an online setting on an e-commerce platform, processing a subset of actual transactions for one month. Compared to the e-commerce platform's existing fraud detection system, the IHGAT model achieved a 1.66% improvement in AUC. This research showcases the significant potential of GAT-based anomaly detection models when compared to other robust machine learning solutions.

2.12 Summary

This chapter laid the groundwork for the research by discussing various topics relevant to the study. Endpoint Monitoring Tools, such as System Monitor and Osqeury, are crucial for gathering network data. The chapter also explored Amazon S3, a popular service for large-scale log storage. Word Embeddings and Graphs were examined, in addition to a particular focus on the PageRank algorithm. The MITRE ATT&CK Framework was introduced to offer a standardized understanding of cybersecurity threats.

Unsupervised Learning techniques were investigated, centering on anomaly detection, associated metrics, and challenges with model evaluation. Autoencoders and Graph Autoencoder Architecture were studied within the context of anomaly detection, including a provided example. Furthermore, Graph Neural Networks, such as Neural Message Passing and Graph Attention Networks, were discussed in relation to their application in Link Prediction tasks. Finally, a brief discussion of related works was covered, with particular focus on link prediction in a cybersecurity context, unsupervised evaluation strategies, and graph attention networks. This offered a comprehensive overview of the key concepts and techniques necessary for grasping the fundamentals of an edge-based graph anomaly detection model over large network captures.

3 Methodology and Design

This chapter describes the research methodology and design to develop an anomalous edge detection model. The first phase covers an exploratory investigation of corporate and artificial network environments, summarizing characteristics and identifying patterns and trends in the host-based logs. The second phase is designing and implementing the end-to-end anomaly detection pipeline capable of scoring connections using a probabilistic reconstruction score to yield outlying data. The third phase of research includes the development and application of *ATT&CK*-sourced scenarios using a combination of hybrid and synthetic data to collect metrics related to the quantitative performance of the model.

3.1 Phase 1: Data Extraction, Exploratory Data Analysis, and Feature Engineering

The data extraction, EDA, and feature engineering phase is a crucial step in understanding and analyzing host-based network connection logs in large networks. EDA provides a means to summarize and visualize insights, identify patterns and trends, and identify potential issues or limitations that may affect downstream tasks. In the context of massive log data files, it also provides an opportunity to filter superfluous data to greatly reduce the processing overhead of the pipeline. The EDA phase informs the development of subsequent features which are used during the graph encoding process.

3.1.1 Data Extraction

In the initial stage of the experimentation process, a solution is needed for querying large-scale data stores containing unstructured log data. Therefore, a solution must be designed and implemented to effectively extract the relevant data needed for analysis. This solution should satisfy the following requirements:

- Enable fine-grained control over the time period of logs being analyzed, which means allowing the selection of specific time windows or intervals for data extraction; and
- Process vast amounts of data efficiently and rapidly to prevent it from becoming the primary bottleneck during experimentation.

To meet these requirements, the log extraction tool must be capable of retrieving only the fields of interest to minimize the amount of unnecessary data that needs processing. Developing and implementing this solution is crucial for the success of the Exploratory Data Analysis (EDA) process.

3.1.2 Exploratory Data Analysis

The data exploration sub-phase involves constructing the graph representation of the target computer network, which allows for a better understanding of the network structure and facilitates the identification of patterns and relationships among nodes. Analyzing the data in both tabular and graph-based formats using intermediate data formats helps to gain deeper insights into normal host communication patterns and behaviors, as well as to summarize relevant characteristics.

In this process, EDA is performed across multiple networks varying in size, ranging from academic to production transnational networks in scale. Using features available in Sysmon Event ID 3, source and destination address/port features are employed to attribute nodes in the graph. Osquery is utilized to collect hardware interface details, which serve to enhance the network features further.

Constructing the graph representation enables the extraction of valuable insights from the network topology, such as identifying central nodes, detecting communities, or revealing potential vulnerabilities. This graphical representation aids in the development of effective anomaly detection models by providing a comprehensive view of the network's structure and its underlying communication patterns.

The inclusion of relational-based analysis allows a wide range of new metrics to be derived from the data. A key step in this phase is filtering; Sysmon logs collected from a wide time window on the target network can total hundreds of millions of records and communications between hundreds of hosts. To reduce the computational overhead incurred by GNN-based computations, events related to web-browser connections and isolated sub-communities are removed from the data.

3.1.3 Feature Engineering

Feature engineering is a critical steps following the EDA process, as it ensures that relevant and informative features are used to build the anomaly detection model. This step involves transforming existing features, or deriving new ones.

Based on the insights gained during the EDA process, new features can be created to better capture the underlying patterns and relationships within the network. These features may include aggregated or derived statistics, such as importance measures like PageRank, or connectivity metrics like the in/out degree of a node. Additionally, domain-specific knowledge can be incorporated to create features that reflect the unique aspects of the network, such as the administrative credentials or protocol-specific attributes.

Some features may require transformation to improve their usefulness in the model. Techniques such as frequency encoding or categorical variable encoding can be applied to ensure that features are weighted appropriately and compatible with the chosen modeling approach.

After engineering and transforming features, it is essential to select a subset of features that contribute most to the model's performance. By incorporating the results from the EDA process, feature engineering and selection steps ensure that relevant features are used to build the anomaly detection model. These steps ultimately contribute to improved model performance and generalization.

3.2 Phase 2: Model Development

The purpose of phase two is to develop an unsupervised model capable of processing the static, attributed network created in phase one. This model will be transductive, scoring relational anomalies based on their probability of occurrence during reconstruction, as seen in other anomalous edge detection models [6], [12]. The strategy is to use graph representations with an autoencoder architecture to determine which edges are anomalous. This approach is effective because outlying connections are sparse, and will be mapped to uncertain latent representations [5]. In turn, this produces a higher reconstruction error during the decoding process [19].

The model development stage involves constructing an end-to-end anomaly detection pipeline using an anomalous edge detection method based on a GAE link prediction architecture [27]. The performance is enhanced by incorporating GAT layers during the encoding process [26]. A significant challenge is handling the large network sizes; to address this, the GAT-GAE model is specifically designed for large graphs, providing an effective solution for identifying anomalous edges. Hyperparameter tuning is conducted during the development of the GAT-GAE model to further optimize performance.

3.2.1 Loss Function

The choice of loss function is a critical decision when measuring the difference between the predicted output of the model and the true output, as it provides a signal to update the model's parameters during training. Link prediction can be formulated as a probability estimation, where potential links are associated with existence probabilities. For measuring the error in the reconstruction of a given sample of links, a binary cross-entropy (BCE) loss with an applied sigmoid function is selected.

In the context of graph reconstruction, the task is to predict the presence or absence of edges between nodes, which can be framed as a binary classification problem: either an edge exists (1) or it does not (0). BCE loss measures the dissimilarity between the predicted probabilities of edge existence and the actual presence of edges in the adjacency matrix of the graph. By minimizing BCE loss during training, the model learns to generate accurate predictions for edge existence. This loss function is particularly useful when the presence or absence of edges can be interpreted as probabilities yielded by the sigmoid function, as BCE loss accounts for the uncertainty in the predictions and penalizes wrong predictions with high confidence.

The BCE with Logit function is described in equation 9 [32]:

$$l(x, y) = L = \{l_1, ..., l_N\}^T, l_n = -w_n[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n))] \quad (9)$$

where:

- L is the BCE loss value;
- N is the number of training samples;
- w_n is the weight assigned to the example as a scalar value;
- x_n is the raw output of the model;
- y_n is the binary label for an example; and
- σ is the sigmoid function, mapping logits to a probability values between 0 and 1.

This loss accepts a tensor of logits and binary labels, given as x and y, respectively. The term w_n is a weight assigned to each training example. and computes the binary cross-entropy loss between the logits and labels. A binary selection of labels is created by sampling positive and negative edges within the network structure.

3.2.2 Optimizer

The optimization algorithm plays a crucial role in a machine learning model, as it is responsible for minimizing the loss function during the training process. In this case, a standard implementation of the Adam optimizer [33] was selected, which is an adaptive stochastic gradient-descent optimizer.

Adam is a widely-used optimization algorithm, valued for its adaptive learning rate and low memory requirements. It offers an adjustable learning rate hyperparameter for the model, controlling the step size at which the model's weights are updated during the training process. This flexibility allows for more efficient and effective training, as the learning rate can be adjusted to optimize the model's performance.

3.2.3 Detection Threshold

With labeled data available through scenario injection, automating the determination of the detection threshold for identifying anomalies proved valuable in obtaining quantitative metrics. The optimal cutoff threshold for edge likelihood scores was formulated as an optimization problem.

By leveraging the known labels in the data, the optimization algorithm adjusts the anomaly threshold to maximize the F_1 score, which balances precision and recall. This approach enables a more efficient and accurate identification of anomalies in the network by selecting the detection threshold that achieves the best trade-off between recall and precision, leading to the highest possible F1-score.

3.3 Phase 3: Model Evaluation

During this phase, model performance is evaluated by inserting a known quantity of synthetic data into the corporate network capture. Four distinct scenarios encompass a range of anomalous network behaviors, generating metrics that assess the model's performance under various conditions. By analyzing the model's ability to detect these synthetic anomalies, its effectiveness in identifying different types of anomalous behaviors can be quantified and compared.

3.3.1 Insertion of Data for Model Evaluation

Evaluating unsupervised models, particularly at scale, presents a significant challenge due to the absence of ground truth anomalies in the networks under analysis. Obtaining ground truth anomalies is a well-known challenge in the field [21], [12]. To enable quantitative evaluation of the anomaly detection model, synthetic data is inserted into the networks using anomaly injection.

Anomaly injection involves manually introducing synthetic anomalous edges into the original dataset. These edges represent connections that deviate from the typical patterns observed in the network. By injecting these synthetic anomalies, a pseudo-ground truth is created, which can be used for model evaluation and comparison.

This solution is commonly adopted in literature [6] [12] [21]; however, it does not address the presence of fine-grained anomalous behaviors already present in the data. These unidentified events may be detected as false positives, potentially negatively affecting the metrics associated with the model and impacting the overall evaluation of its performance.

3.3.2 Model Performance Metrics

In the model evaluation sub-phase, various performance metrics are used to quantify the effectiveness of the anomaly detection system. These metrics are derived from four binary classification outcomes: True Positives (TP), False Positives (FP), False Negatives (FN), and True Negatives (TN).

Moreover, the concept of reconstruction error is critical in the context of anomalous edge detection. Reconstruction error can be viewed as an inverse indicator of an edge's presence probability within the network. Specifically, a high reconstruction error suggests a low probability of an edge being present in the original network. Consequently, edges with high reconstruction errors are more likely to be anomalies, as they deviate from the expected network patterns.

These performance metrics, along with the understanding of the relationship between reconstruction error and edge presence probability, enable an effective measurement of the anomaly detection system's performance.

3.4 Scenarios

In this experiment, custom scenarios are used as a conceptual framework that defines how hybrid (real labeled events) or synthetic (artificial) edges are connected to source and destination nodes within the actual network data. These scenarios provide a structured way of introducing edges into the network, allowing for a controlled evaluation of the model's ability to detect anomalies.

Four scenarios are implemented: `NMAP`, `MoziDDoS`, `ExternalToEndpoint`, and `EndpointToExternal`.

3.4.1 NMAP

The NMAP scenario is designed to emulate the network traffic behaviour generated by the NMAP tool. NMAP, or Network Mapper, is a widely-used open-source tool for network discovery and security auditing. It scans hosts and services on a network by sending packets and analyzing responses to identify active hosts, open ports, and running services.

Regarding host-based connections, NMAP generates a significant amount of traffic by sending various types of packets to target hosts. These scans may create a noticeable pattern in the network traffic as NMAP probes multiple ports on the target hosts, often in a sequential or near-sequential manner.

NMAP fits into the MITRE ATT&CK framework as a tool that can be used during the "Reconnaissance" stage of the attack lifecycle. Attackers may use NMAP to gather information about the target network, identify vulnerabilities, and plan their attacks accordingly. In the MITRE ATT&CK framework, NMAP is commonly associated with technique T1046 (Network Service Scanning).

In the NMAP scenario, synthetic edges are inserted between nodes to mimic NMAP scanning the immediate subnet of nodes artificially. This process results in the addition of numerous edges to the graph, simulating the network traffic generated by NMAP scans. Moreover, node features are modified to indicate that a range of ports is being scanned, further replicating the behaviour of NMAP in the network.

By incorporating the NMAP scenario into the graph anomaly detection system, the effectiveness of the system can be evaluated in the context of detecting network traffic patterns associated with NMAP scanning, which is a common tool used by adversaries during the reconnaissance phase of an attack.

3.4.2 MoziDDoS

The MoziDDoS scenario simulates the behaviour of the Mozi IoT botnet conducting a Distributed Denial of Service (DDoS) attack [34]. Mozi is an IoT botnet that targets poorly secured IoT devices, such as Smart TVs, DVRs, and IP cameras, exploiting known vulnerabilities. Once compromised, these devices can be remotely controlled by the attacker to conduct various attacks, with DDoS attacks being one of the most common use cases.

Botnets like Mozi are often used to execute DDoS attacks by overwhelming a target system or service with an excessive volume of traffic. This influx of traffic can lead to the target system becoming unresponsive or unavailable to legitimate users.

In the context of the MITRE ATT&CK framework, DDoS attacks fit within the "Impact" tactic, specifically the T1498 (Network Denial of Service) technique. This tactic aims to disrupt, degrade, or deny the availability of target systems, services, or network resources.

In the MoziDDoS scenario, synthetic nodes representing real malicious IP addresses are incorporated into the graph structure. User endpoints within the corporate network initially establish connections with these new malicious IPs, creating both request and response edges between the user endpoint and the new node. Following this, a large-scale DDoS attack is simulated, targeting a random external IP address that already exists within the network structure. This mimics the command and control aspect of a DDoS operation. As a result, numerous edges are added to the network, reflecting the substantial volume of traffic generated during a DDoS attack.

Incorporating the MoziDDoS scenario into the graph anomaly detection system enables the evaluation of its effectiveness in detecting network traffic patterns associated with DDoS attacks conducted by IoT botnets like Mozi. This evaluation is crucial due to the prevalence of DDoS attacks in the current cybersecurity landscape, emphasizing the importance of early detection for mitigating their impact.

3.4.3 ExternalToEndpoint

The ExternalToEndpoint scenario is designed to simulate real anomalous connections between external (public) addresses and user workstations within a corporate network. In this scenario, hybrid edges are inserted into the graph as incoming connections, representing potential security threats from external sources to internal endpoints.

In the context of the MITRE ATT&CK framework, the ExternalToEndpoint scenario can be associated with several tactics and techniques, including:

1. Lateral Movement: This tactic involves an attacker moving through a network after gaining initial access to expand their foothold. Techniques such as T1021 (Remote Services) or T1570 (Lateral Tool Transfer) can be linked to this scenario, as they involve accessing or transferring tools to internal endpoints from external sources.

2. Command and Control: This tactic pertains to how an attacker communicates with compromised systems within a network to execute their commands. Techniques such as T1562 (Impair Defenses) are relevant to the ExternalToEndpoint scenario, as they involve incoming connections to establish or maintain control over internal systems.

3. Infiltration: This tactic emphasizes the unauthorized transfer of data into a target network. The ExternalToEndpoint scenario could be associated

with techniques such as T1105 (Infiltration of Files or Information), as they involve sending data from external addresses to internal endpoints within a network.

By integrating the ExternalToEndpoint scenario, the graph anomaly detection system's effectiveness in identifying suspicious connections between external addresses and user workstations can be evaluated. This is crucial for preventing breaches or data exfiltration in corporate networks.

3.4.4 EndpointToExternal

The EndpointToExternal scenario is a similar conception to the ExternalToEndpoint scenario, designed to simulate connections initiated by user workstations within a corporate network to potentially dangerous public IPs. In contrast to the ExternalToEndpoint scenario, where the connections were incoming, the connections in this scenario are outgoing, representing potential security risks associated with internal endpoints connecting to external sources.

In the context of the MITRE ATT&CK framework, the EndpointToExternal scenario may be associated with similar tactics to ExternalToEndpoint, such as file exfiltration or command and control. In the latter case, techniques such as T1071 (Application Layer Protocol) or T1090 (Proxy) may be related to the EndpointToExternal scenario, as they involve using outgoing connections to establish or maintain control over external systems.

The non-trivial change in direction from incoming to outgoing connections in the EndpointToExternal scenario affects the order of message passing within the graph anomaly detection system. This alteration emphasizes the importance of evaluating the system's effectiveness in detecting potential security risks associated with internal endpoints connecting to external sources. Identifying and mitigating such connections is crucial for preventing potential breaches, data exfiltration, or unauthorized control over internal systems in a corporate network environment.

In summary, scenarios are utilized to assess the efficacy of a graph anomaly detection system that relies on anomalous edge detection. By defining how hybrid or synthetic edges are connected to nodes in real network data and incorporating various parameters to refine the graph, scenarios play a crucial role in evaluating the performance of the detection system under different conditions.

3.5 Summary

In this chapter, three research phases are outlined to achieve the project's aim. The first phase involves data extraction and exploration from the target network. This phase concludes when the target network's characteristics are sufficiently described through data exploratory analysis. The second phase focuses on model development, during which the message passing architecture and loss functions are refined to determine the reconstruction loss of connections that exceed a predetermined threshold. This phase ends when the model successfully scores edges and detects connections considered anomalous within the broader network context. Lastly, the third phase comprises model validation against synthetic and hybrid edges. These data points are used to gather performance metrics assess the quantitative performance of the model in detail.

4 Results

This chapter outlines the development and assessment of an anomaly detection system aimed at identifying irregular connections within computer networks. The chapter begins with an overview of the experimental environment employed for conducting this research. Subsequently, it delves into the findings obtained from the exploratory data analysis. The chapter then proceeds to describe the essential feature engineering techniques utilized in the experiment. In the Model Development section, the design and implementation of the GAT-GAE architecture are discussed. Finally, the Model Evaluation phase examines the system's verification and the model's validation against various scenarios, including a smaller, artificially constructed computer network.

4.1 Experimental Design

Figure 8 displays a high-level architectural design for the complete anomaly detection pipeline. Key components of the architecture are elaborated upon in the subsequent sections.

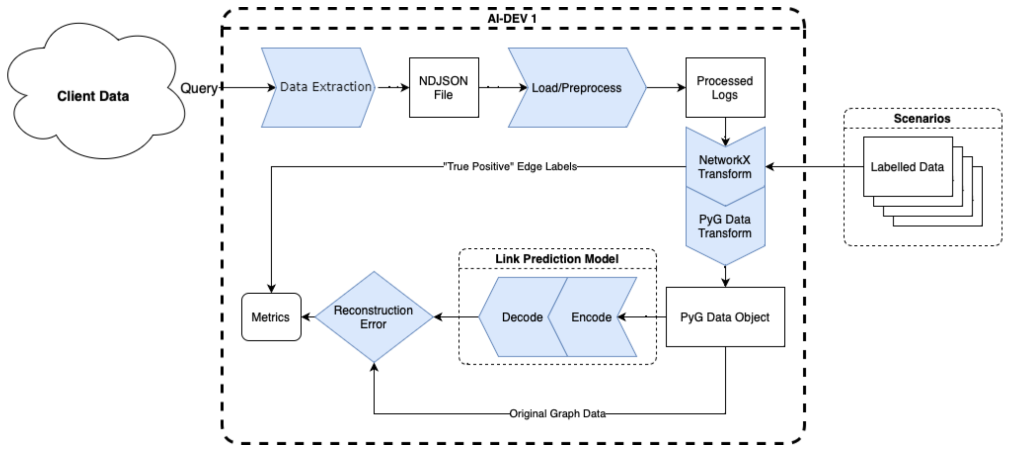


Figure 8: Experimental Architecture

4.2 Experimental Environment

The entire research process was carried out in a high-performance computing environment, which had privileged access to four large corporate networks. This included a local S3-API compliant archive of host log data and active remote endpoint monitoring tools.

The entire pre-processing, training, and evaluation process was powered by a state-of-the-art Nvidia RTX A6000 GPU. Under the largest processing loads, the pipeline demanded up to 900 gigabytes of memory, with the most significant consumption occurring during parallel conversion to network file formats described shortly. This robust computational setup enabled efficient handling and analysis of the vast data sets involved in the experiment, ensuring timely results.

Log records obtained from these corporate networks for anomaly detection purposes are subject to a confidentiality disclosure agreement and cannot be shared publicly. To prevent the disclosure of any identifiable corporate information, several data visualizations and other findings from the exploratory analysis are anonymized.

Lastly, log data was obtained from an artificial network environment (ANE). This data capture spanned approximately twenty hours, during which automated network user activity occurred in the background. Simultaneously, an attack campaign mimicking the behaviors of APT-29 was conducted. This simulated attack involved activities such as file download, the establishment of a long-running command and control (C2) channel, and file exfiltration [35]. The artificial network environment provided a unique dataset for analyzing the performance of the anomaly detection model in a controlled environment.

4.3 Phase 1: Data Extraction, Exploratory Data Analysis, and Feature Engineering

This phase encompasses data generation and extraction from log data, transformation of the tabular data into NetworkX format, initial feature selection and processing based on Sysmon Event ID 3, and exploratory data analysis (EDA) to gain a deeper understanding of the data. The data was collected using a custom program developed specifically for this research, designed to efficiently extract relevant data from an S3-compliant log storage system. A 12-hour window across local working hours for each network (approximately 0600h - 1800h EST) was selected for high user activity. Four corporate networks were analyzed across a 12-hour window, except for the APT-29 artificial network environment, which was a continuous 20-hour capture.

The insights derived from the EDA process serve as the foundation for feature engineering and model development. The focus of this phase is on extracting valuable information from the data and transforming it into a suitable format for the subsequent modeling process. By carefully selecting features and conducting a thorough analysis, this phase ensures that the data is well-prepared for the anomaly detection model, ultimately contributing to the model's overall performance and effectiveness.

4.3.1 Data Generation

Sysmon drivers on the target network are configured for live host monitoring and required no modification for this research. A log aggregation pipeline continually ingested events from remote endpoints and wrote to a remote archive. Log records were stored as compressed NDJSON files.

Osquery returns selected features from host operating systems using SQL [9]. Queries were executed against endpoints in real-time. The data returned from queries was returned as a standard JSON list array [36].

The period of data extraction between the tools coincides in order to accurately attribute nodes with features extracted from Osquery and Sysmon. Fine-grained control over periods of monitoring is relatively straightforward by performing a *JOIN* on features common to both datasets [8], [9].

4.3.2 Data Extraction

Access to a large quantity of enterprise network logs was facilitated through a remote archival server cluster via a research partner. The log storage server is an S3-compliant fault-tolerant distributed object storage system that is suitable for storing large amounts of unstructured logs. Log data is stored per-client as newline-delimited JSON files, typically sized around 100Mb per log file uncompressed. One hour of log data typically produces between 700 and 1200 log files, dependent on the client and working hour. Summarily, 70-120GB of log data is scanned for each hour of analysis. Consequently, it is intractable to download all logs in a fixed time period. However, S3-compliant systems provide a mechanism to filter the data server-side by leveraging the S3-SELECT API.

S3-SELECT is a client programming interface that allows various file formats to be queried directly using a SQL-like syntax. Using prepared statements, a query is dynamically generated to retrieve log data that fits within user-defined conditionals. The query is concurrently applied against each log file with the user-defined start and end time period. An example of a query literal is provided in Listing 1.

```sql
SELECT s.computer_name, s.event_data.image, s.event_data.user ...
FROM S3Object s
WHERE s.event_id in (3) AND s.source_name in 'sysmon')
```

Listing 1: S3-SELECT example

The response JSON is serialized into a single compressed file, producing a much smaller file that can be immediately loaded into memory for downstream analysis.

The tooling developed for this research effectively distributed queries in parallel; however, a performance bottleneck was encountered in the pipeline due to the serialization of response queries into a single file. This bottleneck resulted from the acquisition of a mutual exclusion lock, which limited the speedup gains. To enhance scalability, an alternative solution could involve writing data to temporary intermediate files and consolidating the results as a final step. It is important to note that during the experimentation phase of this research, a readily available solution to this problem did not exist, highlighting the significance of the custom tooling created for this research.

4.3.3 Data Preprocessing

The data was then read into a tabular in-memory format via the Python `Pandas` library by reading the filtered NDJSON from disk [37]. To accommodate the size of the file in working memory, the data was processed in chunks. Preprocessing steps included:

1. File format conversion to Parquet, a space-optimized columnar file format that significantly reduces the memory footprint and greatly speeds up reads from disk [38];

2. Filtering: records from outside the window of analysis or otherwise containing NULL values were droppped from the dataset. In total, these out-of-band records comprised less than 1% of the total dataset;

3. Image editing: The absolute paths of each image was removed, yielding only the executable name; and

4. Feature engineering: an *IsAdmin* feature as assigned to each record, based on whether or not the USER feature contained NT AUTHORITY or other client-specific strings such as "<Corporate Network>-ADMIN".

Upon completion of this step, the data exists in two formats: on disk, as a space-efficient Parquet file, which is compressed for optimal storage, and an in-memory representation as a Pandas DataFrame object. This dual representation ensures efficient data handling and processing throughout the analysis.

4.3.4 NetworkX Transformation

The next intermediate representation of the data involves transforming the log data into a graph data structure. This is supported by the Python programming library NetworkX [39]. The advantage of using NetworkX graphs as an intermediate representation is that the library has a wide range of graph methods available, including *PageRank*, eigenvector centrality, and other connectivity metrics.

Composing a DataFrame into a graph involves a complete scan of the data. Each record provides additional context about the structure of the graph, which requires updating relevant node and edge features via the *update_edge* and *update_node* functions.

The algorithm is described in Listing 2.

```python
import networkx as nx
import pandas as pd

def to_graph(
    df: pd.DataFrame,
    top_k_images: Optional[dict] = None
) -> nx.MultiDiGraph:
    """Convert DataFrame to a multi-relational directed graph."""
    graph = nx.MultiDiGraph()

    for row in df.itertuples():
        update_edge(graph, row, top_k_images)
        update_node(graph, row)

    return graph
```

Listing 2: NetworkX Transformation

By partitioning the input argument `df: pd.DataFrame`, the algorithm described in Listing 2 could be executed in parallel with the resulting output collected as a list of sub-graphs. A performant strategy to combine the sub-graphs is described in Appendix A.1.2.

```python
import functools

def reduce(graphs: list[nx.MultiDiGraph]) -> nx.MultiDiGraph:
    """Compose a single graph from a list of sub-graphs."""
    return functools.reduce(merge, graphs)
```

Listing 3: Sub-Graph Composition

4.3.5 Feature Selection and Engineering

Almost all Sysmon Event ID 3 features are characterized by high degrees of non-ordinal cardinality. Consequently, few features are immediately usable without further processing. This experiment used manual feature selection based on domain knowledge to derive meaningful context from the log data.

Table 6: Selected *Sysmon* Event ID 3 Features

Field	Description
DestinationIsIpv6	Indicates if the destination is an IPv6 address
DestinationIp	Destination IP address
DestinationHostname	DNS name of the contacted host
DestinationPort	Destination port number
DestinationPortName	Name of the destination port in use
Image	The file path of the process that made the network connection
Initiated	Indicates whether the process initiated the TCP connection
ProcessGuid	The GUID of the process that made the network connection
ProcessId	PID of the process that made the connection
Protocol	Protocol used for the network connection (e.g. TCP, UDP)
SourceIsIpv6	Indicates if the source IP is an IPv6 address
SourceIp	Originating IP address
SourceHostname	DNS name of the host that made the network connection
SourcePort	Source port number
SourcePortName	Name of the source port in use
User	Name of the account that made the network connection
UtcTime	Time in UTC when event was created

Table 6 features highlighted in green were selected for the model because they provide valuable information for characterizing and understanding network connections. The selected features are categorized as follows:

Address and Port Information: Features like DestinationIp, DestinationPort, SourceIp, and SourcePort are essential for identifying the endpoints of network connections. These features allow the model to differentiate between various connections and capture patterns in the communication between hosts and services on the network.

Process Information: The Image feature indicates the file path of the process involved in the network connection, either as the initiator or the handler. This information sheds light on the applications and services responsible for the connections, playing a critical role in identifying anomalous behaviours.

Connection Details: The Initiated and Protocol features capture whether the process initiated the TCP connection and the protocol used for the connection, respectively. These features provide insight into the direction of the connections and can help identify unusual or suspicious behavior in the network.

User Information: The User feature captures the name of the account that made the network connection. This information helps identify the users responsible for the connections and can be useful in detecting anomalous behaviours associated with accounts.

Timestamp: The UtcTime feature records the time in UTC when the event was created. This information enables the model to window time appropriately, and can be leveraged during EDA to examine access patterns over time.

These selected features provide a comprehensive view of network connections and their associated processes, users, and timestamps. By incorporating this information into the model, it can better identify patterns, relationships, and anomalies in the network activity.

4.3.6 Exploratory Data Analysis

The aim of EDA is to collect insights into the structure and features of the corporate networks, particularly in filtering superfluous data to reduce the processing overhead of the pipeline. This consisted of analysis steps on both the tabular data, and relational metrics on the directed graph. During the EDA phase of the research, several important patterns and findings emerged concerning scale, time processes, users, and connectivity. The following sections provide more detail on each of these aspects.

4.3.6.1 Network Scale

Table 7: 12h Corporate and 20h Artificial Network Capture Scale

Client	# Records	# Nodes	# Edges	Business Sector
A	92,793,807	33,459	420,935	Industrial
B	136,201,466	18,197	247,396	Industrial
C	124,169,703	330,585	981,566	Natural Resources
D	45,097,758	14,526	140,961	Transit
ANE	48,630	437	256	-

1. **Scale.** As seen in Table 7, the corporate networks were exceptionally large, both in terms of the number of users, and the number of interconnected devices.

2. **Applications.** The quantity and types of applications used in the corporate networks featured many business-related applications attributed to the

broader scope of operations, often exceeding 3000 unique programs during the period of analysis

3. **Usage patterns.** The user behaviour in a corporate network is far more diverse than in a simulated network environment, where automated processes generate traffic. Corporate network traffic reflects the habits and needs of a heterogeneous group of users, but also extends beyond human interaction, including IoT devices and smart devices within the network. In the simulated network environment, the traffic generated is not representative of the real-world user behaviour represented in the four corporate datasets.

Special attention must be given to the size of the client networks in this study. As illustrated in Table 7, even though the feature set was limited to Sysmon Event ID 3 records, the resulting graphs were still remarkably large. Throughout the experimentation process, a diverse range of monitored networks were analyzed, spanning from academic institutions to national-level networks. The distribution of public-to-private IP addresses can be seen in Figure 9. In general, the distribution is fairly balanced, with the notable exception of Network C, which has a significantly broader internal monitoring coverage.

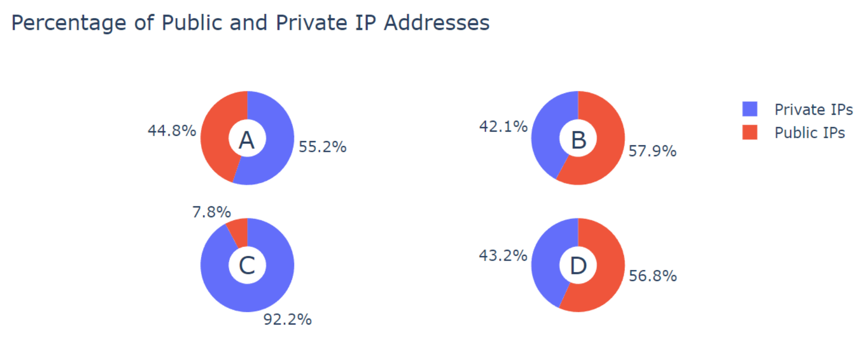


Figure 9: Distribution of public versus private IPs in corporate networks

As described in the preprocessing transformation, the number of edges in the networks was significantly reduced by aggregating and weighting edges along *(Source IP, Destination IP, Image)* pairs.

4.3.7 Graph Visualization and Analysis

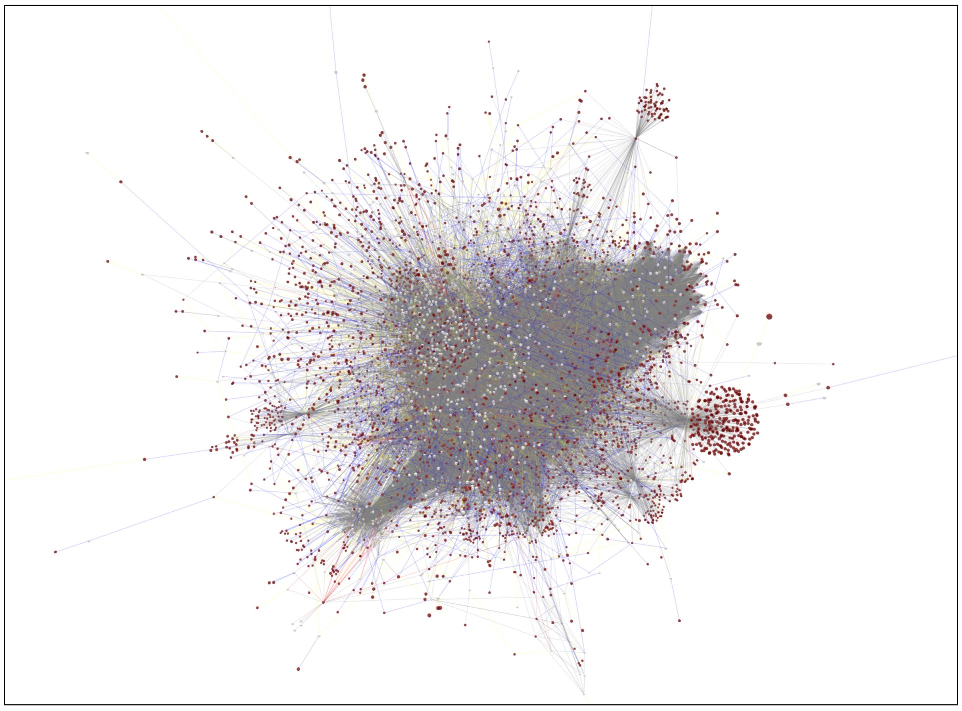


Figure 10: One-Hour Sampled Client D - Network Graph

Attempting to visualize enterprise-scale networks presents challenges. Figure 10 demonstrates that even filtered one-hour network samples suffer from node occlusion and edge crossings that almost completely obfuscate the inter-node space. This results in what has affectionately presented in graph literature as a *hairball* topology [40]. Still, focusing attention on particular sub-structures presents insight into network behaviours. Note that white nodes signify internal addresses, whereas red nodes highlight external addresses. There is no significance to the colouring of edges in the following visualizations.

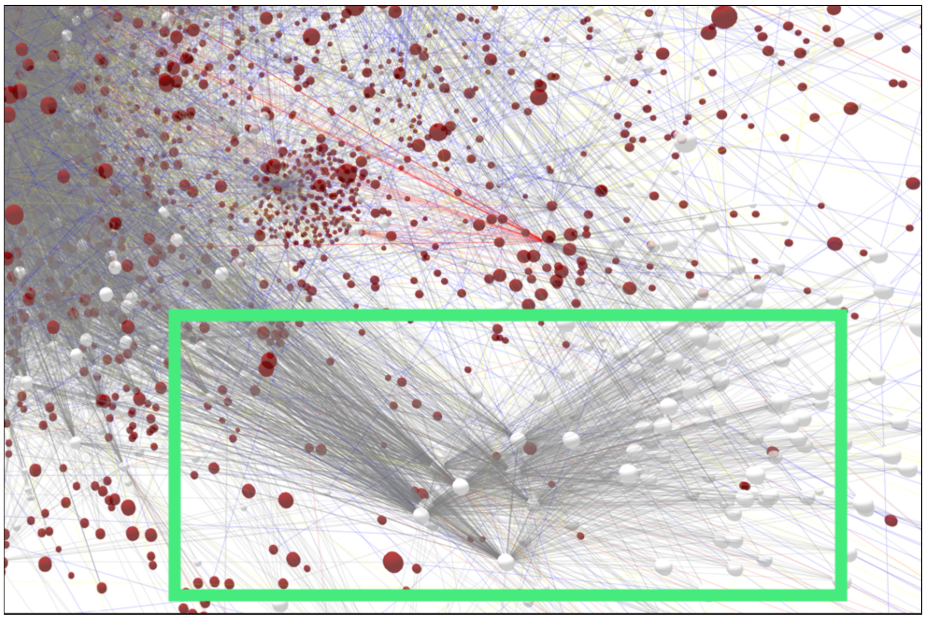


Figure 11: Client D - Active Directory Server Cluster

Figure 11 presents a closer look at a community within the visualization given by Figure 10, and shows several well-connected internal addresses. Further analysis of the edges associates this intranetwork connection density with active directory servers, specifically *lsass.exe* and *netlogon.exe* connections.

In Figure 12, an internal address is connected to multiple external addresses. Analysis revealed that the external addresses were associated with AWS Cloud-Compute EC2 instances. These instances provide computing resources to companies on demand. It is important to consider these behaviours to yield high link presence probabilities that exhibit similar connection behaviours. Consequently, a representative node embedding should reflect the connection patterns seen in EC2 instances to effectively capture the underlying network structure.

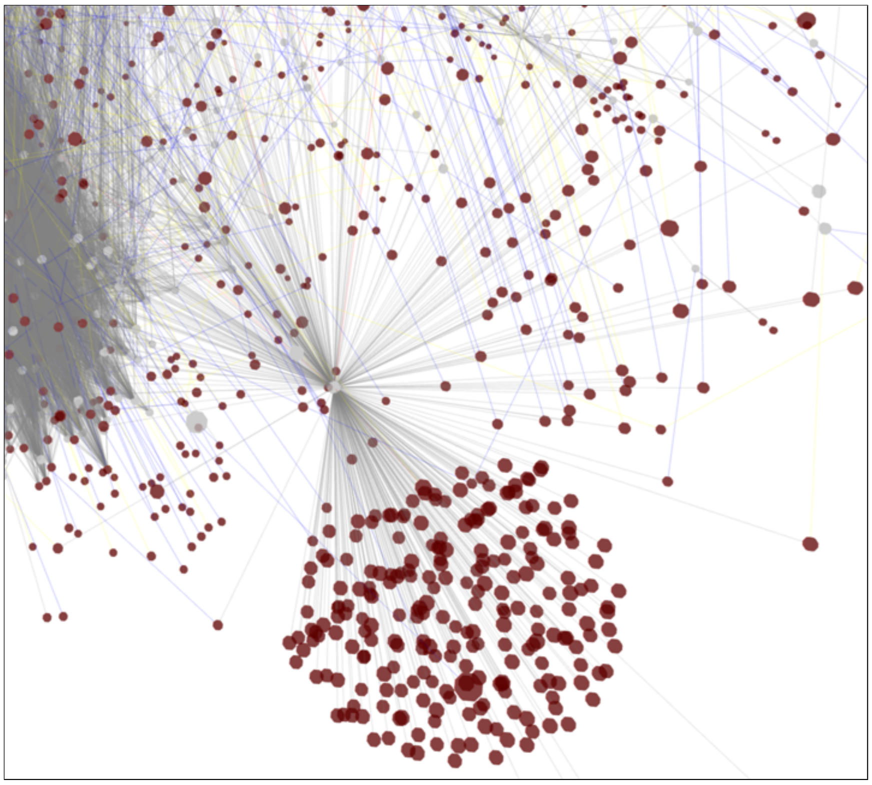


Figure 12: Client D - Cloud Computing Instances

Figure 13 is a graph visualization of the artificial network environment with the node and link sizes scaled by the total links and number of connections, respectively. The centralized, well-connected nodes are user workstations interconnected by various networked processes. Each user workstation also features a prominent hub-and-spoke topology generated by bidirectional background services traffic .

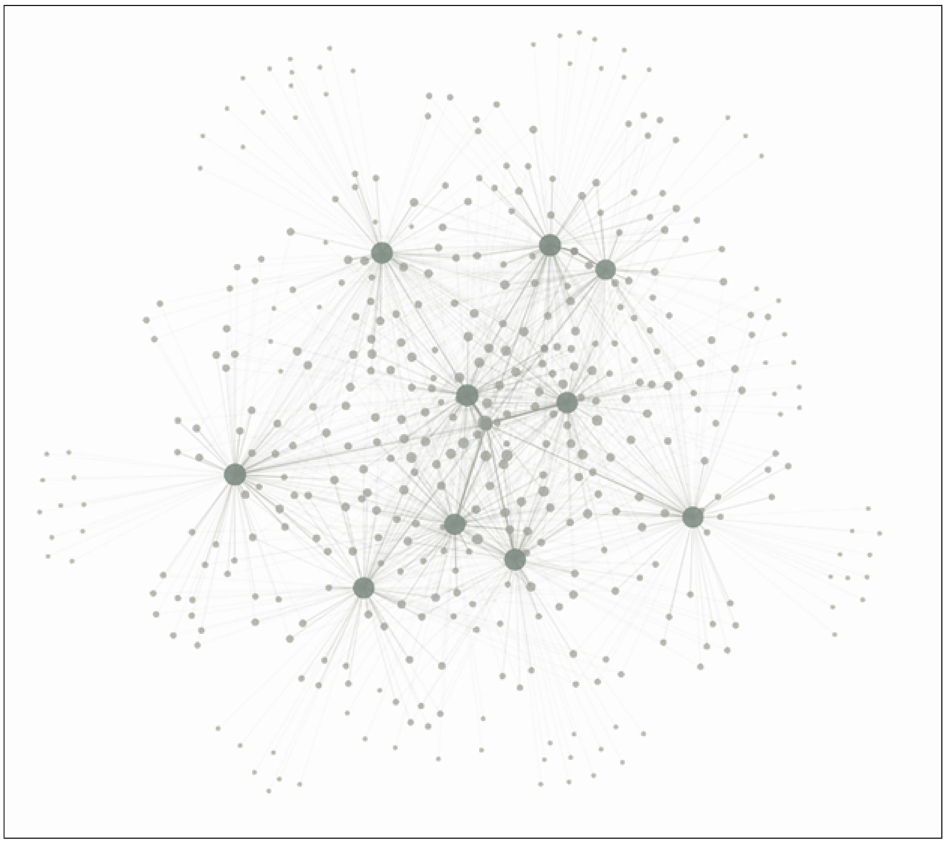


Figure 13: Artificial Network Environment Graph

Despite the limited scope of the network and the stark contrast to larger corporate environments, it remains non-trivial to visually identify any structural abnormalities between nodes.

4.3.7.1 Time Period

The duration of the analysis, referring to the continuous capture of log data, was determined by taking into account several factors. First, larger time windows effectively capture a wider range of potential anomalous behavior. Second, larger samples provide more contextual information, allowing for further reinforcement of network connection patterns. Finally, larger samples ranging from 18-24 hours can significantly increase disk usage, network bandwidth, and processing overhead, limiting the ability to rapidly iterate on the datasets. In summary, selecting the analysis duration is a trade-off between the need for wider coverage, increased contextual information, and computational constraints.

4.3.7.2 Processes

The presence of a large volume of unique users and unique processes poses a significant challenge in the realm of anomaly detection. Corporate networks are complex ecosystems where thousands of employees access numerous resources, services, and applications daily. This results in an extensive number of unique users and processes, each with their specific behavior patterns, access privileges, and usage patterns.

During the EDA process, it was discovered that browser traffic made up the majority of network connections in corporate environments. However, it is important to note that Sysmon Event ID 3 data offers limited metadata for these connections, providing no details beyond the browser image involved in the request or response. To improve the signal-to-noise ratio and emphasize other types of connections, browser traffic was excluded from the analysis. This decision not only enhanced computational performance by eliminating a substantial number of edges but also allowed for a more focused examination of the more interesting connection types.

Upon excluding browser traffic from the analysis, the most significant contributing processes in the corporate networks were found to be background Windows processes like *svchost.exe* and programming language runtimes such as *java.exe*. These process images were kept in the network as they exhibit unique behaviors that reveal the underlying network structure and communication patterns between various services and applications. By retaining these background processes, edge-based anomaly detection techniques were better equipped to identify deviating patterns and dependencies in the data without the large volume of unique edges and nodes produced by outbound browser traffic.

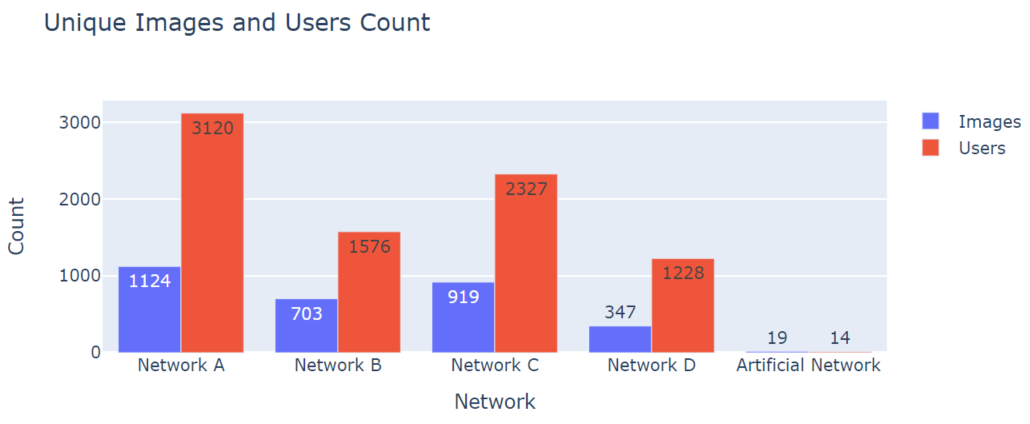


Figure 14: Number of unique users and images per network

Figure 14 displays the number of unique user accounts and images for both artificial and corporate networks. The corporate networks exhibit a significantly higher number of users and unique images compared to the artificial network, with both falling within a similar order of magnitude. This emphasizes the extensive variety of unique users and images present within the four corporate networks.

In the artificial network environment, there was no need for rigorous filtering like excluding browser traffic, as seen in corporate networks, due to the lower traffic volume. As seen in Figure 15, the connection behavior distribution in this environment was mainly dominated by background processes, with user-based Microsoft Office processes occurring at predetermined intervals. The simpler network structure facilitated a more direct analysis, allowing graph-based anomaly detection techniques to efficiently identify patterns and dependencies without requiring extensive filtering or preprocessing.

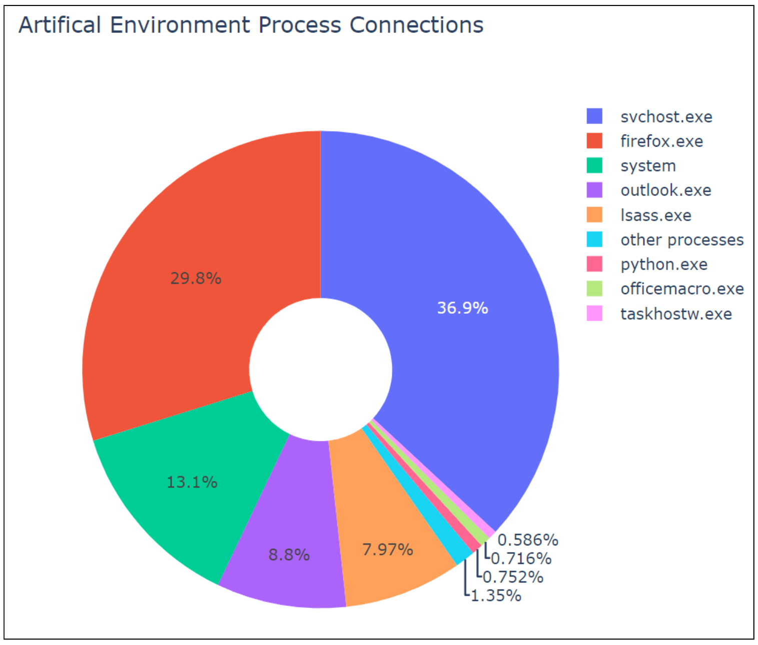


Figure 15: Distribution of Images in the Artificial Network Environment

4.3.8 Users

In Figure 16, it is observed that the most prevalent users across all networks are the predefined local accounts under the `NT AUTHORITY` category, including `SYSTEM`, `LOCAL SERVICE`, and `NETWORK SERVICE`, as well as their locale-specific counterparts. This is an expected outcome, as these accounts play a crucial role in managing security, network resources, and system configurations. They are responsible for tasks such as software updates, security scans, and system maintenance. Consequently, a significant proportion of Sysmon Event ID 3 connection events are associated with these accounts, given their active participation in numerous system processes, services, and scheduled tasks.

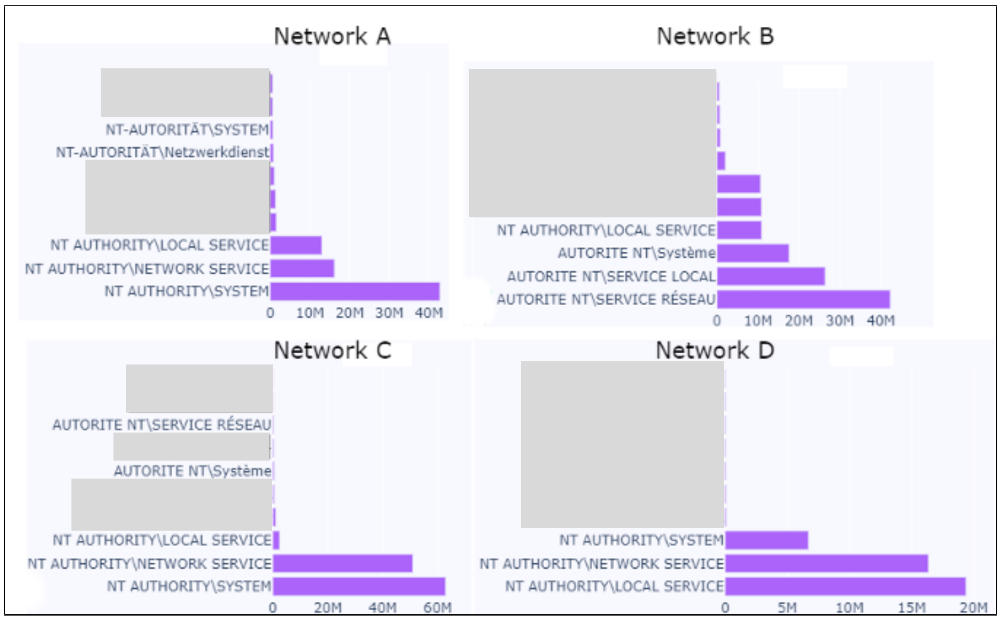


Figure 16: Top 10 Users Across Corporate Networks

As illustrated in Figure 17, the distribution of unique user-based connections in the artificial network appears relatively consistent throughout the log data. This uniformity reflects the communication patterns within the network are predominantly automated. This automated nature of communication is a result of scheduled processes, and other background tasks running at regular intervals, maintaining a consistent level of user activity across the network.

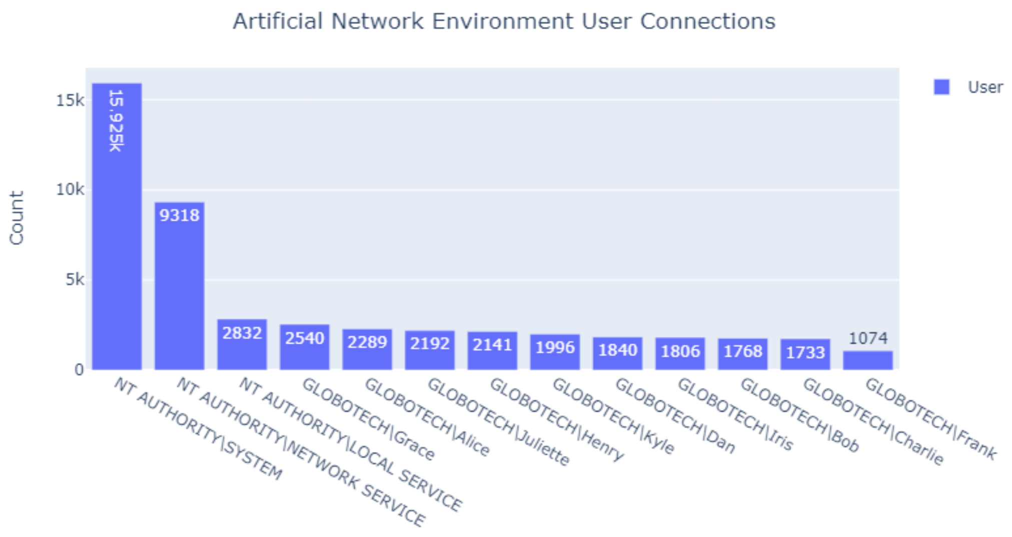


Figure 17: Unique User Connection Count in Artificial Network Environment

4.3.8.1 Network Connectivity

The `NetworkX` transformation of the data facilitated the exploration of several graph-based metrics, such as node connectivity and importance.

The distribution of node importance, measured using eigenvector centrality as utilized in the PageRank algorithm detailed in section 2.4.1, is divided into two distinct sets. According to this metric, a node's importance is recursively determined based on the importance of its immediate neighbors [17]. This pattern was consistently observed across all networks and is illustrated in Figure 18, which displays a log-scaled graph of eigenvector centrality.

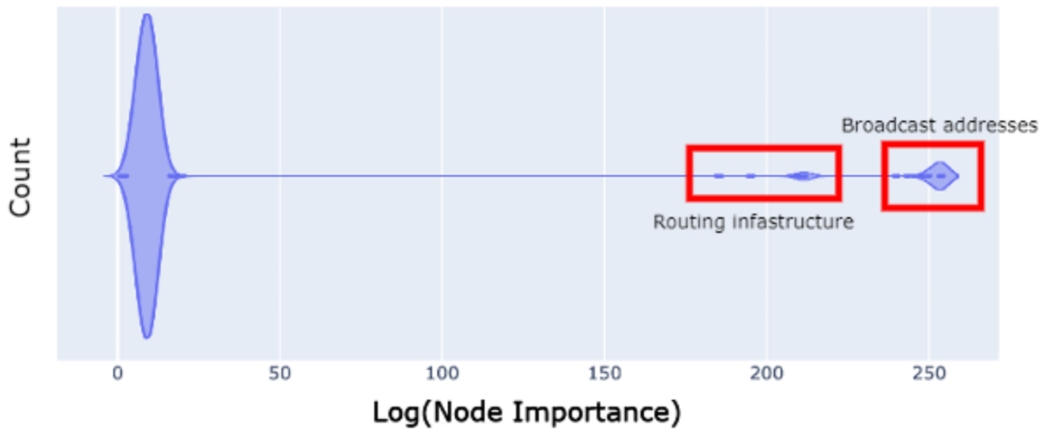


Figure 18: Network A - Log-Eigenvector Node Importance

The majority of nodes hold minimal influence, but a small subset of routing nodes are highly interconnected. A closer examination of this heuristic revealed that internal routing structures are considered highly influential nodes, which is a reasonable conclusion given their function in connecting network components. Additionally, this analysis emphasized the role of broadcast addresses. To reduce the overall noise in the graph data, these broadcast addresses were consequently filtered out of the graphs.

4.4 Feature Engineering

Feature engineering plays a critical role in building a machine learning model. It involves generating new features from raw data or selecting the most relevant features to enhance the model's predictive capabilities. This study categorizes features into two schemas: node and edge. The node schema pertains to individual node characteristics, while the edge schema relates to connections between nodes. This categorization is made possible by the GAT layers, which allows the model to process both node and edge features during the learning phase. This method enables the model to not only recognize structural connections but also learn behaviors based on the inherent characteristics of the node and edge features themselves.

4.4.1 Node Schema

The node schema, displayed in Table 8, comprises features related to the entities within the network. The approach to feature selection relies on manual selection and the development of custom features, drawing upon the findings from the earlier EDA and existing domain knowledge.

In this network, an IP address serves as the "entity", and acts as a unique identifier for each node in the graph. Although additional features are derived from the IP address, such as private or external characteristics, the primary purpose of the IP address is to uniquely identify nodes within the network. This convenient approach establishes the foundation of the directed graph.

4.4.1.1 Binned Ports

A port number is a 16-bit unsigned integer associated with a local or remote connection endpoint. More specifically, ports may be associated with System/Known Ports (0-1023), User Ports (1024-49151) and Ephemeral/Dynamic Ports (49152-65535). This encoding strategy realizes the port ranges outlined in RFC6335 as

Table 8: Node Schema

Node Schema	Encoding Strategy
Binned Ports	Integer
IsPrivate	Binary Flag
OUI Manufacturer	One-Hot Encoding
In Degree	Integer
Out Degree	Integer
PageRank	Floating Point

"bins", where instances of a port connection on a node increments the corresponding bin.

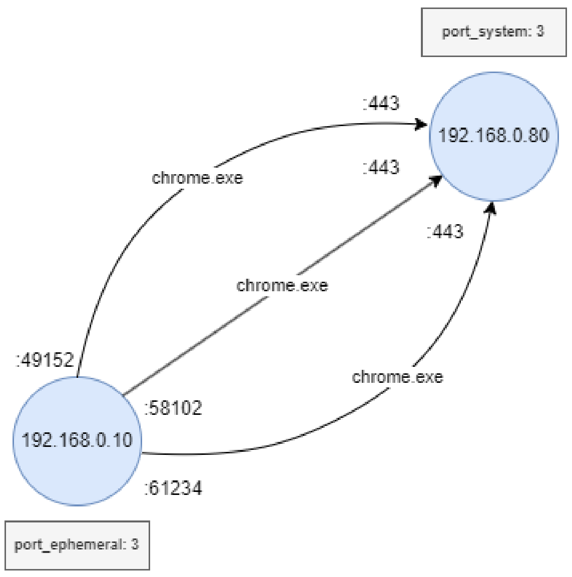


Figure 19: Binned Ports Example

Figure 19 describes an example of this encoding strategy. A significant number of ephemeral outbound browser connections from a monitored host would result in a correspondingly large weight assigned to the ephemeral port bin of a node. On the receiving side, a host serving web content on port 443 would increase the system port weighting. This strategy provides a context for node behaviour based on service communication patterns in typical port ranges.

An issue with a fixed-bin strategy for port encoding is described under the *Security Considerations* section of RFC6335 [41]:

The fact that network traffic is flowing to or from a registered port does

> not mean that it is "good" traffic, nor that it necessarily corresponds to the assigned service. Firewall and system administrators should choose how to configure their systems based on their knowledge of the traffic in question, not whether there is a port number registered or not.

Consequently, the reliability of using port numbers to identify services is limited because there is no strict enforcement governing their assignment. This approach can be disrupted by changing port numbers on the transmitting or receiving side. Despite these limitations, using binned ports to group behaviours still serves as a reasonable heuristic for analyzing network traffic, as it records state associated with the port activity on a node.

4.4.1.2 IsPrivate

A private network IP and a public network IP differ in terms of their accessibility and scope of use. Private network IPs are reserved for internal networks and are not routable over the internet, while public network IPs are globally unique and used for communication over the internet. This experiment uses the $IsPrivate$ flag in conjunction with the semantics in *RFC 6890 - Special-Purpose IP Address Registries* to encode this information as a node feature [42].

According to the Internet Protocol (IP) specifications, private IP addresses are reserved for use in private networks, such as those within an organization, and are not routable on the public internet. The Internet Assigned Numbers Authority (IANA) has reserved several blocks of IP addresses for private networks, including 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16, which are commonly used for internal network communication.

On the other hand, public IP addresses are globally unique and routable over the internet. These IP addresses are assigned to devices by Internet Service Providers (ISPs) and can be accessed from anywhere on the internet. Public-facing IP addresses are essential for devices that need to communicate with other devices over the internet, such as servers and routers.

The $IsPrivate$ feature serves a valuable purpose in the anomaly detection model, as it aids in differentiating between internal and external network traffic. By making this distinction, the model can better analyze network communication patterns and effectively identify potential anomalous behaviors.

4.4.1.3 OUI Manufacturer

An OUI (Organizationally Unique Identifier) is a 24-bit identifier assigned to a manufacturer by the IEEE Registration Authority. OUIs are used as the first three octets (bytes) in the MAC (Media Access Control) addresses of devices produced by that manufacturer.

Using data collected from Osquery, a mapping of OUIs to their corresponding manufacturers was conducted using an OUI database. The IEEE provides a public OUI database that can be used for this purpose [43]. This facilitates the creation of a categorical feature in the model, which can be helpful in identifying patterns or anomalies related to specific manufacturers, device types, or network equipment. OUI manufacturer information may provide some insight into differentiating between devices like client workstations, phones, routers, or servers, as different manufacturers often specialize in producing specific types of network devices.

4.4.1.4 In/Out Degree

The in-degree and out-degree of a node refers to the number of inbound and outbound connections. When directly encoded as a node feature, these structural features can highlight a role of importance in the network. These features can aid in the predictive power of the model, providing a learnable context for increasing or decreasing the likelihood of being connected to another given node in a network.

4.4.1.5 PageRank

The output of the PageRank algorithm is applied to all nodes in the graph, characterizing each node based on its importance in the entire network [17]. The underlying assumption is that important nodes are more likely to receive links from other nodes. The PageRank algorithm outputs a probability distribution associated with the likelihood of connecting to that node from a randomly selected node in the graph. This attribute provides a similar role to the in/out-degree feature, but the importance calculation incorporates a holistic view of the network structure as opposed to node-specific.

4.4.2 Edge Schema

The edge schema consists of features related to the connections between entities in the network. As with the node schema, the selection of relevant and informative features for the edge schema is based on manual selection and hand-crafted features, drawing from EDA and domain expertise.

Table 9: Edge Schema

Edge Schema	Encoding Strategy
Top-K Images	Integer / One-Hot/Frequency Encoding
IsAdmin	Binary Flag
Protocol	One-Hot Encoding

4.4.2.1 Top-K Images

The name of a given process, recorded by Sysmon as an `Image`, is a valuable categorical feature, as it describes the transmitting or listening process handling a network connection. However, this feature exhibits unbounded cardinality in a large network: in a single capture, a network may present thousands of unique images. Further challenges arise from the process names. Images do not follow a natural language grammar, making a word embedding strategy like *word2vec* [15] intractable. Similarly, a one-hot encoding strategy leveraging each unique image as a separate feature would critically affect both the computational time and space complexity of the model.

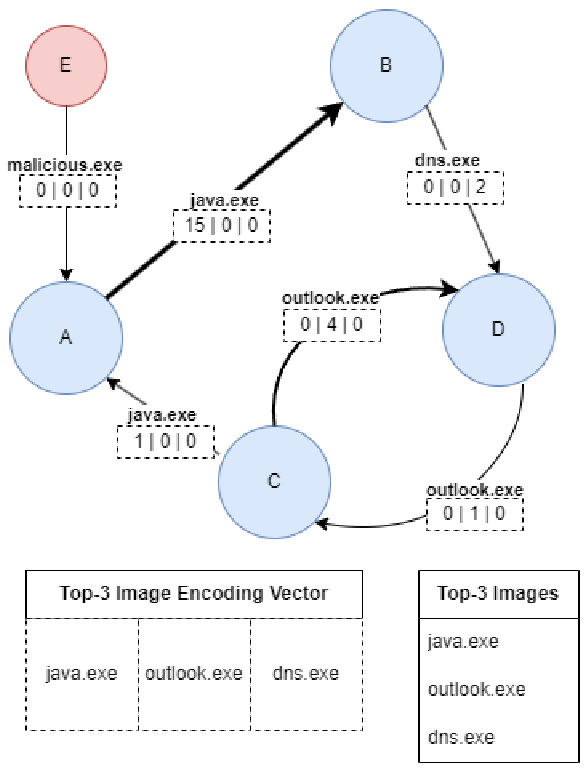


Figure 20: Top-3 Images Example

As a compromise, this experiment selects the Top-K images from the dataset for encoding process names, where K is a user-defined parameter. This is a frequency encoding that only encodes the Top-K images as a fixed-length, weighted vector. As an added benefit, this feature aggregates multiple *(Source, Destination, Image)* links into a single weighted edge, greatly reducing the number of edges used during model training and inference.

A Top-K Image scenario is described in Figure 20. In this example, $K = 3$, therefore the top-3 most frequent images will be encoded. In descending order, *java.exe*, *outlook.exe*, and *dns.exe* are the top-3 most frequent images in the network, so these are the only images that will be encoded. Integer values correspond to a connection summation: the edge between node A and B is indicative of 15 separate *java.exe* connections in a fixed time period. Similarly, the link between B and D is two separate *dns.exe* connections. Due to the top-3 constraint in Figure 20, the $(E, A, malicious.exe)$ connection presents as an empty vector of size K.

The selection of K as a hyper-parameter should be determined experimentally. If K is a very low number, the model will not capture the behaviour of many processes. Contrarily, If K is too large, model performance will be significantly impacted. Overall, this heuristic allows the model to learn traffic patterns associated with the most common processes in the network, and penalizes outlying, one-off connections.

One disadvantage to this approach this that any semantic meaning associated with the process is essentially discarded.

4.4.2.2 IsAdmin

The `User` feature associates a user session with a logged network connection. The `IsAdmin` feature extends this behaviour by associating the connection with an administrative account. This is realized as a binary encoding, with 0 and 1 denoting an unprivileged and privileged connection, respectively.

`NT AUTHORITY\SYSTEM` is the default superuser account on Windows computers. However, client-specific EDA can also identify administrative accounts: in a contrived example, `GLOBOTECH\AdminJoe` can be interpreted as an administrative account. Consequently, identifying strings are trivially matched using regular expressions, as given in Listing 4.

```python
import re

def is_admin(
    user: str,
    ADMIN_IDENTIFIERS: list[str] = ["NT AUTHORITY", ...],
) -> bool:
    """Checks if a given user is an administrative account."""
    return any(re.match(user, _id) for _id in ADMIN_IDENTIFIERS)
```

Listing 4: Admin Account String Matching

A downside to this approach centers on the expert knowledge required to populate the `ADMIN_IDENTIFIERS` vector, as the administrative accounts were not provided by the companies in question during this experimentation. This is mitigated by one pass of client-level EDA, using best judgment for identifying strings associated with administrative accounts.

4.4.2.3 Protocols

The transport-layer protocol of a connection captured by *Sysmon* is a string value, indicating *UDP* or *TCP* traffic. *Sysmon* only captures traffic event ID 3 using these protocols, so there is no possibility of missing values in this field. This feature is trivially encoded as a binary feature. In combination with other edge-features, like Top-K Images, a model will learn to associate certain protocols with certain images. This feature can derive potentially anomalous behaviour deviating from a learned context, like a spurious *dns.exe* connection leveraging TCP instead of common UDP-based communication.

4.5 Phase 2: Model Development

Model Development is a critical stage in the overall process, as it involves the creation and training of an unsupervised anomaly detection system that can score graph data. In this phase, the focus is on implementing the model as well as adjusting GAT hyperparameters to ensure optimal performance. This phase is essential for laying the groundwork for a robust unsupervised detection system, which, in turn, plays a vital role in addressing the challenges associated anomalous edge detection.

Figure 21 outlines a high-level design of the link prediction model. The pipeline begins with node and feature-level encoding using the message-passing framework of GNN layers. During this phase, a reconstruction strategy that considers the presence of anomalous edges in the original graph data will be used [27]. This strategy uses losses derived from the original graph and the resultant reconstruction to predict the presence probability of edges between adjacent nodes following the decoding process.

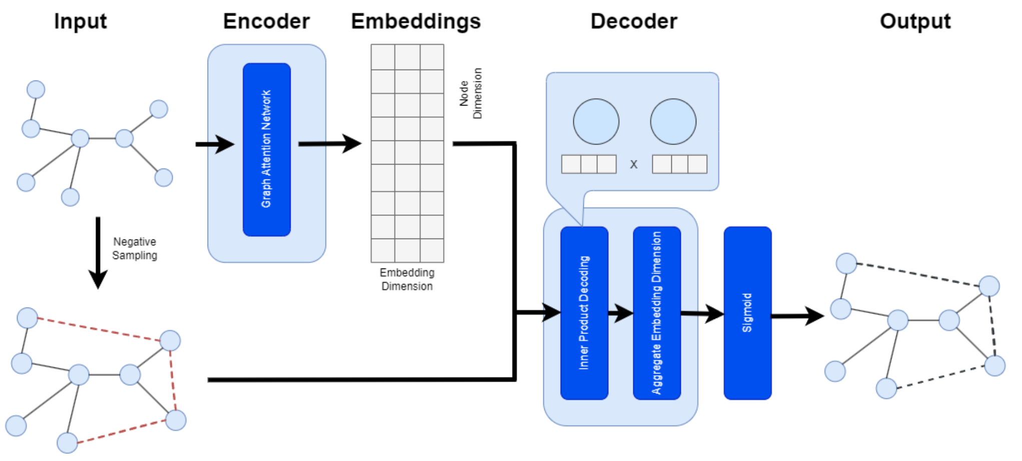


Figure 21: Link Prediction Model

In order to effectively handle graph data, the widely-recognized PyTorch Geometric library was utilized. The GATv2Conv encoding layers were chosen based on their outstanding performance in previous work and their ability to accommodate both node and edge features [31] [26]. The model worked directly with the node features, edge index, and edge attribute matrices. Through the encoding process, the encoder produced embeddings for each node that incorporated information about the node itself, as well as its local edges and surrounding neighborhood. This approach enabled the model to effectively learn and represent complex relationships within the graph data. Granular details regarding the implementation of the model are available in Appendix A.1.2.

The decoder utilized was a standard InnerProductDecoder, which is a common selection used in literature [21], [12], [27]. The InnerProductDecoder functions by computing the inner product of the node embeddings generated by the encoder. This computation results in a similarity score between each pair of nodes, which is then transformed into a probability using a sigmoid activation function. The predicted presence probability of the edge represents the likelihood of an edge existing between the two nodes.

The predicted presence probability of the edge was compared against the original graph to evaluate the model's ability to reconstruct the original graph accurately. When the model showed no sign of improvement during the train/test loop, large reconstruction errors for edges were considered indicators of anomalous behaviour in the input network. The GAT-GAE training and test loops are detailed in Appendix A, sections 1.5 and 1.6, respectively.

4.5.1 Model Training

The model parameters are updated using the Adam optimizer, with an initial learning rate of 0.01, determined through trial and error. The model is trained separately on each dataset, and the training process continues until a user-defined *patience* parameter in the `EarlyStopper` algorithm is reached. This stopping criteria, available in Appendix A.1.4, is triggered when there is no improvement in the model's performance for 50 consecutive epochs, at which point the learning rate is reduced by a factor of 10.

Table 10 outlines the values assigned to a variety of hyperparameters within the GAT-GAE model. The activation function, loss type, and optimizer were choices derived from the original GAE model developed by Kipf et. al. [27]. The remainder of the parameters were determined experimentally. Deeper networks yielded poorer reconstruction performance, which is a consistent result associated with the over-smoothing problem in GNNs [5].

To preserve the best-performing model parameters for each dataset, a standard greedy algorithm is employed. This process compares the current reconstruction accuracy with the highest achieved accuracy for the specific dataset. When the current accuracy surpasses the previous best, the model's weights are serialized, ensuring that the optimal parameters for each dataset are retained.

Table 10: GAT-GAE Autoencoder Parameters

Parameter	Value
Activation Function	ReLU
Loss	Binary Cross-Entropy With Sigmoid Loss
Optimizer	Adam
Number of GAT Layers	3
Learning Rate	0.01
K (Top-K Images)	256
Attention Heads	4

In all scenarios, a threshold was chosen to maximize the F_1 score, ensuring the best balance between precision and recall. In practice, the optimal threshold typically fell within a range of 2 to 3 standard deviations from the mean reconstruction error.

4.6 Phase 3: Model Evaluation

In this section, the results of the scenarios and corporate network events are discussed. This includes insights surrounding the reconstruction error distributions, the distributions of anomalies in the scenarios, and metrics associated with the performance of the anomaly detection pipeline.

4.6.1 Synthetic Data Insertion for Model Evaluation

With the absence of ground truth labels for anomalous behaviour present in the networks, model performance was evaluated by inserting a known quantity of synthetic data into the corporate networks. Four distinct scenarios encompassed a range of anomalous network behaviors, generating metrics that assessed the model's performance under various conditions. In the case of the artificial network environment, the resulting graph was small enough, with only 256 edges, that it could be manually labeled and inspected without the need for scenario-derived metrics. By analyzing the model's ability to detect anomalies, its effectiveness in identifying different types of anomalous behaviors could be quantified.

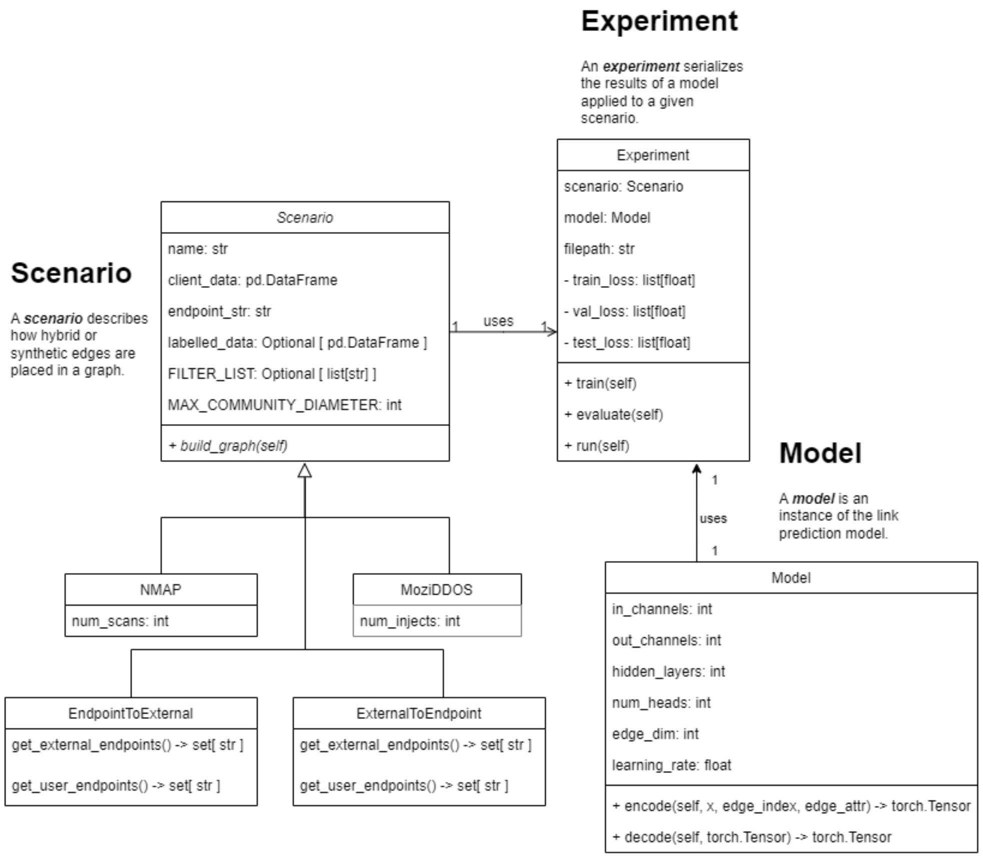

Experiment
An experiment serializes the results of a model applied to a given scenario.
Experiment
scenario: Scenario
model: Model
filepath: str
- train_loss: list[float]
- val_loss: list[float]
- test_loss: list[float]
+ train(self)
+ evaluate(self)
+ run(self)
Scenario
A scenario describes how hybrid or synthetic edges are placed in a graph.
Scenario
name: str
client_data: pd.DataFrame
endpoint_str: str
labelled_data: Optional [pd.DataFrame]
FILTER_LIST: Optional [list[str]]
MAX_COMMUNITY_DIAMETER: int
+ build_graph(self)
1
uses
1
Model
A model is an instance of the link prediction model.
1
uses
1
NMAP
num_scans: int
MoziDDOS
num_injects: int
EndpointToExternal
get_external_endpoints() -> set[str]
get_user_endpoints() -> set[str]
ExternalToEndpoint
get_external_endpoints() -> set[str]
get_user_endpoints() -> set[str]
Model
in_channels: int
out_channels: int
hidden_layers: int
num_heads: int
edge_dim: int
learning_rate: float
+ encode(self, x, edge_index, edge_attr) -> torch.Tensor
+ decode(self, torch.Tensor) -> torch.Tensor

Figure 22: Experiment Class Diagram

Figure 22 depicts the software architecture used in the experiment. Key features of this architecture include that an Experiment is composed of one Scenario instance and one Model instance. A Scenario, acting as an interface, outlines the process of injecting labeled data into an input graph. Each Scenario object contains an identifying endpoint string, stored as *endpoint_ str*. This string literal is employed to identify human users operating workstations within a corporate network environment.

Additionally, a scenario may contain an optional FILTER_LIST, which is a list of images that are to be excluded from the graph. For this specific experiment, the list was populated with browser images such as chrome.exe, edge.exe, and firefox.exe. Furthermore, a constant integer value, the maximum community diameter, is recorded in the scenario. Small, isolated subgraphs may represent noise or rare events that are not representative of the overall network behavior. Removing them can help focus the analysis on more meaningful patterns and relationships.

4.6.1.1 NMAP

The NMAP scenario was designed to simulate the behaviour of NMAP scans within the network structure. The NMAP scenario was unique in that it solely relied on synthetic data, and none of the inserted edges were derived from previously labeled events.

Initially, the scenario examined the graph to locate /24 private IPv4 subnets in the network structure, with a requirement that each subnet contained at least 20 endpoints. Within these subnets, a node was chosen at random to create connections to all other nodes in the same subnet. This procedure was repeated for up to 50 identified subnets, until at least 3,000 synthetic edges were added to the graph. While there was no specific reason for selecting 3,000 as the minimum number, it was an intentional decision to ensure that the newly inserted edges constituted a statistically significant portion of the total population.

The new edges were assigned the image name *nmap.exe* and given TCP as their protocol, in line with default NMAP behavior [44]. Edges also originated from an administrative account, which simulated a compromised workstation. Additionally, this scenario uniquely affected the targeted destination nodes by updating the binned port node feature to accurately represent a full port scan of the targeted nodes.

4.6.1.2 MoziDDoS

The MoziDDoS scenario aimed to simulate the behavior of an IoT botnet DDoS attack, specifically reflecting the known activities of the Mozi botnet [34]. This was the only scenario that involved adding new nodes to the graph. To initiate the process, malicious URLs and associated binaries were collected from urlhaus.org, a collaborative platform where security researchers, ISPs, and security vendors can submit and share information about malicious URLs hosting malware, exploit kits, or phishing sites. URLhaus maintains a regularly updated database for use by security professionals to enhance their security measures, such as blocking known malicious URLs or analyzing emerging threats.

In this study, a database of actively malicious IPs and metadata describing their associated binaries was downloaded from URLhaus. This data was filtered to select all Mozi-associated hosts, amounting to approximately 2,000 new nodes. Existing user workstations in the network structure were randomly chosen to undergo a sequence of events:

1. The user workstation would visit the malicious IP, resulting in an edge being inserted into the network.

2. The malicious IP would send back a response, analogous to the malicious IoT botnet software being downloaded. This would lead to another edge being inserted into the network.
3. Finally, the botnet would be "activated," and an edge would be inserted between the user workstation and a randomly selected public IP already present in the graph. This randomly selected public IP would be the same address for all nodes in the scenario, simulating numerous requests from internal addresses.

As a result of this sequence of events, approximately 6,000 new edges are inserted into the network structure during this scenario. It is important to note that, since the network input is formulated as a static graph, there is no concept of sequenced activities based on elapsed time, making all these activities appear to occur concurrently.

4.6.1.3 ExternalToEndpoint and EndpointToExternal

The objective of the ExternalToEndpoint and EndpointToExternal scenarios was to simulate lateral movement and file infiltration/exfiltration.

Both scenarios shared similarities in their implementation. Initially, a labeled set of "true positive" events were provided by the research partner granting access to the environment. These events were actual Sysmon Event ID 3 events originating from the networks under analysis, triggered by a production signature-based detection system. A separate team of analysts individually investigated each event. In every case, the events stemmed from potentially unwanted programs. As Sysmon Event ID 3 events, they could be seamlessly integrated into the network structure by trivially modifying the source and target nodes in the record.

The modified edges were incorporated into the graph structure by randomly selecting existing internal user workstations and connecting them to a randomly chosen external IP address. This injection resulting in approximately 7,000 edges added to the graph under test. User workstations were identified by the presence of client-specific strings, an instance variable defined in the `Scenario` object of Figure 22. Finally, the direction of the inserted edge was determined by the scenario, with ExternalToEndpoint representing an inbound connection and EndpointToExternal representing an outbound connection, relative to the internal company network.

4.6.2 Verification

Over the course of experimentation, a complete anomaly detection pipeline was built and utilized to score presence probability connections in a multi-relational directed graph. This pipeline includes the collection, ingestion, and transformation of corporate *Sysmon* log data into several intermediate formats. These intermediate formats facilitated EDA in both tabular and relational formats to identify particulars associated with the corporate networks. Next, a methodology for feature selection and engineering was developed for the anomaly detection model. Finally, a GAT-GAE model was developed and trained on the host log data, yielding a vector of probability scores. These scores, when ranked, provide insight into anomalous connections in the host network.

4.6.3 Validation

The aim of this research is to determine whether an unsupervised GNN model can detect anomalous network connections in a static, attributed network. To validate the effectiveness of the anomaly detection method, four corporate datasets were injected with anomalous data. This allowed for the calculation of pseudo-performance metrics, such as precision, recall, and F_1 Score, to evaluate the overall performance of the anomaly detection framework. The results were assessed based on the model's performance under various custom scenarios, which controlled the insertion of anomalies into the networks under test.

The analysis demonstrated that the GNN model was effective in segregating a significant number of potentially interesting and outlying behaviors from the majority of the network traffic. The limitations of the evaluation method impacted the ability to have numerically precise performance metrics, though this does not deter from the fundamental effectiveness of the model in detecting anomalies, including malicious ones.

Despite these limitations, the entire anomaly detection pipeline serves as an effective coarse filter using Sysmon Event ID 3 logs, which are readily available on Windows networks.

4.6.3.1 Reconstruction Error Distributions

Two recurring patterns appear in all network reconstructions. Firstly, there is a significant majority of edges with low reconstruction error, which indicates a high presence probability in the network. This results in a pronounced overall score distribution that is skewed positively. Secondly, there is a group of edges with higher reconstruction errors that form a distinct and uniform "long tail" in the score distribution. This is visually distinguishable from the primary population, and can present for several reasons:

1. Lack of discernible anomalous behaviours: If the anomalies in the graph do not follow any learned patterns or share common characteristics, the GAT-GAE model may struggle to learn features that distinguish them from the rest of the edges. Consequently, the reconstruction errors for these anomalies can spread uniformly across a range of values.

2. Feature insensitivity: If the node and edge features used by the GAT-GAE model are not sensitive enough to capture the nuances between anomalous and benign edges, the reconstruction errors for the anomalies may end up being uniformly distributed. This may be an indication that the model is not able to effectively distinguish between anomalous and non-anomalous edges based on the available feature set.

3. Inherent randomness: It is also feasible that the anomalies themselves are inherently random, and do not follow any specific pattern or structure. In this case, the uniform distribution of reconstruction errors may be a reflection of the real distribution of anomalies in the graph.

The results of all experiments are provided in Appendix B, in the form of both scatter plots and confusion matrices. Each scatter plot features a marginal distribution on the right side, illustrating the edge distribution at the same scale as the main plot. The threshold value, selected based on the optimal F_1 Score, efficiently segregates the majority of edges. This separation is especially evident in the marginal distribution. It's important to note that false negatives in the scatter plots may visually obscure the true negatives, but the marginal plot reveals that the majority of true negatives are clustered around 0. This segregation is conveyed in the corresponding confusion matrix, given in Figure 24.

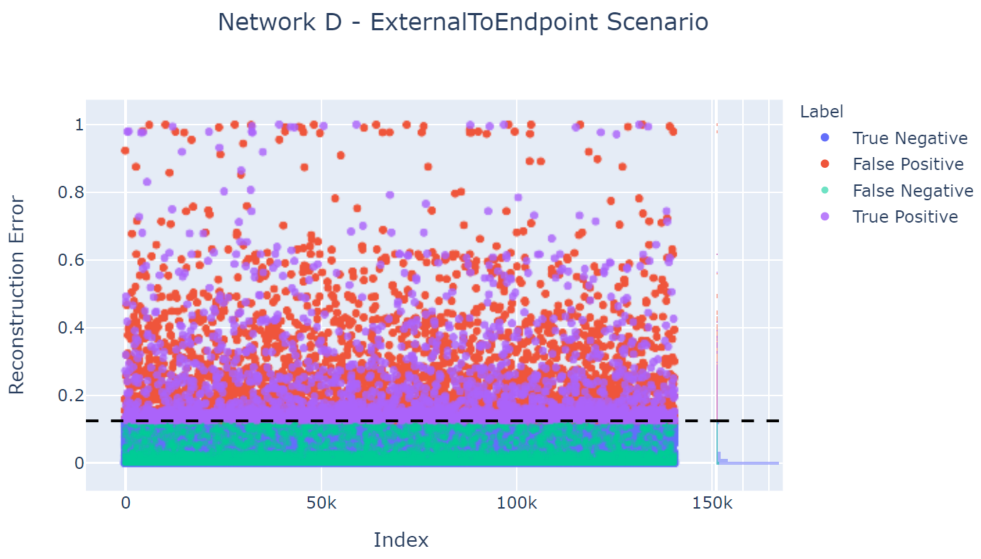
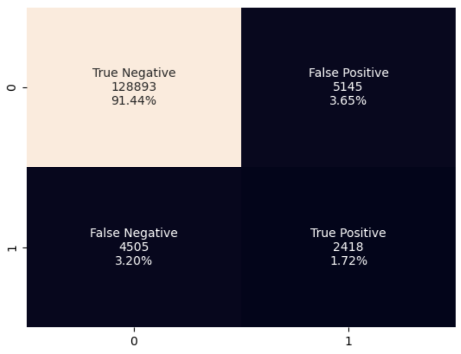


Figure 23: Reconstruction Scatter Plot - Network D (ExternalToEndpoint)

A closer look at the false positives in scatter plots, such as Figure 23, was necessary. Even though they were not part of the labeled data set, the false positives displayed some degree of unusual behavior, often as a single connection event in the broader context of the network. For instance, the series of false positives centered around 1 in Figure 23 were all linked to separate one-time connections to external addresses and were notably rare occurrences among the 45 million logged events for Network D. This false positive behaviour with unlabeled datasets in computer networking environments is congruent with results from similar research [21].



	0	1
0	True Negative 128893 91.44%	False Positive 5145 3.65%
1	False Negative 4505 3.20%	True Positive 2418 1.72%

Figure 24: Confusion Matrix - Network D (ExternalToEndpoint)

Figure 25 presents the distribution of control reconstruction scores derived from all four corporate networks. These scores were calculated without incorporating any scenario injections, indicating that the reconstructions represent the baseline behaviors of the networks. The presence of outlying behaviors within these large network captures, even without anomaly injections, is evident.

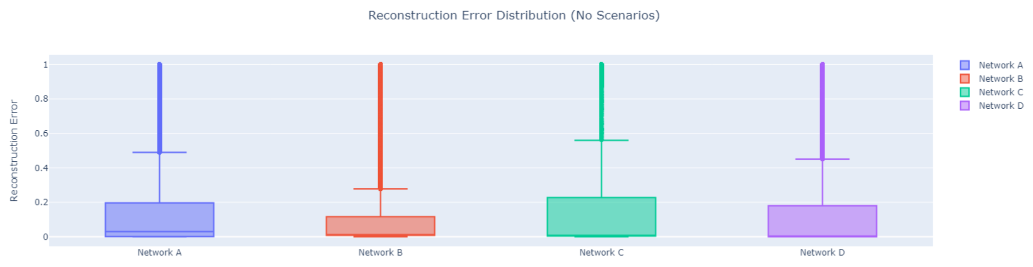


Figure 25: Reconstruction Distributions Across Corporate Networks

The distribution of reconstructions was more closely related to the original input network than the tested scenario. This observation results in relatively similar reconstruction error distributions across the four scenarios per client, with minor variations. This is expected because the input graphs themselves have significantly more variation than the changes introduced by the synthetic anomaly injection.

4.6.3.2 NMAP Scenario Results

In the NMAP scenario, Networks B and D exhibited the best overall performance metrics. Intriguingly, this scenario displayed the most substantial disparity between recall and precision scores compared to all other scenarios. The likely cause for this discrepancy is the nature of the NMAP scenario itself, which ended up inserting fewer edges into the network structure than other scenarios, as the other scenarios had less stringent criteria for anomaly insertion, whereas the NMAP scenario needed to consider viable subnets and workstations for insertion. In practice, the number of edges inserted constituted between a minimum of 0.7% (Network C) and a maximum of 2.41% (Network D) of the total edges in the network. Moreover, this scenario involved altering source and destination nodes, which influenced the reconstruction scores associated with the modified nodes.

Table 11: *NMAP* Scenario Metrics

	# Edges	Recall	Precision	F_1 Score
Network A	420,935	0.32	0.15	0.20
Network B	240,887	0.30	0.22	0.25
Network C	982,592	0.24	0.17	0.19
Network D	134,696	0.35	0.28	0.31

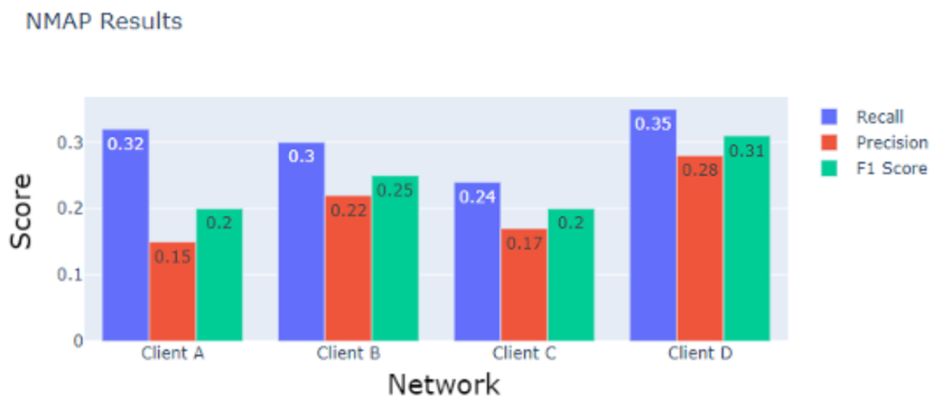


Figure 26: NMAP Grouped Metrics

The smaller number of synthetic edges in the NMAP scenario led to a more challenging anomaly detection task, as the signal-to-noise ratio decreased, making it harder for the graph-based anomaly detection techniques to effectively discern the anomalous connections. Consequently, the larger gap between recall and precision scores in this scenario potentially highlights the sensitivity of the model's performance to the number of synthetic edges.

4.6.3.3 MoziDDOS Scenario Results

In the MoziDDoS Scenario, the model's performance was tested against hybrid edges connecting user workstations, which operated as DDoS nodes participating in an attack against a randomly selected public IP within the network. Across all networks, the model demonstrated the ability to detect edges engaged in this attack sequence. However, Network C underperformed compared to the other networks by a considerable margin.

Table 12: *MoziDDOS* Scenario Metrics

	# Edges	Recall	Precision	F_1 Score
Network A	425,719	0.35	0.29	0.31
Network B	243,291	0.35	0.32	0.33
Network C	988,415	0.27	0.20	0.22
Network D	136,985	0.37	0.33	0.35

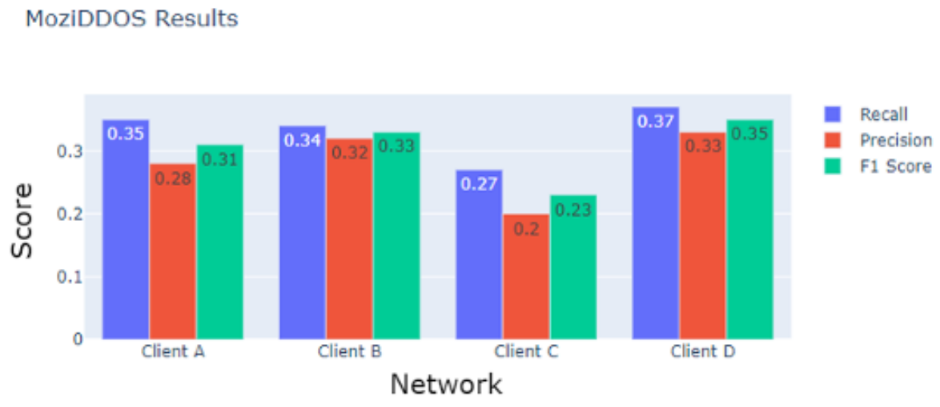


Figure 27: MoziDDOS Grouped Metrics

The disparity in performance for Network C can likely be attributed to the scale of the network itself. Network C possesses almost ten times the total number of edges present in Network D, resulting in a more complex and intricate network structure. The increased complexity and sheer volume of connections in Network C make it more challenging for the graph-based anomaly detection techniques to effectively identify the DDoS attack connections among the vast number of legitimate edges. This highlights the impact of network scale on the model's performance and emphasizes the need for further optimization and adaptation of the techniques to maintain their effectiveness in detecting anomalies in large-scale networks.

4.6.3.4 ExternalToEndpoint Scenario Results

The ExternalToEndpoint Scenario assessed the model's performance against anomalous connections between external addresses and user workstations, which often indicate the presence of a Command and Control (C2) channel or some form of file infiltration. In this scenario, Client B and D once again outperformed the other networks.

Table 13: ExternalToEndpoint Scenario Metrics

	# Edges	Recall	Precision	F_1 Score
Network A	427,853	0.28	0.24	0.26
Network B	247,396	0.33	0.29	0.31
Network C	981,566	0.21	0.20	0.20
Network D	140,961	0.35	0.31	0.33

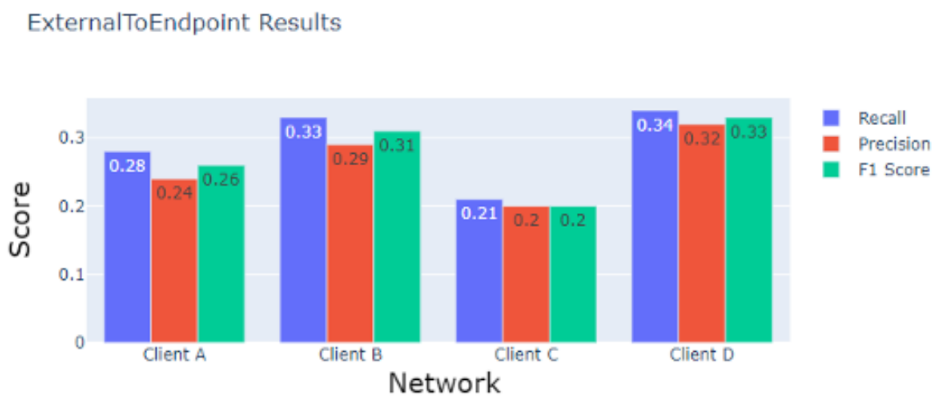


Figure 28: ExternalToEndpoint Grouped Metrics

Despite being based on 45 million records, Network D's network demonstrated an impressive detection capability. Notably, Network B achieved a comparable level of performance despite having twice the number of connections, originating from 136 million Sysmon records. This indicates that the graph-based anomaly detection techniques utilized by the GAT-GAE model are effective in identifying suspicious connections, even when working with a limited dataset in networks with a substantial number of connections.

The detection of anomalous connections in the ExternalToEndpoint Scenario underscores the potential of edge-based anomaly detection techniques for uncovering covert C2 channels and file infiltration attempts.

4.6.3.5 EndpointToExternal Scenario Results

The EndpointToExternal scenario evaluated behaviors similar to those in the ExternalToEndpoint scenario, but with reversed connection directions. This type of behavior could be indicative of file exfiltration attempts, C2 channels, or unauthorized access to external resources. The anomaly detection performance across all instances was largely identical to the results obtained in the ExternalToEndpoint scenario. However, an interesting observation is that all metrics either remained the same or suffered a slight decrease in performance.

Table 14: EndpointToExternal Scenario Metrics

	# Edges	Recall	Precision	F_1 Score
Network A	427,853	0.27	0.22	0.24
Network B	247,395	0.32	0.29	0.31
Network C	981,566	0.19	0.19	0.19
Network D	140,961	0.34	0.31	0.32

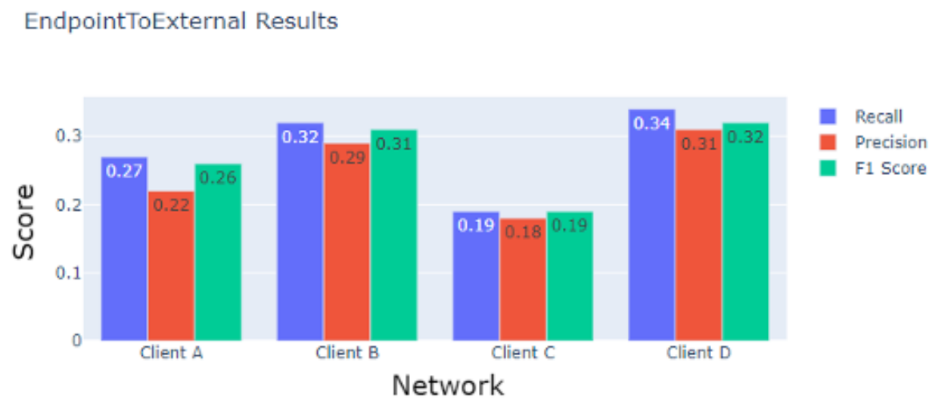


Figure 29: EndpointToExternal Grouped Metrics

This subtle difference suggests that the reversal of connection directions might have assisted the model in learning the behaviors of the anomalous connections, potentially enabling the message-passing framework to aggregate more neighborhood information into the node embeddings. Although the decrease in performance is minimal, it does imply an impact on the model's effectiveness in detecting suspicious connections when the directionality is altered.

The results from the EndpointToExternal scenario emphasize the importance of considering connection directionality in graph-based anomaly detection and highlight the need for further exploration and refinement of the model to ensure robust performance across varying connection behaviors and network characteristics.

4.6.3.6 Artificial Network Environment Results

The Sysmon EID 3 logs of the Artificial Network Environment were extracted, yielding approximately 50,000 records. These records produced a directed multi-relational graph consisting of 435 nodes and 256 edges. This relatively low number of edges in the network highlights the relatively predictive behaviour of the system.

Table 15: APT-29 Artificial Environment Scenario Metrics

	# Edges	Recall	Precision	F_1 Score
APT-29 Campaign	256	0.49	0.76	0.60

Since the artificial network was smaller and less complex, it could be exhaustively labeled for anomalous behaviors manually. This comprehensive labeling process allowed for a more accurate identification of true positives, leading to a significantly higher precision score compared to the custom scenarios in corporate environments. The increased precision in the artificial network demonstrates the effectiveness of the graph-based anomaly detection techniques when working with well-annotated data and further highlights the importance of accurate labeling when evaluating the performance of such models.

The model successfully identified anomalous activity originating from the APT29 attack campaign. Notably, it detected all instances of `python.exe`, a Cobalt Strike beacon concealed under a `gcc.exe` alias, all `powershell.exe` activities, and some, but not all, Microsoft Office-suite software tools found to be networking with external endpoints. It is worth mentioning that two out of three Microsoft Office networked connections were detected, while the one that remained undetected had a higher frequency than the other two combined. The Microsoft Office networked connections that fell below the threshold for alerting occurred more frequently in the network environment, likely appearing more normal as a behavioral baseline in the network.

4.7 Discussion

The results presented in this study raise several points of discussion. One important aspect to consider is the inherent uncertainty in using supervised metrics with incomplete labeling in large, noisy corporate network environments. The complexity and volume of data in these environments can lead to ambiguities and inaccuracies when evaluating the performance of the model. As such, it is crucial to acknowledge these limitations when interpreting the results and making recommendations for practical applications.

The feature selection in this study was intentionally kept light to avoid significant computational overhead in a real-life operational setting, with exceptional network size. While this approach enabled faster processing and analysis, it may also contribute to a higher rate of false positives compared to more sophisticated feature engineering and selection methods. However, the results generally demonstrate the model's capability to filter out a considerable amount of connections and highlight outlying events.

It is important to note that the current rate of false positives may be too high for direct use in a production IDS. Instead, the current formulation of this tooling could serve as a coarse-grain filter to reduce the number of events processed per second, allowing analysts to focus on a smaller, more manageable subset of potentially anomalous connections. Holistically, this outcome is congruent with similarly recent the works of King et. al., discussed in the Chapter 2.11.1 [21].

In summary, though the model demonstrates promising results in identifying anomalous network behaviors, further research and refinements are necessary to address the challenges posed by incomplete labeling and high false positive rates. By acknowledging and addressing these limitations, the model could potentially become a more valuable asset in enhancing the detection and analysis of network anomalies in real-world settings.

4.8 Summary

This chapter presented the results of the experiment focused on anomalous edge detection, detailing the various stages of the research process. The data collection and exploratory data analysis phases laid the groundwork for understanding the problem domain and identifying potential features for detection. Feature selection and engineering were performed to define the node and edge schema for the Graph Autoencoder (GAT-GAE) model. The GAT-GAE model was then developed and optimized for detecting anomalous edges in complex network environments. Quantitative and qualitative assessments were conducted using custom scenarios to evaluate the model's performance and generalizability. The model was further validated against a simulated Advanced Persistent Threat (APT-29) campaign in an artificial network environment, demonstrating its effectiveness in identifying anomalous edges and potential security threats in real-world situations.

5 Conclusion

This research has successfully yielded an edge-based anomaly detection model, utilizing a GAT-GAE autoencoder architecture to identify anomalous connections within large directed graphs. The model was validated against four real-world corporate networks and one artificial network, demonstrating a promising ability to detect anomalous connections in a variety of contexts. Despite the challenges posed by the scale and complexity of the networks, the model has shown its potential to identify anomalous behaviours. As network security threats continue to evolve, this work serves as a valuable foundation for developing more robust and adaptable graph-based anomaly detection techniques that can effectively safeguard corporate environments and mitigate potential risks. With further research and optimization, this edge-based anomaly detection model has the potential to significantly contribute to the advancement of network security and the protection of valuable digital assets.

5.1 Contributions

The contributions from this research are as follows:

1. **Log Extraction Tool Development.** A log extraction tool for S3-compliant archives was designed and implemented, featuring dynamic query generation and a user-friendly API. This tool enables efficient extraction of relevant information from unstructured log data, facilitating the further analysis of network connections in the context of anomaly detection.

2. **NetworkX Graph Merging Tool.** A generic graph merging algorithm utilizing the NetworkX library was developed. This function streamlines the process of merging network graphs while aggregating node and edge features, improving the efficiency and scalability of the anomaly detection pipeline when dealing with large-scale datasets.

3. **Early Stopping Class.** As of writing, the PyTorch library does not have a built-in solution for training cessation based on training/validation loss divergence, and instead forces users to rely on third-party tooling or manual implementation. This document contributes a generic `EarlyStopper` class for use in any machine learning application, available in Appendix A.1.4.

4. **End-to-End Graph Anomaly Detection Pipeline.** An end-to-end graph anomaly detection pipeline was constructed for detecting anomalous network connections using Sysmon Event ID 3 log data. This comprehensive pipeline

integrates the log extraction tool, graph merging tool, and graph neural network-based anomaly detection techniques, providing a robust and efficient solution for identifying anomalous edges in network data.

This exploratory research serves as a foundation for future work in edge-based anomaly detection and other graph techniques. Potential avenues for further investigation include exploring edge/node anomalies, incorporating temporal aspects, and employing different feature engineering strategies, both in general and for specific attack families.

5.2 Future Work

In light of the findings from this exploratory research on edge-based anomaly detection in large corporate networks, several recommendations for future work can be proposed. Firstly, future research could explore the development of a more robust set of features engineered from events beyond Sysmon Event ID 3, balanced with common sense processing constraints when dealing with real life networks. This would facilitate the development of a more comprehensive and robust set of features that could be intelligently incorporated into the graph structure. By doing so, the detection capabilities of the system could be significantly enhanced, allowing for the identification of a broader range of anomalies and ultimately contributing to improved security in large-scale enterprise networks. Secondly, there is potential to streamline the data processing pipeline by reducing the number of intermediate data formats. Logs could be transformed directly into the PyTorch Geometric format, resulting in lower processing overhead and increased efficiency. Finally, an additional direction for future research could involve the investigation of anomalous sub-community detection within the network graphs. This approach could uncover more intricate threat patterns and further improve the overall performance of anomaly detection systems in the information security domain.

5.3 Recommendations

Based on the performance of the edge-based graph anomaly detection model presented in this study, several recommendations can be made to further enhance its effectiveness and adaptability in various network environments. First, it is clearly beneficial to explore methods for experimentation in large-scale, realistic networks, as it has been observed that the size and complexity of the network can significantly impact the model's ability to accurately detect anomalies. Employing scalable graph neural network architectures could be considered to address this challenge.

Additionally, it is important to examine the impact of connection directionality on model performance, as it has been observed that reversing connection directions may slightly affect the effectiveness in detecting suspicious connections. Further research could be conducted to refine the ability to adapt to different connection behaviors, ensuring robustness in detecting anomalies across varying network characteristics.

Another area of interest is the development of techniques for handling imbalanced datasets, as the scenarios with fewer synthetic edges tended to show a larger gap between recall and precision scores. Increasing the number of instances in the anomalies by duplicating or generating far more synthetic sample may help balance the class distribution and provide the model with more examples of anomalous behaviors.

Finally, it is crucial to continuously validate and update the model to account for the evolving nature of network traffic and emerging threats. Regularly incorporating new data and retraining the model will ensure that it remains effective in identifying anomalous connections and safeguarding corporate networks against potential security risks.

References

[1] Atlas. (2022) Av-test institute. [Online]. Available: https://portal.av-atlas.org/

[2] IBM. (2022) The cost of a data breach report. [Online]. Available: https://www.ibm.com/resources/cost-data-breach-report-2022

[3] R. Bridges, "A survey of intrusion detection systems leveraging host data," *ACM Computing Surveys, vol. 252, no.6, pp. 1-35*, 2020.

[4] P. Mishra, V. Varadharajan, U. Tupakula, and E. Pilli, "A detailed investigation and analysis of using machine learning techniques for intrusion detection," *IEEE Communications Surveys, vol. 21, no. 1, pp. 686-728*, 2019.

[5] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 14, no. 3, pp. 1-159*, 2020.

[6] D. Duan and L. Tong, "AANE: Anomaly aware network embedding for anomalous link detection," *2020 IEE International Conference on Data Mining (ICDM)*, 2020.

[7] J. Song, X. Qu, and Z. Hu, "A subgraph-based knowledge reasoning method for collective fraud detection in e-commerce," *Neurocomputing, vol. 461, pp. 587-597*, 2021.

[8] M. Russinovich. (2022) System monitor. [Online]. Available: https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon

[9] Facebook. (2014) Introducing osquery. [Online]. Available: https://code.facebook.com/posts/introducing-osquery/

[10] MITRE. (2019) Lateral movement, Tactic TA0008. [Online]. Available: https://attack.mitre.org/tactics/TA0008/

[11] A. links open overlay panelAli Shiravi and A. n. i. detection, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," Jan 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404811001672

[12] L. Ouyang, Y. Zhang, and Y. Wang, "Unified graph embedding-based anomalous edge detection," *IEEE 2020 International Joint Conference on Neural Networks*, 2020.

[13] (2022) Understanding sysmon events. [Online]. Available: https://rootdse.org/posts/understanding-sysmon-events/

[14] I. Amazon Web Services, "Amazon simple storage service (s3)," 2023, accessed: 2023-04-28. [Online]. Available: https://aws.amazon.com/s3/

[15] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[16] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," 2016. [Online]. Available: https://arxiv.org/abs/1607.04606

[17] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," Stanford Digital Library Technologies Project, Tech. Rep., 1998. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.1768

[18] "MITRE ATT&CK." [Online]. Available: https://attack.mitre.org/

[19] H. Kim, B. S. Lee, W.-Y. Shin, and S. Lim, "Graph anomaly detection with graph neural networks: Current status and challenges," 2022.

[20] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2021. [Online]. Available: https://doi.org/10.1109%2Ftkde.2021.3118815

[21] I. J. King and H. H. Huang, "Euler: Detecting network lateral movement via scalable temporal link prediction," *ACM Trans. Priv. Secur.*, mar 2023, just Accepted. [Online]. Available: https://doi.org/10.1145/3588771

[22] C. Bertero, "Experience report: Log mining using natural language processing and application to anomaly detection," *9th IEEE International Conference on Collaborative Computing*, 2017.

[23] A. Kundu, A. Sahu, K. Davis, and E. Serpedin, "A3d: Attention-based autoencoder anomaly detector for false data injection attacks," *Electric Power Systems Research, Vol 189*, 2020.

[24] (2022) Pytorch geometric. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/

[25] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[26] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" 2022.

[27] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *ICLR 2017*, 2017.

[28] J. Ren, F. Xia, A. N. Hoshyar, and C. C. Aggarwal, "Graph learning for anomaly analytics: Algorithms, applications, and challenges," 2022.

[29] A. D. Kent, "Comprehensive, Multi-Source Cyber-Security Events," Los Alamos National Laboratory, 2015.

[30] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, "Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2672–2681. [Online]. Available: https://doi.org/10.1145/3219819.3220024

[31] C. Liu, L. Sun, X. Ao, J. Feng, Q. He, and H. Yang, "Intention-aware heterogeneous graph attention networks for fraud transactions detection," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 3280–3288. [Online]. Available: https://doi.org/10.1145/3447548.3467142

[32] "torch.nn.BCEWithLogitsLoss," https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLog
accessed: [4 June 2023].

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[34] B. Wang, Y. Sang, Y. Zhang, S. Li, and X. Xu, "A longitudinal measurement and analysis study of mozi, an evolving p2p iot botnet," in *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2022, pp. 117–122.

[35] S. Gagnon, "Host-based anomaly detection using temporal graph autoencoder and sysmon logs [draft]," 2023.

[36] Javascript object notation (json) data interchange format. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8259

[37] W. McKinney and the Pandas Development Team, "pandas: A foundational python library for data analysis and statistics," 2010, version 1.3.3. [Online]. Available: https://pandas.pydata.org

[38] "Apache parquet," 2021, online; accessed 3 May 2023. [Online]. Available: https://parquet.apache.org/

[39] A. A. Hagberg, D. A. Schult, and P. J. Swart, "NetworkX: A python library for studying networks," Sep. 2008, online; accessed 3 May 2023. [Online]. Available: https://networkx.github.io/

[40] C. Correa and K.-L. Ma, *Visualizing Social Networks*, 03 2011, pp. 307–326.

[41] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire, "Rfc6335 - service name and transport protocol port number registry," Aug 1970. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6335.html

[42] M. Cotton, L. Vegoda, R. Bonica, and S. Cheshire, "Special-purpose ip address registries," Internet Engineering Task Force (IETF), RFC 6890, 2013. [Online]. Available: https://tools.ietf.org/html/rfc6890

[43] Institute of Electrical and Electronics Engineers (IEEE), "Organizationally unique identifier (oui) lookup database," https://regauth.standards.ieee.org/standards-ra-web/pub/view.html#registries, 2022, accessed: 2023-05-05.

[44] Fyodor, "Nmap: the network mapper," https://nmap.org/, 2023.

A Appendix A

1.1 Sysmon Event IDs

Sysmon events are given unique identifying numbers known as Event IDs. Each event is designed to give detailed information about host behaviour. These numbers are listed below.

Event ID	Description
1	Process creation event
2	A process changed a file creation time
3	Network connection successful
4	Sysmon service state changed
5	Process terminated
6	Driver loaded
7	Image loaded
8	Process created thread in another process
9	Process reading from drive using *RawAccessRead*
10	Process opened another process
11	File creation/overwrite event
12	Registry event (object created or deleted)
13	Registry event (value set)
14	Registry event (key or value renamed)
15	File stream created
16	Sysmon service configuration changed
17	Pipe connection created
18	Named pipe connected to service
19	WMI event filter activity is detected
20	Registration of WMI consumer
21	Consumer bound to WMI event filter
22	Process executed a DNS query
23	File deletion event, saved file to archive
24	System clipboard contents changed
25	Process tampering detected
26	File deletion event
27	Blocked creation of executable file
28	Detected block file shredding
255	Error within the Sysmon service

1.2 Graph Merge

The parallel execution of the *to_graph* algorithm described in listing 2 posed a challenge: How should the resulting sub-graphs be combined?

As of version 2.8.8, the `NetworkX` library implements *compose*, which returns the structural union of input graphs a and b. However, the composition only retains node and edge features unique to graph a.

This thesis contributes a generic function *merge*, which overcomes the limitations of *compose* by combining structural and feature-based data with user-defined parameters.

```python
from dataclasses import dataclass
from itertools import chain
from typing import Callable, TypeVar

import networkx as nx


@dataclass
class MergeableFeature:
    """A MergeableFeature is a named feature with a function describing how
    to merge instances of the feature via aggregation.

    Existing aggregation functions can be found in the stdlib under
    the operator module. For example:
        operator.add(x, y) is equivalent to x + y;
        operator.and_(x, y) is equivalent to x & y.
    """

    # Defines a private, generic type T internal to this class.
    __T = TypeVar("__T")
    # An aggregation function is defined by two arguments
    # of type T, returning the same type.
    __AggFunc = Callable[[__T, __T], __T]

    name: str
    aggregation_func: __AggFunc


def node_features(graph: nx.Graph) -> list[str]:
    """Extracts node features from a NetworkX graph"""
    return list(
        chain.from_iterable(n.keys()
        for *_, n in graph.nodes(data=True))
    )
```

```python
def edge_features(graph: nx.Graph) -> list[str]:
    """Extracts edge features from a NetworkX graph"""
    return list(
        chain.from_iterable(e.keys()
        for *_, e in graph.edges(data=True))
    )

def merge(
    a: nx.Graph,
    b: nx.Graph, *,
    node_features: list[MergeableFeature] | None = None,
    edge_features: list[MergeableFeature] | None = None,
) -> nx.Graph:
    """Returns the feature and structural union of A & B"""
    if not isinstance(a, nx.Graph) or not isinstance(b, nx.Graph):
        raise ValueError(
            "Input graphs should be of type nx.Graph, or subclassed type"
        )
    if not isinstance(a, type(b)):
        raise ValueError(
            f"Input graphs should be same type; got {type(a)} and {type(b)}"
        )

    # Returns the union of A and B-- does not update feature data
    a_compose_b = nx.compose(a, b)

    # Update node features common to union of A and B
    node_features = node_features if node_features is not None else []
    for feature in node_features:
        aggregate, name = feature.aggregation_func, feature.name
        feature_data = {
            n: aggregate(a.nodes[n][name], b.nodes[n][name])
            for n in a.nodes & b.nodes
        }
        nx.set_node_attributes(a_compose_b, feature_data, name)

    # Update edge features common to union of A and B
    edge_features = edge_features if edge_features is not None else []
    for feature in edge_features:
        aggregate, name = feature.aggregation_func, feature.name
        feature_data = {
            e: aggregate(a.edges[e][name], b.edges[e][name])
            for e in a.edges & b.edges
        }
        nx.set_edge_attributes(a_compose_b, feature_data, name)

    return a_compose_b
```

1.3 GAT Encoder

This code samples provides a Graph Attention Network (GAT) Encoder using the PyTorch framework and associated PyTorch Geometric library. The `GATEncoder` class is a custom implementation of the GAE Encoder, inspired by the GAE architecture proposed by Kipf [27]. It uses *GATv2Conv* layers for message passing, a more advanced version of the GAT (Graph Attention Network) layers that support dynamic attention calculations.

The `GATEncoder` class has two main components:

The constructor initializes three GATv2Conv layers (conv1, conv2, and conv3) with different input and output channels specified by a configuration object. The number of heads, edge dimensions, and dropout rates are also set based on the configuration.

This forward method describes a forward pass of the encoder, which takes three input arguments: the node features (x), adjacency list ($edge\_index$), and edge attribute matrix ($edge\_attr$). The forward method processes the input graph data through the GATv2Conv layers sequentially. The use of this modified GAT encoder encompasses the following four steps:

1. The input tensors are first converted to float.
2. The data is passed through the first GATv2Conv layer (conv1), followed by a ReLU (rectified linear unit) activation function.
3. The output is then passed through the second GATv2Conv layer (conv2) and another ReLU activation.
4. Finally, the data is passed through the third GATv2Conv layer (conv3) without any activation function, and the resulting tensor is returned.

The GATEncoder class implements a graph attention-based encoder for use in Graph Autoencoder models, which can be applied to various graph-based machine learning tasks, such as node classification, link prediction, or graph anomaly detection.

```python
import torch
from torch_geometric import GATv2Conv


class GATEncoder(torch.nn.Module):
    """A GAE encoder, modeled after Kipf (2017) GAE architecture, using
    GATv2Conv layers for message passing.
    """

    def __init__(self, in_channels, out_channels):
        self.conv1 = GATv2Conv(
            in_channels=in_channels,
            out_channels=config.hidden_layers,
            heads=config.num_heads,
            edge_dim=config.edge_dim,
            dropout=config.dropout,
            concat=True,
        )
        self.conv2 = GATv2Conv(
            in_channels=config.hidden_channels * config.num_heads,
            out_channels=config.hidden_channels // 2,
            heads=config.num_heads // 2,
            edge_dim=config.edge_dim,
            concat=True,
        )
        self.conv3 = GATv2Conv(
            in_channels=config.hidden_channels,
            out_channels=out_channels,
            edge_dim=config.edge_dim,
            concat=False,
        )

    def forward(
        self, x: torch.Tensor, edge_index: torch.Tensor, edge_attr: torch.
    Tensor
    ) -> torch.Tensor:
        """forward describes a forward pass of the encoder, yielding an N-
    dimensional
        tensor, where N is the number of nodes in the input graph.

        Args:
            x: node features, where the i'th entry corresponds to the
               feature vector for node i
            edge_index: adjacency list
            edge_attr: edge attribute matrix, where the i'th entry
               corresponds to the feature vector for edge i
        """
        x, edge_attr = x.float(), edge_attr.float()
        x = self.conv1(x, edge_index, edge_attr)
        x = torch.relu(x)
        x = self.conv2(x, edge_index, edge_attr)
        x = torch.relu(x)
        return self.conv3(x, edge_index, edge_attr)
```

1.4 Early Stopping Algorithm

The `EarlyStopper` class is designed to monitor a loss value of a machine learning model during training and stop the training process early if the loss doesn't improve over a given number of epochs. In the context of the GAT-GAE model, this refers to no forward progress on the accuracy of the input graph reconstruction.

The *patience* parameter determines the number of epochs to wait for the loss to improve before stopping the training process. The *delta* parameter determines the minimum percentage difference between the current validation loss and the minimum validation loss seen so far. If the relative difference between the current validation loss and the minimum validation loss is greater than *delta*, the *counter* attribute is incremented. If the counter exceeds *patience*, the *early_ stop* method returns True, indicating that the training process should be stopped.

```python
class EarlyStopper:
    """
    patience: int
        Determines the number of epochs to wait for the validation loss
        to improve before stopping the training process.
    delta: float
        Determines the percentage difference between the current validation
        loss and the minimum validation loss seen so far.
    """

    def __init__(self, patience=1, delta=0):
        self.patience = patience
        self.delta = delta
        self.counter = 0
        self.min_validation_loss = float('inf')

    def early_stop(self, validation_loss: float) -> bool:
        relative_difference = (
            validation_loss - self.min_validation_loss
        ) / self.min_validation_loss
        if validation_loss < self.min_validation_loss:
            self.min_validation_loss = validation_loss
            self.counter = 0
        elif relative_difference > self.delta:
            self.counter += 1
            if self.counter >= self.patience:
                return True
        return False
```

1.5 GAT-GAE Training Loop

This code defines a training loop for training a GAE model using an Adam optimizer and provided training data. The code is composed of the following key components:

1. model.train(): This sets the model into training mode, enabling features such as dropout and batch normalization that are specifically used during training.

2. neg_edge_index = negative_sampling(...): This line generates negative samples (i.e., edges that are not present in the graph) using the negative_sampling function. These samples are used to train the model on both positive (existing) and negative (non-existing) edges.

3. optimizer.zero_grad(): This clears any existing gradients from the previous iterations, ensuring that the gradients for the current iteration are calculated correctly.

4. z = model.encode(...): This line performs the forward pass through the model's encoder, which generates the embeddings (z) for each node in the graph based on the input features (train_data.x), the edge indices (train_data.edge_index), and the edge attributes (train_data.edge_attr).

5. link_logits = model.decode(...): This line calculates the logits (pre-activation values) for the edges using the model's decoder. It takes the node embeddings (z) and computes the logits for both positive and negative edge samples.

6. link_labels = get_link_labels(...): This line generates the ground truth labels for the positive and negative edge samples, with positive edges labeled as 1 and negative edges labeled as 0.

7. loss = F.binary_cross_entropy_with_logits(...): This calculates the loss using the binary cross-entropy with logits function, which computes the binary cross-entropy loss between the predicted logits (link_logits) and the ground truth labels (link_labels). This loss quantifies the difference between the model's predictions and the actual labels, guiding the model to learn the correct edge predictions.

8. loss.backward(): This performs backpropagation, calculating the gradients of the loss function with respect to the model's parameters.

9. optimizer.step(): This updates the model's parameters using the calculated gradients and the Adam optimizer.

```python
def train(
    model: torch.nn.Module,
    optimizer: torch.optim.Adam,
    train_data: Data
) -> float:
    model.train()

    neg_edge_index = negative_sampling(
        edge_index=train_data.edge_index,
        num_nodes=train_data.num_nodes,
        num_neg_samples=train_data.edge_index.size(1),
    )

    optimizer.zero_grad()

    z = model.encode(train_data.x, train_data.edge_index, train_data.edge_attr)
    link_logits = model.decode(z, train_data.edge_index, neg_edge_index)

    link_labels = get_link_labels(train_data.edge_index, neg_edge_index)
    loss = F.binary_cross_entropy_with_logits(link_logits, link_labels)

    loss.backward()
    optimizer.step()

    return loss
```

In summary, this code defines a training function for the GAT-GAE pipeline that takes the model, optimizer, and training data as inputs. It performs forward and backward passes, computes the loss, and updates the model's parameters using the Adam optimizer.

1.6 GAT-GAE Testing Loop

This code defines a *test* function for evaluating the GAT-GAE model on validation and test datasets using the AUC score as a performance metric. The function takes the model and data (validation and test) as inputs and returns the performance scores. The code is composed of the following key components:

1. @torch.no_grad(): This decorator disables gradient calculation during the evaluation process, which reduces memory consumption and speeds up computation since gradients are not needed for model evaluation.

2. model.eval(): This sets the model into evaluation mode, disabling features such as dropout and batch normalization that are only used during training.

3. pos_edge_index and neg_edge_index: These variables store the indices of positive and negative edge samples in the dataset, respectively.

4. z = model.encode(...): This line performs the forward pass through the model's encoder, which generates the embeddings (z) for each node in the graph based on the input features (data.x), the edge indices (data.edge_index), and the edge attributes (data.edge_attr).

5. link_logits = model.decode(...): This line calculates the logits (pre-activation values) for the edges using the model's decoder. It takes the node embeddings (z) and computes the logits for both positive and negative edge samples.

6. link_probs = link_logits.sigmoid(): This applies the sigmoid activation function to the logits, converting them into probabilities.

7. link_labels = get_link_labels(...): This line generates the ground truth labels for the positive and negative edge samples, with positive edges labeled as 1 and negative edges labeled as 0.

```python
from sklearn.metrics.roc_auc_score

@torch.no_grad()
def test(
    model: torch.nn.Module,
    val_data: Data,
    test_data: Data
) -> float:
    model.eval()
    perfs = []
    for data in [val_data, test_data]:
        pos_edge_index = data.pos_edge_label_index
        neg_edge_index = data.neg_edge_label_index

        z = model.encode(data.x, data.edge_index, data.edge_attr)
        link_logits = model.decode(z, pos_edge_index, neg_edge_index)
        link_probs = link_logits.sigmoid()

        link_labels = get_link_labels(pos_edge_index, neg_edge_index)

        perfs.append(
            roc_auc_score(link_labels.cpu(), link_probs.cpu())
        )
    return perfs
```

The *test* function returns the ROC-AUC scores for both validation and test datasets, which can be used to evaluate the model's performance in reconstructing the original graph.

This code defines an EarlyStopper class, which is a utility for implementing early stopping in the training process of machine learning models. Early stopping is a technique used to prevent overfitting by halting the training process if the validation loss does not improve beyond a certain threshold for a specified number of epochs.

```python
class EarlyStopper:
    """
    patience: int
        Determines the number of epochs to wait for the validation loss
        to improve before stopping the training process.
    delta: float
        Determines the percentage difference between the current validation
        loss and the minimum validation loss seen so far.
    """

    def __init__(self, patience=1, delta=0):
        self.patience = patience
        self.delta = delta
        self.counter = 0
        self.min_validation_loss = float('inf')

    def early_stop(self, validation_loss: float) -> bool:
        relative_difference = (
            validation_loss - self.min_validation_loss
        ) / self.min_validation_loss
        if validation_loss < self.min_validation_loss:
            self.min_validation_loss = validation_loss
            self.counter = 0
        elif relative_difference > self.delta:
            self.counter += 1
            if self.counter >= self.patience:
                return True
        return False
```

B Appendix B

2.1 Confusion Matrices

The binary classification results from all scenario/network pairings are listed in this section. All confusion matrices the format below.

NMAP	MoziDDoS
ExternalToEndpoint	EndpointToExternal

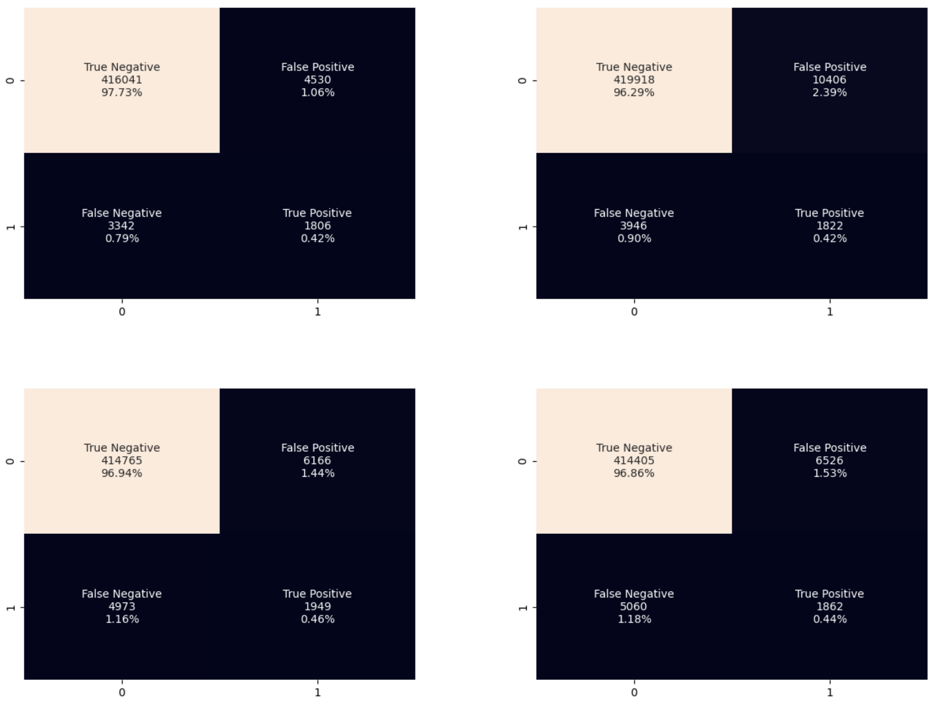


Figure 30: Confusion Matrices - Client A

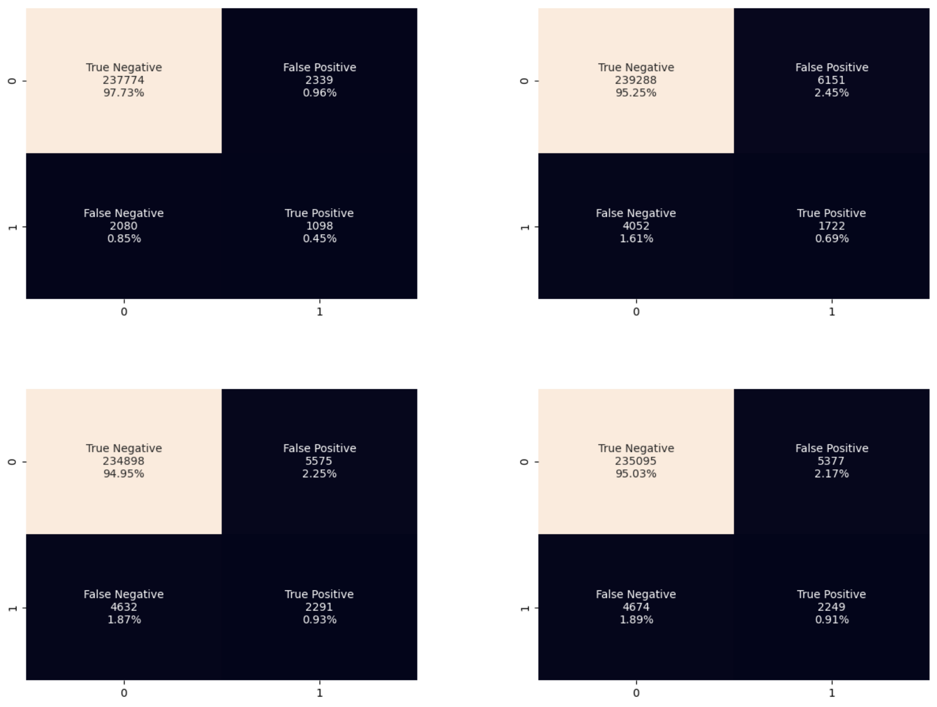


Figure 31: Confusion Matrices - Client B

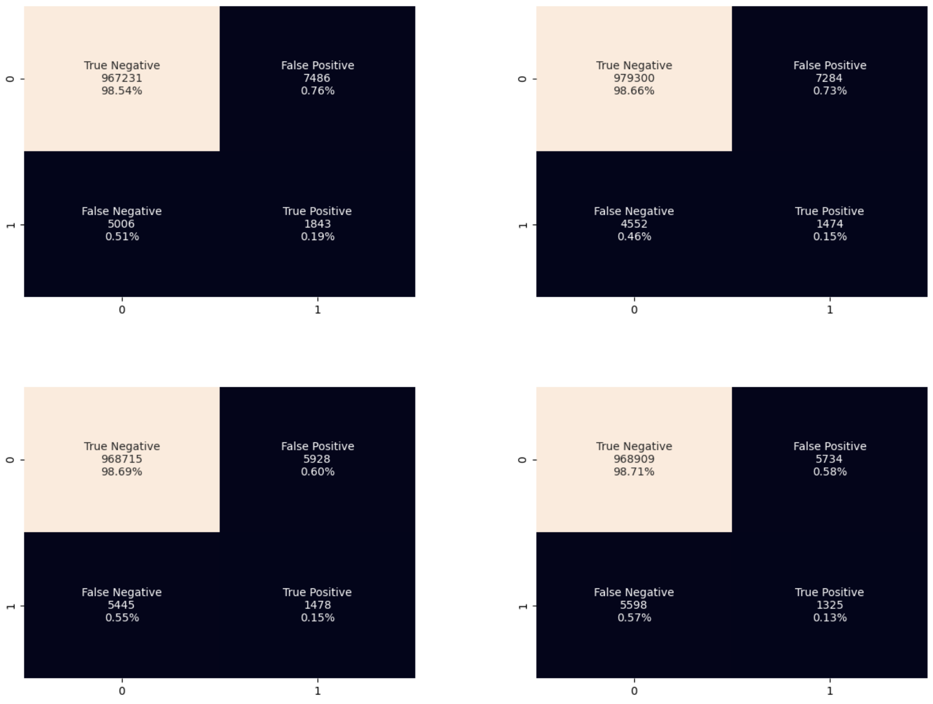


Figure 32: Confusion Matrices - Client C

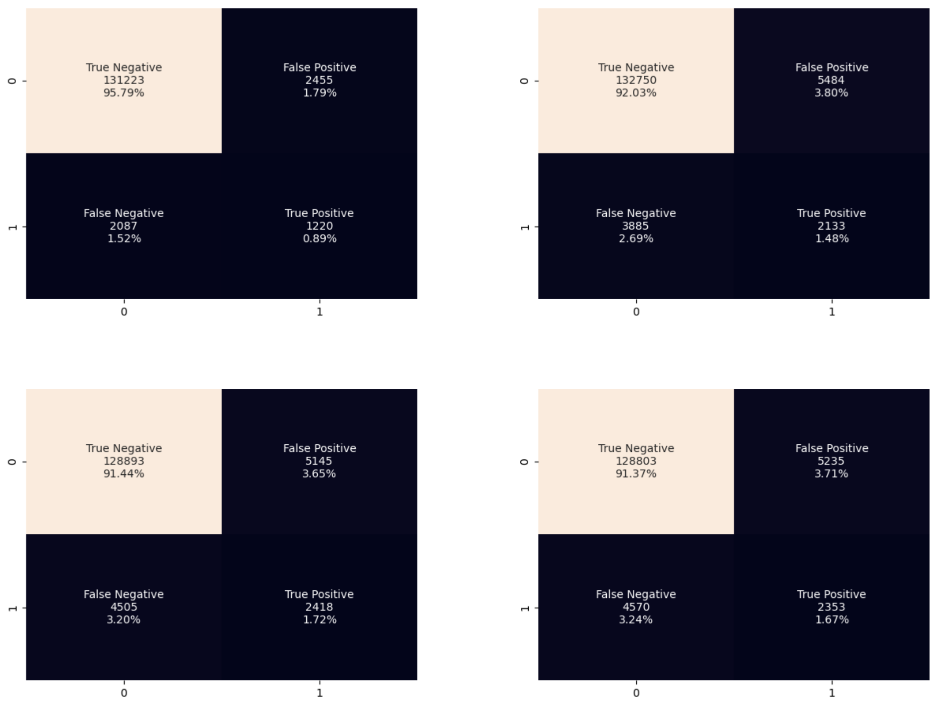


Figure 33: Confusion Matrices - Client D

2.2 Reconstruction Plots

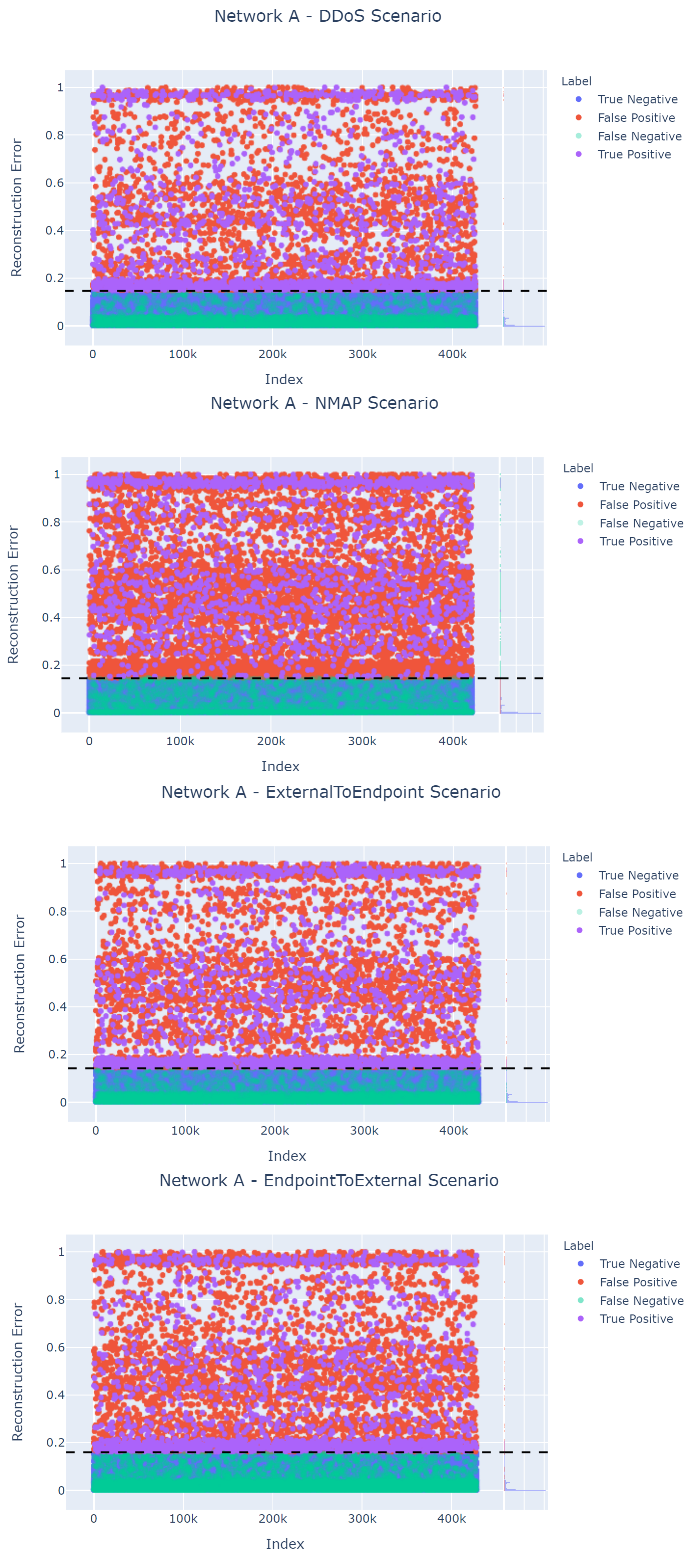


Figure 34: Reconstruction Plots - Client A

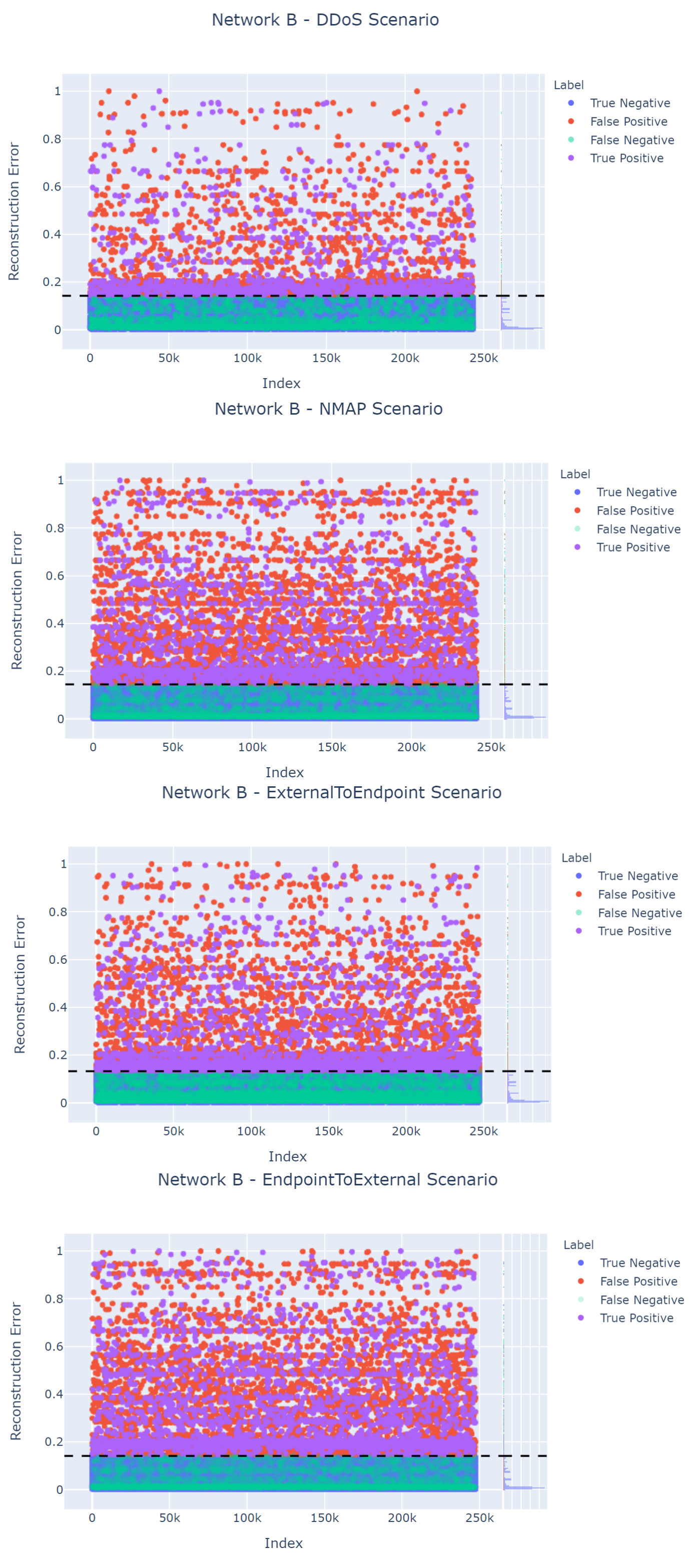


Figure 35: Reconstruction Plots - Client B

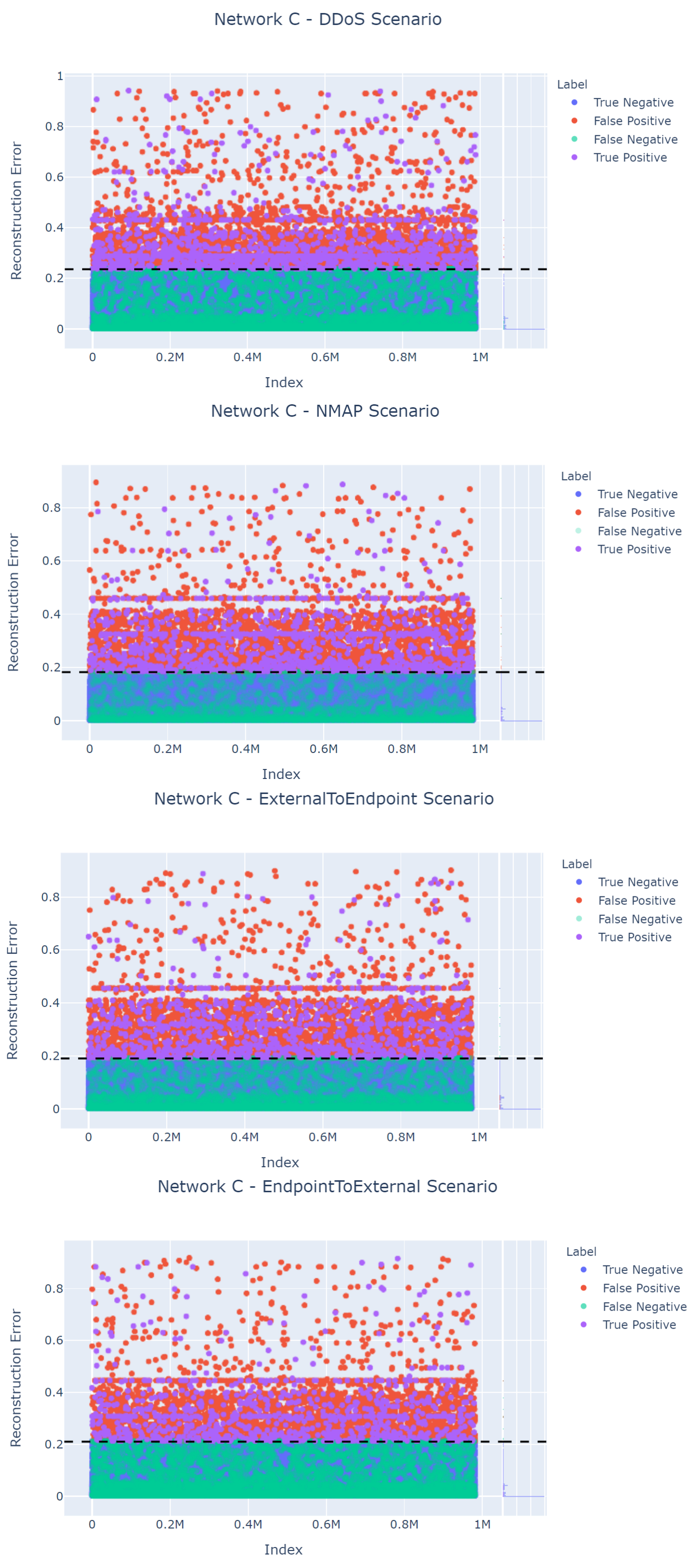


Figure 36: Reconstruction Plots - Client C

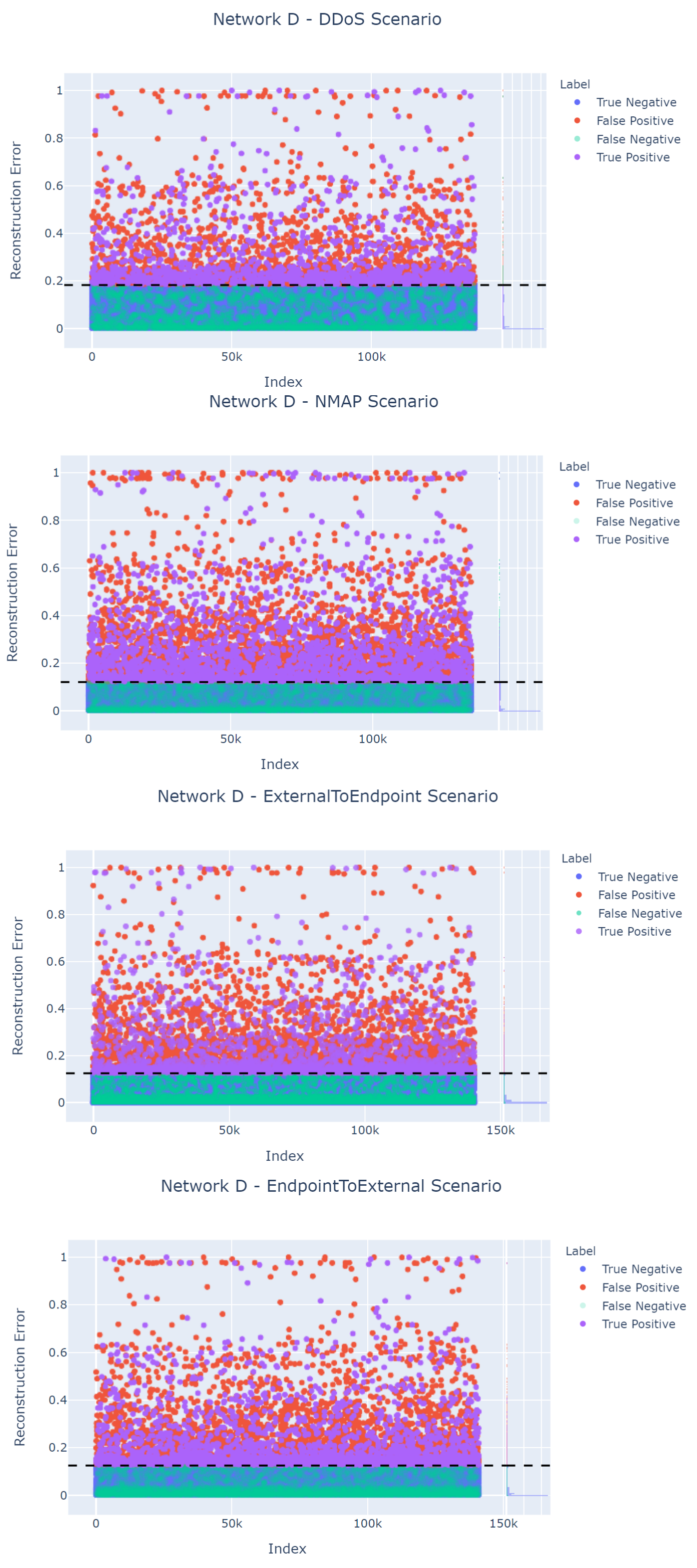


Figure 37: Reconstruction Plots - Client D