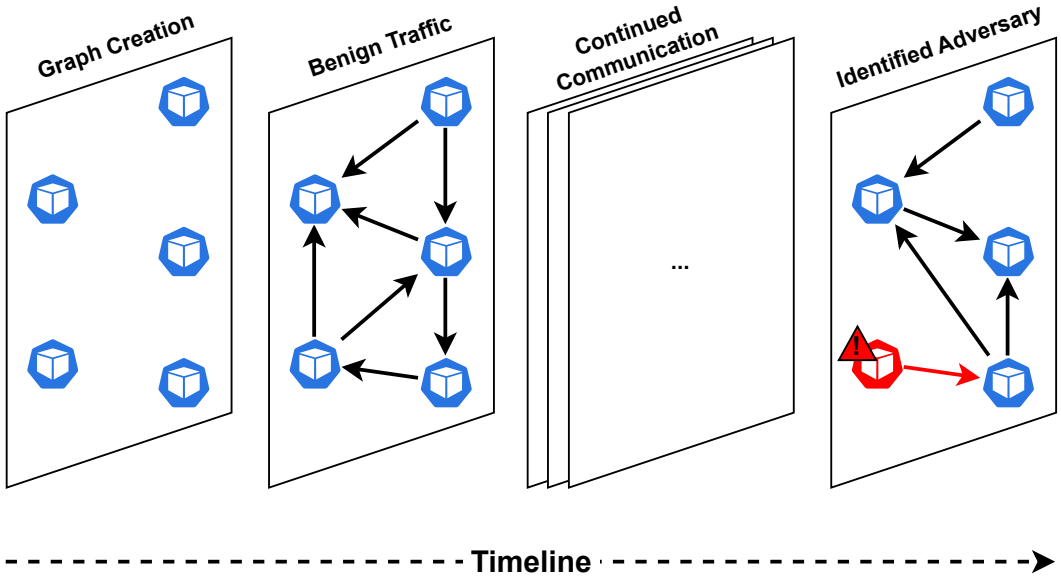
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Graph-based Anomaly Detection in 5G Core Networks

An Evaluation of Temporal Graph Neural Networks for detecting Multi-Step Attacks in Kubernetes-based 5G Core Networks.

Master's thesis in Computer science and engineering

GUSTAV SANDELL & ISAK NORDIN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Graph-based Anomaly Detection in 5G Core Networks

An Evaluation of Temporal Graph Neural Networks for detecting Multi-Step Attacks in Kubernetes-based 5G Core Networks.

GUSTAV SANDELL & ISAK NORDIN

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Graph-based Anomaly Detection in 5G Core Networks
An Evaluation of Temporal Graph Neural Networks for detecting Multi-Step Attacks in Kubernetes-based 5G Core Networks.
GUSTAV SANDELL & ISAK NORDIN

Supervisor: Atmane Ayoub Mansour Bahar, Department of Computer Science and Engineering
Advisors: Mikael Eriksson, *Ericsson* & Gyanendra Singh, *Ericsson*
Examiner: Romaric Duvignau, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Temporal Graph Representation of Kubernetes Cluster with Adversary.

Typeset in LaTeX
Gothenburg, Sweden 2026

Graph-based Anomaly Detection in 5G Core Networks
An Evaluation of Temporal Graph Neural Networks for detecting Multi-Step Attacks in Kubernetes-based 5G Core Networks.
GUSTAV SANDELL & ISAK NORDIN
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The increasing adoption of service-based and distributed architectures in 5G Core (5GC) networks has introduced new challenges for traditional intrusion detection systems (IDS), which often struggle to identify sophisticated, multi-stage attacks hidden within large volumes of network traffic. While many preventative measures exist to secure system perimeters, there are still scenarios where a malicious actor could gain access to a network.

At the same time, Graph Neural Networks (GNNs) have demonstrated strong capabilities in modeling complex relationships and dependencies in networked graph-structured data, making them a promising approach for detecting advanced cyber threats in highly interconnected systems.

This thesis investigated the capability of GNN-based intrusion detection to identify multi-step attacks in the 5GC. Due to the lack of publicly available datasets containing such attacks in internal 5GC systems, a new dataset was developed by combining generated benign network traffic with a simulated multi-step attack targeting production components of the 5GC environment.

A complete detection framework was designed and evaluated, including data preprocessing, feature extraction, graph construction, and a GNN architecture. The model architecture consists of a Memory Module, Graph Convolutional Layers and a Edge-level Classification Head. Different architectures and configurations of the model were trained and tested on the data, with the highest stable scoring model having a median F1 Score of 99.55%, showing that it is possible for a graph-based model to detect attacks in a production-like 5GC test environment using simulated network data and a simulated multi-step attack.

Keywords: Temporal Graph Neural Networks (TGN), Graph Neural Networks (GNN), Anomaly Detection, Intrusion Detection Systems (IDS), 5G Core Networks, Kubernetes, Cloud-Native Networks, Multi-Step Attack Detection.

Acknowledgements

We would like to express our sincerest gratitude to our supervisor, Atmane Ayoub Mansour Bahar, for his guidance and support throughout this thesis project. His expertise in Graph Neural Networks in Cybersecurity has been invaluable to our work. His insights, constructive feedback, and willingness to share his knowledge have greatly contributed to both the quality of this thesis and our understanding of the field.

We would also like to thank our examiner, Romaric Duvignau, for his valuable feedback, thoughtful discussions, and support throughout the thesis process. His comments and suggestions have helped us to strengthen and refine our work.

Furthermore, we would like to extend our appreciation to our advisors at *Ericsson*, Gyanendra Singh and Mikael Eriksson, for their support and guidance during this project. Their expertise in system architecture and cybersecurity helped shape the direction of this work. Their practical insights and experience provided valuable perspectives that complemented the academic aspects of the thesis.

We would also like to thank Sathya Prakash Kadhirvelan, Joakim Lisander and Peter Tranberg at *Ericsson* for their practical support and assistance in designing a multi-step attack and data generation during the project.

We are also grateful to everyone who has contributed, directly or indirectly, to this work through discussions, advice, and support throughout the course of this thesis.

Gustav Sandell & Isak Nordin, Gothenburg, July 2026

Contents

Acronyms

5GC 5G Core. 1

AMF Access and Mobility Management Function. 6
APT Advanced Persistent Threat. 1
AWS Amazon Web Services. 31

CTDG Continuous-Time Dynamic Graph. 23

EC2 Elastic Compute Cloud. 31

GAE Graph Autoencoder. 12
GAT Graph Attention Network. 9
GCN Graph Convolutional Network. 8
GNN Graph Neural Networks. 1
GRU Gated Recurrent Unit. 11
GTN Graph Transformer Network. 9

HTTP Hypertext Transfer Protocol. 6

IDS Intrusion Detection System. 1

kNN k-Nearest Neighbors. 15

MGNN Multi-Graph Neural Network. 15
ML Machine Learning. 2
MLP Multilayer Perceptron. 16

NF Network Function. 5

PIDS Provenance-based Intrusion Detection Systems. 13

ReLU Rectified Linear Unit. 8
REST Representational State Transfer. 6
RNN Recurrent Neural Network. 17

SMF Session Management Function. 6
SSH Secure Shell. 21
ST-GNN Spatio-Temporal Graph Neural Networks. 17

TGNs Temporal Graph Networks. 10

UE User Equipment. 6
UPF User Plane Function. 17

List of Figures

List of Tables

1
Introduction

In recent years, network infrastructures have become increasingly service-based and distributed, particularly in the context of 5G Core (5GC) networks. This has introduced additional challenges for traditional signature- and statistics-based intrusion detection methods, as they often struggle to detect the subtle, multi-stage, and stealthy attack patterns found in modern networks due to their limited ability to capture hidden interactions and long-range dependencies within system activity.

In parallel, graph-oriented approaches, particularly Graph Neural Networks (GNN), have shown strong potential for modelling the complex dependencies, event flows, and causal relationships present in modern network systems(Keramatfar et al. [2022]). GNNs have shown proficiency when it comes to modelling relational graphs from large datasets, and could therefore be appropriate for modelling the connections in large-scale telecom networks as well.

Cyberattacks are becoming more advanced, with sophisticated adversaries like nation states with access to considerable resources targeting high-profile actors such as governments and corporations, as well as critical infrastructure (Yuldoshkhujaev et al. [2025]). Common initial attack vectors include social phishing and malicious documents, with many attacks not relying on zero-day vulnerabilities. This requires Intrusion Detection Systems (IDSs) to not only protect against attacks from outside the system, but also monitor the internal systems for threats that have gained a foothold within.

As more services and systems become digitalized, reliable and secure connectivity is becoming increasingly important, where 5GC networks is one such system. Therefore, attacks against 5GC networks could have considerable consequences, and security administrators need to be prepared to mitigate threats as soon as they appear. Due to the vast amount of data flowing through the 5GC, there can be significant delays between an attack occurring and administrators detecting the threat. Consequently, automated IDS solutions are essential for enabling rapid response and minimizing downtime and service disruption.

Advanced cyberattacks, such as Advanced Persistent Threats (APTs), are difficult to detect using traditional rule-based IDSes, as these multi-step attacks are able to bypass network rules and firewalls by acting like benign users. Therefore, alternative approaches have been taken to combat these attacks and maintain security.

Currently, IDSes are adopting graph-based Machine Learning (ML) solutions to improve detection of advanced attacks and stealthy intrusions, as they represent system activity as dynamic graphs and learning patterns of normal and abnormal behaviour across processes, hosts, and services. This enables the system to identify subtle anomalies, multi-step attack paths, and previously-unseen intrusion strategies; capabilities that are particularly relevant for the highly interconnected and microservice-oriented architecture of the 5GC.

Graph-based IDS methods, and IDS solutions for 5GC networks have both been developing rapidly in recent years, but there is still very little work combining the two approaches. Therefore, this project aims to evaluate Graph-based IDS for 5G Core systems, and investigates how such models could be integrated into *Ericsson*'s current infrastructure.

1.1 Problem Statement

Multi-step attacks are difficult to detect in 5GC networks due to the high volume of internal traffic and the stealthy nature of these attacks. Graph Neural Networks have proven effective in detecting Advanced Persistent Threats in domains such as operating systems; however, their effectiveness in identifying multi-step attacks within 5GC environments remains largely unexplored. In particular, it is unclear whether network communication data alone can be used to distinguish between benign and malicious behaviour in such networks. Therefore, the primary objective of this thesis is to investigate whether GNNs can effectively detect multi-step attacks in 5GC networks by constructing graphs from internal packet flow data.

Since there are currently no existing datasets containing multi-step attacks executed within the 5GC, a new dataset must be created consisting of both simulated benign network traffic and executed multi-step attack scenarios. The dataset should closely resemble real-world production traffic while still enabling the model to effectively distinguish between benign and anomalous behaviour. Consequently, the proportion of attack traffic may be higher than what would typically occur in a real-world 5GC environment in order to provide the model with a sufficiently balanced dataset for training.

1.2 Contribution

This thesis contains a set of contributions, some of which relate to previous research into GNNs in IDS systems as well as 5GC IDS systems, and some novel such as a dataset containing a multi-step attack in a simulated 5GC environment. The contributions are as follows:

- Created a model for detecting multi-step attacks from packet flows in a 5GC environment, containing preprocessing and feature extraction from packet logs, as well as different frameworks for GNN models which were evaluated and tested on their accuracy in distinguishing malicious packets from benign pack-

ets. Different modules for the model was tested, including different GNN models as well as memory embedding modules.
- Created a dataset where benign network data was generated using a simulated stability test provided by *Ericsson*, and combined with a multi-step attack performed inside this environment targeting components of the simulated network.

1.3 Research Questions

This thesis aims to evaluate and answer two research questions, but the main focus of the thesis is the first research question:

1. Can Graph Neural Networks effectively distinguish between normal and malicious activity patterns in (real-world) 5G Core network graphs.
2. What would be the potential of graph-based models for proactive intrusion detection and threat prediction in 5G Core network monitoring systems.

1.4 Challenges

This thesis contains a set of challenges:

1. **Data-related Challenges**
   (a) **Preprocessing and graph construction at scale:** Converting raw 5GC packet flows with respect to system architecture, into structured graph representations (nodes, services, hosts; edges for flows, dependencies, or temporal links) is complex and requires careful design.
   (b) **Handling diverse data types and high throughput:** 5GC networks generate heterogeneous data streams with high volume and velocity. Efficient storage, sampling, and graph updates are required to train models in a reasonable timeframe. While GNNs are proven to be able to identify APTs with data from the OS layer (system calls) (Bahar et al. [2025]), it is unknown if they are able to identify APTs with data from the Network layer (packet flows), due to the difference in information from the data.
2. **Model and Algorithmic Challenges**
   (a) **Scalability and efficiency:** GNN models must handle large graphs with thousands of nodes and millions of edges while still allowing reasonably fast training and inference.
   (b) **Anomaly detection and class imbalance:** Even with access to benign and malicious traffic, rare or subtle attacks (like multi-step APTs) are challenging to detect. The model must distinguish fine-grained anomalies without generating too many false positives.
   (c) **Generalization to unseen patterns:** Models must be robust to new attack types or network configuration changes while still leveraging prior knowledge from historical deployments.

2

Background

This section contains the theory behind the 5G Core Networks architecture, Graph Neural Networks, Threat Model, and related utilities.

2.1 5G Core Networks

The 5GC constitutes the central component of the 5G architecture and is responsible for managing the core functionality of the mobile network. It acts as the primary hub for communication and control, handling tasks such as traffic routing, user authentication, mobility management, session management, data storage, and signaling across the network ([5gc, 2025]). 5GC adopts a service-based architecture (SBA), where Network Functions (NFs) communicate through standardized service interfaces. This architectural design improves scalability, flexibility, and interoperability, while enabling advanced 5G features such as network slicing, ultra-low latency communication, and massive IoT connectivity. The service-based 5GC architecture can be viewed in figure 2.1

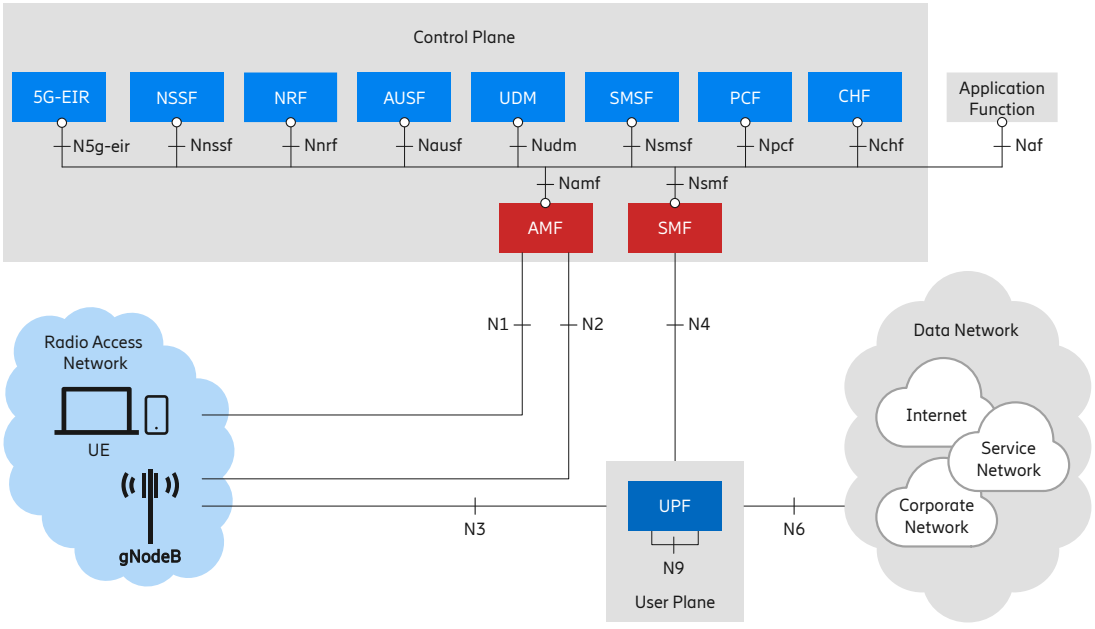


Figure 2.1: 5GC Service-Based Architecture by *Ericsson*

The User Equipment (UE), consisting of a Mobile Station and a USIM, connects to the Radio Access Network (RAN), which in turn interfaces with the 5GC. Standards for the 5GC, specified by 3GPP [2022], define requirements for services, architecture, and protocol stacks. Within the service-based architecture, the core network is composed of multiple NFs, where each NF represents a collection of related services and functionalities. Together, these network functions form the complete 5GC system (Rommer et al. [2019]).

Communication between NFs in the 5GC follows the HTTP REST paradigm, where services are exposed and accessed through RESTful APIs. This service-oriented communication model increases modularity and enables flexible deployment of network services. However, it also introduces additional security challenges, as malicious activity can propagate between interconnected services and interfaces inside the core network.

The network function that this thesis focuses on is the Access and Mobility Management Function (AMF). The AMF is responsible for handling registration, connection, reachability, and mobility management of UE devices. It enables identification and authentication of users and interacts closely with both the radio access network and connected devices (such as phones). Although the AMF does not directly handle session management, it plays a central role in signaling procedures within the 5G network.

For session-related communication, the AMF forwards signaling messages between UE devices and the Session Management Function (SMF), which is responsible for session establishment and management. Since the AMF communicates with nearly all other network functions through service-based interfaces, it is involved in the majority of signaling flows within the 5GC. This central position makes the AMF an important observation point for monitoring network behaviour and detecting anomalous or malicious activity.

2.2 Kubernetes Clusters

Kubernetes is an open-source platform for managing containerized workloads and services. Each Kubernetes container has its own filesystem, CPU share and memory. Kubernetes Containers can be connected through Kubernetes Clusters, which can consist of either physical or virtual machines.

A Kubernetes cluster consists of a control plane and one or more worker nodes. The control plane manages the overall state of the cluster, including scheduling workloads, maintaining desired system state, and handling cluster-wide coordination. Worker nodes execute application workloads in the form of pods, which are the smallest deployable units in Kubernetes and may contain one or more containers (Kubernetes Authors [2025]). In production environments, the control plane and worker nodes are typically distributed across multiple machines to ensure scalability

and fault tolerance. Kubernetes pods are vulnerable to security risks such as misconfiguration and privilege escalation, and this could compromise the Kubernetes cluster. The 5GC is designed and operated as Kubernetes clusters, and a security solution for these Kubernetes clusters is presented in Flon and Sulaiman [2025].

2.3 Graph Neural Networks

Graph Neural Networks (GNNs) are deep learning methods designed to operate directly on graph-structured data (Liu and Zhou [2022]). A graph consists of a set of nodes and edges, where edges represent relationships between pairs of nodes. Due to their ability to naturally model relational data, graphs are widely used across many scientific domains, which has contributed to the increasing attention given to GNNs in recent years (Keramatfar et al. [2022]) In particular, several studies have explored the application of GNNs in cybersecurity contexts, including Mitra et al. [2024], Bahar et al. [2025], Holmkvist Bergqvist [2025] and Jia et al. [2024].

In a graph, each edge (i, j) represents a relationship between node i and node j. Both nodes and edges may also be associated with feature vectors that describe their respective attributes. Most GNNs learn node representations through a process known as message passing (illustrated in figure 2.2), where information from a node's neighbouring nodes is iteratively aggregated and combined with the node's own feature representation. Through this iterative aggregation process, each node embedding is updated to incorporate information from its local neighbourhood. As additional layers are added, information is propagated further through the graph, allowing nodes to indirectly incorporate information from more distant parts of the structure. This effectively increases the *receptive field*[1] of each node, enabling the embedding to capture progressively larger and more complex substructures of the graph.

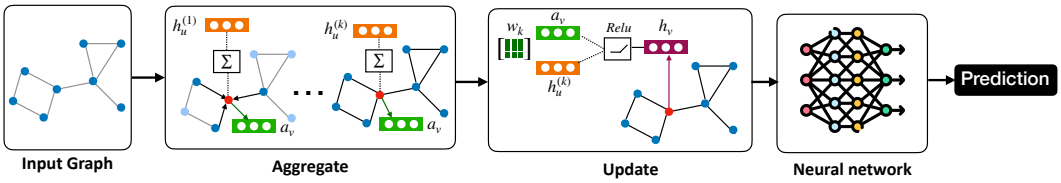


Figure 2.2: Example of message-passing GNN Architecture from Mitra et al. [2024].

Equation 2.1 presents a general formulation of a GNN layer as a message-passing update rule, where each node updates its representation by combining its current embedding with aggregated information from its neighbours:

$$h_i^{(l+1)} = \sigma\left(W^{(l)} h_i^{(l)} + \sum_{j \in \mathbf{N}(i)} W_{\text{neigh}}^{(l)} h_j^{(l)}\right) \tag{2.1}$$

[1]The receptive field represents the area of the graph that each node receives inputs from

Where $h_i^{(l)}$ represents the feature vector (or embedding) of node i at layer l, $\mathbf{N}(i)$ denotes the set of neighbouring nodes of node i, $W^{(l)}$ and $W_{\text{neigh}}^{(l)}$ are learnable weight matrices applied to the node's own representation and its neighbours respectively, and $\sigma(\cdot)$ is a non-linear activation function such as Rectified Linear Unit (ReLU). The first term preserves the node's current representation, while the second term aggregates and transforms information from its local neighbourhood.

In temporal contexts, where the graph structure includes a time domain, GNNs can generally be applied to two types of graph settings (Rossi et al. [2020]):

1. **Discrete-time dynamic graph.**
   The graph evolves over time as a sequence of graph snapshots, where each snapshot represents the state of the whole graph at a specific time interval.
2. **Continuous-time dynamic graphs.**
   The graph is represented as a stream of individual time-stamped events, where nodes and edges can be added, altered or removed continuously over time.

There exist several different taxonomies and architectures of GNN, with taxonomies including Convolutional GNNs, Recurrent GNNs, Spatial GNNs and Spectral GNNs (Li et al. [2021]). Some common GNN architectures are presented below.

2.3.1 Graph Convolutional Networks

Graph Convolutional Network (GCN)s, introduced by Kipf and Welling [2017] is a variant of Convolutional Neural Networks operating on graphs. GCNs are used for semi-supervised learning, by classifying nodes in a graph where labels are only available in a small subset of nodes. A multi layer GCN has the layer propagation rule:

$$H^{(l+1)} = \sigma\Big(\bar{D}^{-\frac{1}{2}}\bar{A}\bar{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\Big) \tag{2.2}$$

Where $\bar{A}$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections, $W^{(l)}$ is a layer-specific trainable weight matrix. σ is an activation function, $H^{(l)} \in \mathbb{R}^{N\times D}$ is the matrix of activations in the l^{th} layer. The layer propagation rule is based on a first-order approximation of spectral convolution on graphs, and GCNs outperformed several baseline methods while being computationally efficient (Kipf and Welling [2017]).

2.3.2 Graph Sample and Aggregate

Graph Sample and Aggregate (GraphSAGE), introduced by Hamilton et al. [2018] is a general *inductive*[2] framework for generating node embeddings for previously unseen data from node features. GraphSAGE trains a set of aggregator functions that learn to aggregate feature information from a node's local neighbourhood instead of training an embedding vector for each node. An illustration of the sample and aggregate approach can be seen in Figure 2.3. At each iteration, the nodes in the graph aggregate information from their neighbours and with each iteration they

[2]Inferring rules and patterns from examples

incrementally gain information from further reaches of the graph, which allows the algorithm to effectively generate representations for unseen nodes, as well generalize to completely unseen graphs in a multi-graph dataset.

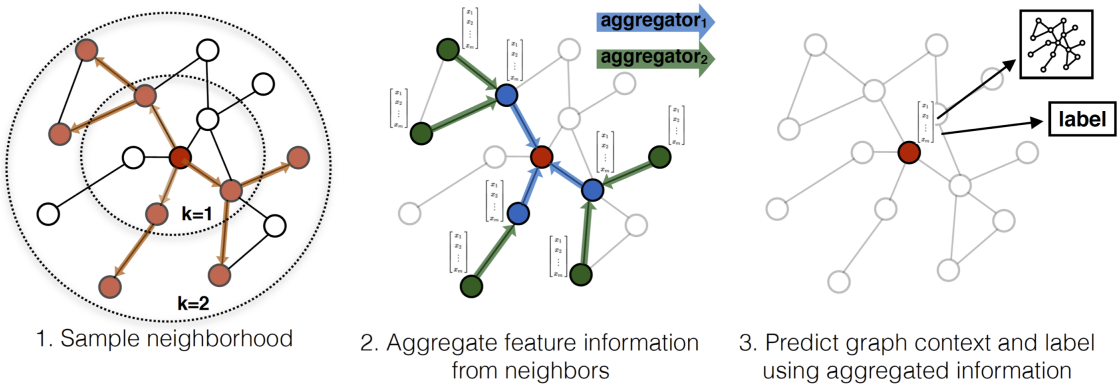


Figure 2.3: Illustration of the Sample and Aggregate approach of GraphSage from Hamilton et al. [2018]

In Hamilton et al. [2018], the GraphSage algorithm is shown to outperform baseline approaches such as Logistic Regression (Pedregosa et al. [2018]) and DeepWalk (Perozzi et al. [2014]) in both a supervised and unsupervised setting, with the unsupervised GraphSage being competitive with the supervised GraphSage in F1-score.

2.3.3 Graph Attention Networks

Graph Attention Network (GAT)s, introduced in Veličković et al. [2018], enables specifying weights to nodes without requiring costly matrix operations (such as inversion) or knowing the specific graph structure upfront. A GAT computes the hidden representation of each node in a graph by attending over the neighbors, through self-attention. This operation is efficient since its parallelizable over node-neighbour pairs and it can be applied to nodes with different degrees by specifying arbitrary weights to the neighbours. The input to a GAT layer is a set of node features:

$$h = \vec{h_1}...\vec{h_N}, \quad \vec{h_i} \in \mathbb{R}^F \tag{2.3}$$

where N is the number of nodes and F is the number of features in each node(Veličković et al. [2018]). The output of the layer is a new set of node features h'.

2.3.4 Graph Transformer Network

Graph Transformer Network (GTN), introduced in Shi et al. [2021], is a unified message passing model which performs feature propagation and label propagation within a graph transformer. GTN first adopts a Graph Transformer and takes feature and label embedding as input for propagation, and then trains using a masked label prediction strategy, where a subset of the input label information is masked at

random and then predicted, which reduces the risk of overfitting.

In Shi et al. [2021], GTNs are shown to outperform both GCNs and GATs on the open dataset Open Graph Benchmark (Hu et al. [2021]).

2.4 Temporal Graph Networks (TGNs)

Temporal Graph Networks (TGNs) was introduced by Rossi et al. [2020], and is a modular GNN framework for learning on continuous-time dynamic graphs represented as sequences of time-stamped edge events between nodes. TGN handles two types of events:

1. Node-wise events, which are node creation and deletion, as well as updates of node features.
2. Interaction events, which are directed temporal edges between nodes.

Given a stream of temporal events as input, TGN iteratively update and maintain dynamic node embeddings, producing time-dependent representations of nodes that capture the temporal evolution of nodes in the graph. The internal processes of TGN is outlined in figure 2.4 and consists of:

Message Function

The message function acts as the first transformation step in TGN, encoding raw event information together with the current node memories into message representations that can later be used by the memory update mechanism.

For a node-wise event $v_i(t)$, a single message is computed for node i

$$\mathbf{m}_i(t) = msg_n\Big(\mathbf{s}_i(t^-), t, \mathbf{v}_i(t)\Big) \tag{2.4}$$

where $\mathbf{s}_i(t^-)$ is the memory of node i immediately before the event time t, and t denotes the timestamp at which the event occurs. The function msg is a learnable message function, for example an MLP.

For interaction events between two nodes, two messages are computed. For an event e_{ij} between source i and target j at time t:

$$\begin{aligned} \mathbf{m}_i(t) &= msg_s\Big(\mathbf{s}_i(t^-), \mathbf{s}_j(t^-), \Delta t, \mathbf{e}_{ij}(t)\Big), \\ \mathbf{m}_j(t) &= msg_d\Big(\mathbf{s}_j(t^-), \mathbf{s}_i(t^-), \Delta t, \mathbf{e}_{ij}(t)\Big) \end{aligned} \tag{2.5}$$

where Δt represents the elapsed time since the previous update or interaction involving the corresponding node, enabling the model to capture temporal dynamics between events.

This thesis will explore solutions with msg implemented as either *identity* (id) or an MLP, where id corresponds to a simple concatenation of the inputs.

Message Aggregation

As multiple events may involve the same node within a temporal batch, the message aggregation function combines the set of encoded messages associated with a node into a single aggregated representation. The purpose of the aggregation step is to compress potentially multiple event-dependent message embeddings into one message vector that can subsequently be used by the memory updater.
For node i at time t, the aggregation mechanism is defined as

$$\bar{\mathbf{m}}_i(t) = agg\Big(\mathbf{m}_i(t_1), \ldots, \mathbf{m}_i(t_b)\Big) \tag{2.6}$$

where $t_1, \ldots, t_b \leq t$ denote the timestamps of messages generated for node i prior to or at time t, and agg is an aggregation operator applied over the message set. The aggregated message $\bar{\mathbf{m}}_i(t)$ represents the condensed temporal information associated with node i over the considered interval.

This thesis explores two aggregation strategies:

- *Most recent message*, where only the latest generated message for node i is retained:
$$\bar{\mathbf{m}}_i(t) = \mathbf{m}_i(t_b) \tag{2.7}$$
where $t_b = \max\{t_1, \ldots, t_b\}$.
- *Mean aggregation*, where the aggregated message is computed as the element-wise mean over all messages associated with node i:
$$\bar{\mathbf{m}}_i(t) = \frac{1}{b} \sum_{k=1}^{b} \mathbf{m}_i(t_k) \tag{2.8}$$

Memory Update

The memory update mechanism updates the persistent node state using the aggregated messages generated from recent events. The objective of the update step is to incorporate newly observed temporal interactions into the node memory while preserving relevant historical information.
For node i at time t, the memory update mechanism is defined as

$$\mathbf{s}_i(t) = mem\Big(\bar{\mathbf{m}}_i(t), \mathbf{s}_i(t^-)\Big) \tag{2.9}$$

where $\bar{\mathbf{m}}_i(t)$ is the aggregated message for node i at time t, $\mathbf{s}_i(t^-)$ denotes the memory state immediately before the update, and mem is a learnable memory update function. The resulting state $\mathbf{s}_i(t)$ represents the updated memory after incorporating information from the current event(s).

For interaction events between two nodes i and j, the memories of both nodes are updated independently using their corresponding aggregated messages:

$$\begin{aligned} \mathbf{s}_i(t) &= mem\Big(\bar{\mathbf{m}}_i(t), \mathbf{s}_i(t^-)\Big), \\ \mathbf{s}_j(t) &= mem\Big(\bar{\mathbf{m}}_j(t), \mathbf{s}_j(t^-)\Big) \end{aligned} \tag{2.10}$$

This thesis explores implementations where mem is parameterized as a Gated Recurrent Unit (GRU), enabling gated recurrent updates of node memories over time.

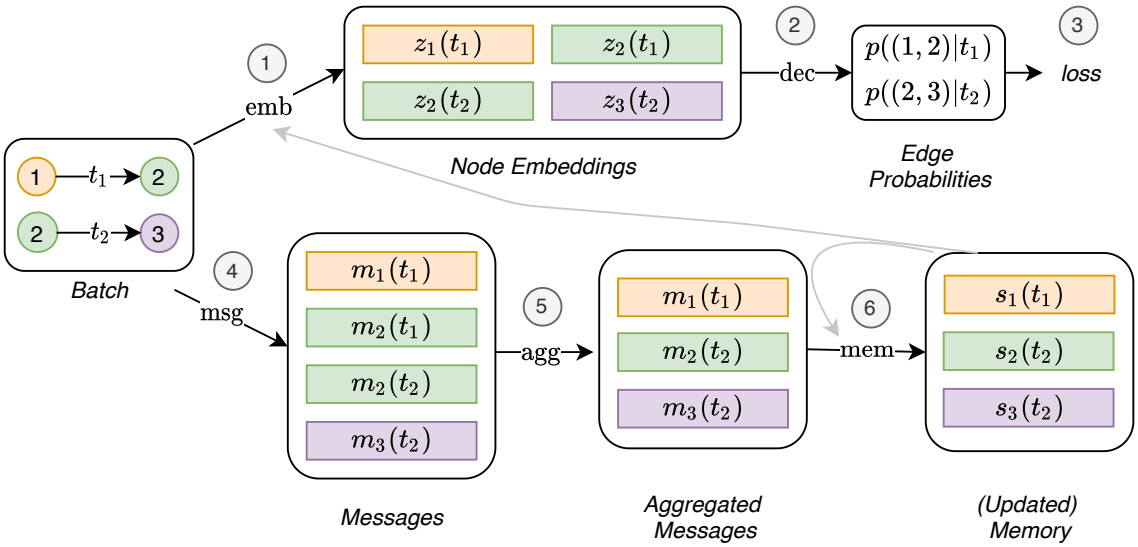


Figure 2.4: Processing pipeline of the TGN architecture for batches of temporal interactions, shown together with an example edge decoder. *Top:* the embedding module computes node representations from the temporal interaction graph and the current node memory states (1). *Bottom:* the observed interactions are then used to update node memories via the message function, aggregation, and memory update operations (4, 5, 6). (Rossi et al. [2020]).

Embedder Module

The temporal embedding $z_i(t)$ of node i at time t is generated by the embedding module. The memory of i is only updated when i is involved in an event, which means that the memory of i might become stale if there is an absence of events involving i for a long time. The embedding module is of the form:

$$z_i(t) = emb(i,t) = \sum_{j \in n_i^k([0,t])} h(s_i(t), s_j(t), e_{ij}, v_i(t), v_j(t)), \tag{2.11}$$

Where h is a learnable function, $s_i(t), s_j(t)$ is the memory of nodes i, j at time t, e_{ij} is an interaction event between i and j, and $v_i(t), v_j(t)$ are node-wise events at i, j at time t.

In Rossi et al. [2020], TGNs are shown to outperform other state-of-the-art models such as GAT and Graph Autoencoder (GAE) in dynamic node classification and future edge prediction.

2.5 Graph Neural Networks in Cybersecurity

GNNs have gained significant attention in cybersecurity, especially for intrusion detection tasks. Conventional IDSes typically depend on rule-based or signature-based methods (Han et al. [2020]), which are often inadequate when confronting rapidly evolving attack techniques and previously unseen threats. By contrast, GNNs can capture complex dependencies within networked data, allowing them to uncover subtle and hidden patterns.

Research shows that GNNs can aid in breaking each stage of the Lockheed Martin Cyber Kill Chain 2.5, one of the most renowned attack life cycles (Mitra et al. [2024]). In Mitra et al. [2024], a comprehensive amount of GNN implementations are evaluated corresponding to each stage in the aforementioned attack cycle.

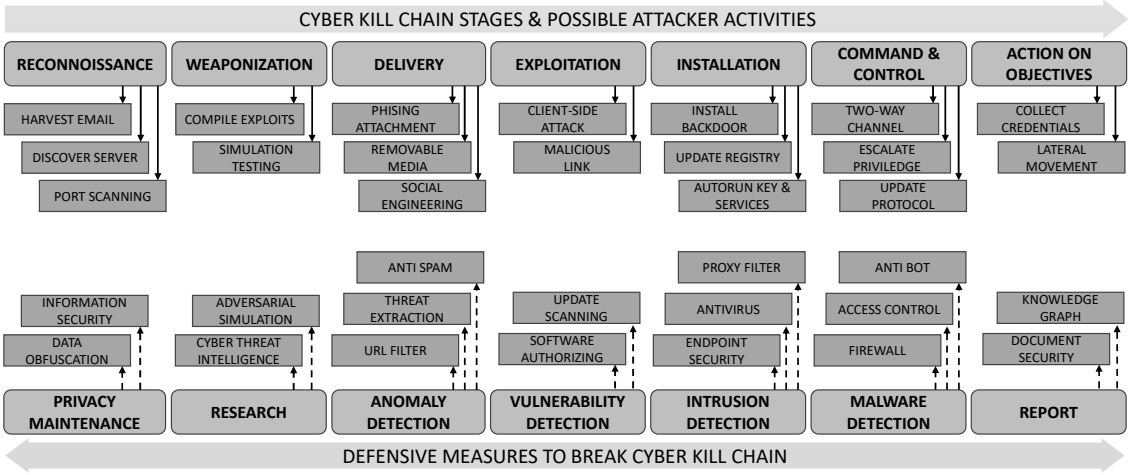


Figure 2.5: The Lockheed Martin Cyber Kill Chain, as shown in Mitra et al. [2024]

GNNs can prevent leakage of training data to maintain privacy by using bounding mechanisms. To proactively neutralize vulnerabilities, GNNs can be used to simulate attacks against IDS, or by injecting malicious data into the training data for the GNN-IDS to learn from.

There exists a large number of implementations of GNNs for anomaly detection, through either node and edge classification or through graph classification. Chaudhary et al. [2019] uses a GNN on social connection graphs to detect anomalies such as spam or malicious data within social and email networks. KAIROS is an anomaly detection system using provenance graphs that has been shown to outperform state-of-the-art Provenance-based Intrusion Detection Systems (PIDS) in detecting APTs, while incurring minimal performance overhead at run-time (Cheng et al. [2023]). KAIROS uses a temporal graph network, to analyze graphs in a streaming fashion and find both temporal and structural dependencies and changes.

GNNs can detect vulnerabilities in source code by transforming source code into a multi-edged graph where the model learns from composite programming representation of the source code, which is proposed in Zhou et al. [2019]. A different implementation is proposed by Wang et al. [2021], which is encoding code relationships into relation graphs, which allows the model to learn vulnerabilities in function relations (Mitra et al. [2024]). GNNs can be used in IDS by analyzing complex relationships in networked data, such as in Bahar et al. [2025] where GNNs are used to detect APTs with both spatial and temporal data, or in Lo et al. [2022] where a model is proposed to classify malicious network flows in IoT networks.

For malware detection, GNNs can be used to learn benign system interactions and determine if an unknown program adheres to this behaviour, as proposed in Wang et al. [2019], or a classifier using GCN can be used to learn characteristics from API call graphs as proposed in Li et al. [2021]. The final stage of the attack life cycle is reporting, to extract information from possible attacks. GNNs can be used to identify fake data in this information based on the propagation pattern, and effectivize prompt reporting and knowledge processing (Mitra et al. [2024]).

2.6 Advanced Persistent Threats

An APT is an advanced stealthy multi-step cyberattack that is covert and tries to mask itself along benign behaviour in the network. APT attacks attempt to gain access to unauthorized machines, often through phishing or other social engineering techniques. Once inside a machine, the attack would move laterally through the network and attempt to escalate privileges. APT attacks are highly-sophisticated, targeting high-profile actors like countries or companies. A total of 154 countries have been identified as targets in APT attacks, with the 10 most targeted countries accounting for 650 attacks (Yuldoshkhujaev et al. [2025]). Signature-based and statistic-based intrusion detection techniques struggle to identify APT attacks due to their stealthy nature and masking among benign network traffic, but ST-GNNs are proven to be effective in detecting APTs (Bahar et al. [2025]).

APT attacks could be used for data exfiltration, where an adversary is attempting to steal data from the network (MITRE ATT&CK Authors [2025a]), defined by MITRE ATT&CK as techniques the adversaries use to collect data, whereupon they package the data to avoid detection while removing it. Some techniques for data exfiltration are Scheduled Transfer, where adversaries schedule data exfiltration to be performed at certain times to blend in with normal network activity, or apply Data Transfer Size Limits where data is exfiltrated in fixed size chunks to avoid triggering data transfer threshold alerts.

Another goal of an APT attack could be Lateral Movement, which are techniques that adversaries use to enter and control remote systems on a network, and then pivoting through systems and accounts to gain access to their target (MITRE ATT&CK Authors [2025b]). Examples of Lateral Movement techniques are SSH Hijacking,

where adversaries hijack a legitimate user's SSH session, or Exploitation of Remote Services, where adversaries exploit remote service to gain unauthorized access to internal systems.

2.7 Threat Model

To evaluate the capabilities of graph-based IDS architectures, a threat model is employed to exemplify one or more attacks within a given scenario. To evaluate GNNs capabilities in regards to discovering complex attacks inside a 5GC environment, a simulated multi-step attack is chosen as the threat model.

For this thesis, the threat model is an attacker that is assumed to have breached the perimeter of the system, and obtained access to an application pod. The attacker is in control of a pod inside the cluster, and uses the pod to launch attacks against the other pods in the cluster. The attacker is able to exploit a vulnerability due to a misconfigured pod, with an open port that is not supposed to be open for communication. The misconfigured pod contains a list of encrypted user credentials, and the attacker also knows the plaintext password of a regular user in said list of credentials.

2.8 Related Work

One of the recent contributions to graph-based IDS research for 5G is Alnazzawi et al. [2025]. In this paper, the proposed model uses a topology-aware Multi-Graph Neural Network (MGNN) architecture with two complementary graphs:

- A communication topology graph processed by GCN layers to capture structural dependencies.
- A k-Nearest Neighbors (kNN)-based feature similarity graph processed by GAT layers to learn feature-level relationships.

The resulting embeddings are fused via a trainable attention mechanism and passed to fully connected layers for classification. The authors tested the performance of their solutions on different datasets, mostly non-5G datasets containing network flows between entities. But one notable dataset used is the *5G-NIDD* dataset (Samarakoon et al. [2022b]), created in 2022 and last updated in 2025.

This dataset provides a comprehensive capture of traffic from a functional 5G network, including both benign user activity and multiple attack scenarios such as port scans and DoS attacks. It counts 1.2 million network flows fully labeled with benign/malicious tags, as well as detailed attack types and tools, enabling both binary and multiclass classification. It also includes combined traffic as well as per–base station subsets, which are useful for research in areas such as federated learning. Reported performance metrics show that the dataset supports strong results across various ML models, demonstrating its value for evaluating new intrusion detection and multiclass classification approaches.

However, it remains relatively small, comprising only 1.2 million network flows and packets, and covering just two attack types: DoS and PortScan (Samarakoon et al. [2022a]). Furthermore, it mainly contains malicious activity (60.7%), which for modern graph- and autoencoder-based models is disadvantageous, since they need benign data-flows to a large extent when training to familiarize with complex networks' normal behaviour.

Within this specific area, lack of good quality data is a **major issue** (Komisarek et al. [2021]). This project will be an attempt at implementing modern graph-based IDS models in collaboration with *Ericsson* using substantially larger datasets, comprising of larger networks and for longer durations. The main problem to investigate is whether Graph-based models can be used for intrusion detection in 5G Core Networks, to evaluate whether this is a viable security solution. In papers such as in Bahar et al. [2025], it has been proven that GNNs are effective in discovering threats that are otherwise difficult to identify, but it is not known if they are effective in the 5GC architecture due to the difference in data flows and topology as well as scale.

In recent years, different works have investigated the capabilities for GNNs in IDSes, as well as IDSes for discovering APT attacks in 5GC environments, but there are no works combining the two.

2.8.1 GNN for IDS

One of the best performing GNN-based IDS systems is MAGIC, introduced by Jia et al. [2024]. MAGIC is a self-supervised APT detection framework based on an extended masked graph autoencoder with GAT layers to capture structural dependencies in provenance graphs. It learns representations of benign behaviour by reconstructing masked node features and aggregates embeddings through a graph representation module for multi-granularity detection. Anomalies are identified using a Kd-Tree based outlier detector, enabling efficient and scalable detection without requiring labeled attack data. MAGIC is implemented using OS-layer data, and is trained on system audit logs, as opposed to the network-layer data that is found in 5GC environments.

Another GNN-based IDS system is KAIROS, introduced by Cheng et al. [2023]. KAIROS is a streaming provenance-based intrusion detection system built on a graph neural network encoder decoder architecture. The encoder uses a TGN to model dynamic neighborhood structure and evolving node states, while a lightweight Multilayer Perceptron (MLP) decoder reconstructs newly observed edges. Anomalies are identified via edge reconstruction errors learned on benign data. Within each time window, KAIROS flags suspicious nodes based on (I) anomalousness edges whose reconstruction error exceeds a dynamic threshold (mean + 1.5 std.) and (II) rareness measured using IDS. Suspicious activity is then aggregated to reconstruct compact attack footprints. KAIROS is also implemented using OS-layer data, which limits its capabilities in a 5GC environment, but it shows the effectiveness of TGNs

in IDS systems, which is relevant as TGNs are a suitable model for the properties of network layer data.

One of the most recent works in GNN-based IDS solutions is CONTINUUM, introduced by Bahar et al. [2025]. CONTINUUM is an APT detection framework based on a Spatio-Temporal Graph Neural Network (STGNN) autoencoder deployed in a federated learning setting. The encoder captures both structural interactions (spatial dependencies) and graph evolution over time (temporal dynamics, via Recurrent Neural Network (RNN) layers), producing node embeddings that represent system behaviour. These embeddings are used for anomaly detection through a K-Nearest Neighbors (K-NN) classifier, which flags nodes as anomalous based on deviation from benign patterns. For graph-level tasks, embeddings are aggregated using average pooling. Model updates are securely shared using homomorphic encryption, enabling privacy-preserving and scalable deployment. CONTINUUM is also trained on OS-layer data, but it shows the high potential of Spatio-Temporal Graph Neural Networks (ST-GNN) in discovering APTs.

Holmkvist Bergqvist [2025] proposed a model using GNNs to detect DDoS attacks from network flows, and constructed flow graph from these network flows. This work therefore build a GNN-based IDS from network flows, but not in a 5GC environment but instead from an open dataset containing DDOS attacks. The model used in this thesis is a Graph Isomorphism network, with a F1 Score of 93.87% for detecting DDoS attacks on the full flow graphs.

2.8.2 IDS for 5G

Flon and Sulaiman [2025] introduced an IDS solution for discovering APTs in 5GC environments, by training ML models on security alerts within 5GC Kubernetes Clusters. In this paper, an open-source dataset consisting of security alerts for APT attacks inside 5GC clusters was used to train machine learning models to evaluate if they could differentiate between false alerts(false positives) and true alerts (true positives). The models used in this paper are Long Short-term Memory, Markov Chain, Hidden Markov Chain, Convolutional Neural Network and Bayesian Network. This paper shows that discovering APT attacks in 5GC environments is an ongoing research question, but it differs from this thesis since the models are not trained on the network traffic, but instead on security alerts inside the network.

Radoglou Grammatikis et al. [2023] introduced an IDS solution for 5GC using AI methods to detect potential cyberattacks in the SMF and User Plane Function (UPF). This solution uses packet flow statistics to train the models, and uses Decision Tree models such as XGBoost and Random Forest. This paper shows that machine learning algorithms can be used in combination with packet flow data to detect cyberattacks, and the best performing model was XGBoost with an accuracy of 85.5% in detecting attacks.

3
Method

To evaluate the effectiveness of GNN-based IDSs in the 5GC, a mock 5GC environment capable of simulating both benign and malicious activity was employed. The developed GNN-model's objective was to determine where an attacker has gained a foothold within the cluster based on captured network traffic. The GNN-models was trained on a combination of simulated operational and attack data from one of *Ericsson's* 5GC test deployments.

The environment can simulate up to one million active users, approximating a small-scale real-world deployment which ensured sufficient training data under realistic operating conditions. The operational data provided a realistic baseline behaviour, while the simulated scenarios enabled a controlled evaluation of adversarial activity. The operational data also served as camouflage for simulated attacks.

A TGN-based architecture was developed to model the Kubernetes-based 5GC environment as a dynamic continuous-time graph. In this representation, Kubernetes pods and services were modelled as nodes, while network flows and packet exchanges were represented as temporal edges occurring at precise timestamps. This was particularly relevant for Kubernetes environments, where communication patterns were highly dynamic due to container orchestration, autoscaling, service discovery, and ephemeral workloads.

At a high level, the model architecture consisted of three sequential parts:

1. **Temporal Graph Network (TGN) memory module.**
   Encodes the continuous stream of temporal interaction events (edges) into evolving node-level memory representations, capturing the dynamic behaviour of Kubernetes entities over time.
2. **Graph convolutional message-passing layer.**
   Refines node embeddings by aggregating information from spatially relevant neighbours using message passing, thereby incorporating local spatial context and recent interaction structure.
3. **MLP classification head.**
   Performs edge-level node classification on source and destination node embeddings to predict whether endpoints of a communication event is malicious (attacker) or benign.

The project included several key steps:

1. **Attack Execution and Data collection**
   Design and execution of attacks within the 5GC test environment that closely mirrored real-world scenarios, that enabled structured labelling and separation of attack data and into its respective phases.
2. **Data preprocessing.**
   Raw network flows were transformed into a temporal graph structure. This included feature extraction, temporal segmentation, encoding, and data augmentation.
3. **Model development.**
   Implemented and trained the GNN-models with our architecture, suitable both for detecting executed attacks used in high throughput networks.
4. **Model evaluation.**
   Evaluated best performing models and their usage of imputed event features.

Through this approach, the aim was to provide an evaluation of graph-based IDS for 5GC, demonstrating its ability to detect complex attack patterns.

3.1 Attack Execution and Data collection

The dataset used in this study was synthetically generated within an *Ericsson* test environment designed to emulate a production-like 5GC deployment without relying on production data. The environment consisted of a Kubernetes cluster hosting a deployed AMF service across four worker nodes and a total of 152 pods representing both the AMF and other simulated NFs.

Network-level data was collected from one of the worker nodes, capturing packet flows associated with both intra-service and simulated inter-service communication with respect to the AMF within the cluster. The resulting dataset included both benign operational traffic and simulated malicious activity. To generate labelled attack data, controlled vulnerabilities were introduced into the environment and exploited through structured attack scenarios. This enabled the collection of both normal and adversarial behaviour under realistic operating conditions.

The simulated attacks were performed using existing frameworks utilized by *Ericsson*'s penetration testing team, and they assisted in designing the steps and end goal of the attack. A separate attacker pod was deployed inside the Kubernetes cluster to execute attacks against other pods in the environment, thereby generating anomalous network activity. The attacker pod ran Kali Linux with elevated privileges, following the assumption that an attacker had already breached the system perimeter and established a foothold within the cluster.

This attack targets two different pods in the Kubernetes cluster. An artificial vulnerability was introduced into a designated victim pod, which the attacker subsequently exploited to gain administrative access to the pod responsible for managing its communications. The deployed attack was multi-step in nature, beginning with

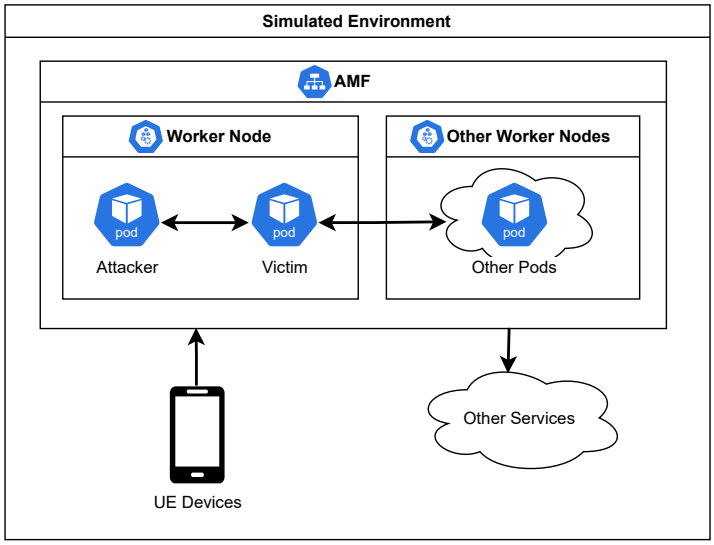


Figure 3.1: Overview of AMF test system architecture where data generation is performed

reconnaissance of the cluster to identify exposed services and later progressing to unauthorized access and credential manipulation.

1. To perform reconnaissance, the attacker enumerated pod IP addresses within the cluster and used `Nmap` to scan for exposed ports. `Nmap` is an open-source tool commonly used for network and port scanning to identify reachable hosts and services (Authors [2026]). After discovering an exposed port on the victim pod, the attacker attempted to identify available API endpoints using `ffuf`, an open-source fuzzing and directory discovery tool frequently used in penetration testing (ffuf Authors [2026]). Through this process, the attacker discovered an HTTP endpoint exposing a configuration file containing encrypted user credentials.
2. The attacker then used `curl` to communicate with the endpoint on the victim pod and verified that it was accessible without sufficient protection. The credential file was downloaded, modified, and uploaded back to the victim pod. It was assumed that the attacker already knew the plaintext password of a regular user account and reused the corresponding encrypted password hash to overwrite the administrator credentials. This enabled the attacker to authenticate as an administrator.
3. The attacker subsequently accessed the compromised system through `Secure Shell (SSH)` by connecting to a pod that acts as a load balancer for the victim pod, and created a new account with administrative privileges, thereby appearing as if an authorized administrator created the account and an opening for future access.

The attack exhibits characteristics of both lateral movement and data exfiltration.

Sensitive credential data was extracted from the victim pod, modified, and reinserted to escalate privileges and gain administrative access to the whole cluster. The overall attack workflow is illustrated in Figure 3.2.

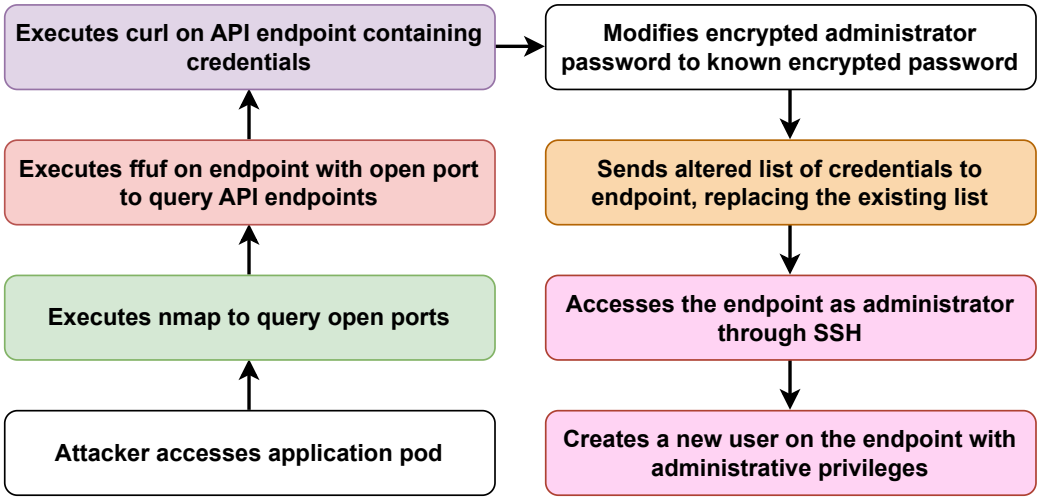


Figure 3.2: Multi-step attack workflow. Matching colours of attack phases as figure 4.4.

It is assumed that the attacker has gained access to an application pod and obtained the credentials of a user account; social phishing could be one plausible vector through which an attacker could achieve this. If an adversary were to gain cluster access (e.g. through social phishing) and communication between pods had not been restricted, the attack would be reproducible.

The data was collected by running `tcpdump` on the worker node hosting the attacker, capturing packet data for all communication with and within the worker node. Each phase of the attack was indicated by sent UDP packets from the attacker with signature payloads for the start and end of each corresponding step in the attack. To prevent the model from overfitting to the background traffic, three separate datasets were created for training, testing, and validation. The attack was executed multiple times for the training dataset in order to supply sufficient data for the malicious data class.

The training dataset was generated by collecting benign data for a duration of 15 minutes, with 8 instances of the multi-step attack, while the test and validation datasets were generated by collecting benign data for a duration of 5 minutes with one instance of the multi-step attack.

3.2 Data Preprocessing

The collected network traffic from `tcpdump` resulted in packet logs stored in Packet Capture (PCAP) format. The collected data was then subjected to feature extraction to reduce dataset size, retain relevant information, and convert the traffic records into the TemporalData representation used by PyTorch Geometric (Fey et al. [2025]). TemporalData is a data structure composed of a stream of events (edges) with structured messages, capable of describing any Continuous-Time Dynamic Graph (CTDG).

An event consists of a source node, a destination node, a timestamp, a message (feature vector), and possibly labelling. TemporalData was chosen as the dataset's data structure because it maps naturally onto the captured packet stream and is the native input format for TGN.

Table 3.1 shows extracted features from the captured packet logs with their respective usage and encoding. Feature extraction and encoding was performed with python tools such as `Pandas` (pandas development team [2020]), `Scapy` (Biondi et al. [2025]), and `Scikit-learn` (Pedregosa et al. [2011]). All extracted and encoded features was then concatenated into a feature vector for the TemporalData's message field, except for the features Timestamp ISO, Source IP and Destination IP. All captured traffic below the network layer (layer 3) was excluded from the dataset.

From the extracted features, a temporal graph was constructed consisting of a set of nodes and an event stream. Each node represents a unique IP address observed in the captured traffic, treating each address as a distinct network host within the dataset. Leveraging the TGN module, each host (node) maintains a persistent state that evolves over time, allowing hosts to be modelled as temporal communication actors whose behaviour and interactions accumulate across the observation window.

Event labelling was performed in two stages:

1. In the first stage, packet-level maliciousness of source and destination was determined by comparing the source and destination IP addresses of each packet against the known IP address of the attacker pod. Traffic originating from or destined for the attacker was marked accordingly with the source or destination as malicious corresponding with table 3.2.

2. In the second stage, attack phase boundaries were identified through a set of out-of-band UDP marker packets injected into the traffic stream at the start and end of each simulated attack phase. Each marker packet carried a distinct ASCII payload signature identifying the phase and whether it constituted a start or end boundary. During parsing, these signatures were matched against the raw UDP payload to determine the temporal extent of each phase, defining closed intervals over the packet stream within which all attacker traffic was assigned the corresponding phase label. These UDP signature packets were removed after phase labelling.

Feature	Description	Encoding
Timestamp	Temporal position of packet	Normalized (min-max)
Timestamp ISO	Human-readable timestamp	Excluded from model
Protocol	Communication type (TCP/UDP/TLS/etc.)	One-hot
Source IP	Sender node identifier	Label encoding
Destination IP	Destination node identifier	Label encoding
Source Port	Indicates source application	Log and StandardScaler
Destination Port	Indicates target application	Log and StandardScaler
Length	Packet size from network layer	StandardScaler
IP Flags	Fragmentation info; relevant for evasion detection	One-hot
IP TTL	Path length and topology hints; spoofing detection	StandardScaler
IP Fragmentation Offset	Fragmentation usage; exploitable in evasion attacks	StandardScaler
ICMP Type	Type of ICMP packet (echo, unreachable, etc.)	One-hot
TCP Flags	Connection state; repeated flags may indicate port scan	One-hot
TCP Urgent Pointer	Captures unusual urgent pointer usage	StandardScaler
TCP Data Offset	Header size and options; signals non-standard behaviour	StandardScaler
TLS Packet Type	TLS message types; handshake anomaly detection	One-hot
HTTP Method	HTTP action type (GET/POST/etc.)	One-hot

Table 3.1: Extracted features with encoding and rationale.

To improve the generalizability of the trained model, data augmentation was applied to the labelled attack traffic. A key concern in training an IDS on captured attack data was that the model may learn to associate malicious behaviour with a specific node identity rather than with the underlying traffic patterns. This was mitigated by reassigning the attacker's identity to a uniformly sampled node identity from

Source	Destination	Source Label	Destination Label
Benign Pod	Benign Pod	0	0
Attacker Pod	Benign Pod	1	0
Benign Pod	Attacker Pod	0	1
Attacker Pod	Attacker Pod	N/A	N/A

Table 3.2: Event labelling truth table, with source and destination referring to source and destination nodes (IP addresses).

the set of known cluster nodes, effectively simulating a scenario in which a specific, alternate node within the cluster is compromised. The attacker's own identity was excluded from the pool of assignable identities, ensuring that the original attacker identity could not be reintroduced through augmentation. Additionally, any packets for which both the source and destination node identifiers resolved to the attacker node were discarded, as such entries would not correspond to a meaningful network interaction.

This augmentation served a dual purpose: it prevented the model from memorizing the node identity of the attacker as a discriminative feature, and it introduced variation in the traffic patterns associated with malicious activity, which exposed the model to a broader range of adversarial scenarios during training. The intended behaviour of the IDS was thus to detect anomalous communication patterns between nodes rather than to flag a specific node identity. Figure 3.3 depicts the data augmentation process on a high level.

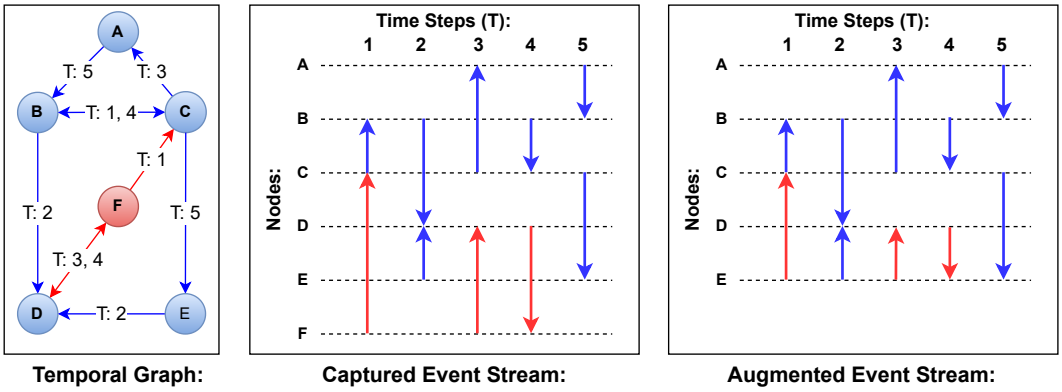


Figure 3.3: *Temporal Graph* (left panel) depicts the full interaction graph as it would be reconstructed from all observed events. Nodes A through F represent network endpoints (IPs), and directed edges encode communications between them.

In Figure 3.3, the colours blue and red for nodes and edges depicts class labelling of benign and malicious entities. Node F functions as the designated attacker, interacting with nodes C and D across time steps (T) 1, 3 and 4. *Captured Event Stream* (middle panel) depicts the same data represented as a chronological sequence of interactions, with nodes on the vertical axis and discrete time steps on the horizontal

axis. (Events are in discrete time for simplicity.) Each arrow denotes an interaction event at that time step. *Augmented Event Stream* (right panel) presents the result of data augmentation, where the attacker's identity (node F) is replaced by the benign node E, which inherits the full temporal interaction pattern originally attributed to node F. This would simulate a scenario in which node E have been compromised.

3.3 Learning Task

The objective of the proposed IDS was to identify compromised nodes within a Kubernetes-based 5GC environment from observed network communication patterns. The model learned to distinguish benign communication behaviour, such as normal service interactions and system operations, from malicious behaviour associated with attacker activity. Examples of malicious behaviour includes reconnaissance, unauthorized access attempts, privilege escalation, and lateral movement within the cluster.

The learning problem was formulated as a supervised binary node classification task. For each observed communication event, the model predicted whether the participating nodes were benign or compromised.

Given a sequence $\mathbf{E}$ of directed interaction events,

$$\begin{gathered}\mathbf{E} = \{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n\}, \quad \mathbf{e} = (u, v, t, m), \\ u, v \in V, t \in T, m \in M,\end{gathered} \tag{3.1}$$

where u and v denote the communicating source and destination nodes, t denotes the timestamp, and m represents the associated message features.
The objective was to learn a mapping

$$f : V \times V \times T \times M \to \{0, 1\}^2, \tag{3.2}$$

where the output corresponds to the predicted classes of the communicating source and destination nodes. Let

$$\hat{\mathbf{y}} = (\hat{y}_u, \hat{y}_v) = f(u, v, t, m), \tag{3.3}$$

where $\hat{y}_u, \hat{y}_v \in \{0, 1\}$ denote the predicted labels of the sender and receiver, respectively. The labels are defines as

$$y = \begin{cases} 0, & \text{benign}, \\ 1, & \text{compromised}. \end{cases} \tag{3.4}$$

Thereby, for each observed communication event, the model predicts the binary classes of both participating nodes.
In practice, the learned mapping was implemented as a composition of a temporal graph encoder and a node classification head. Given an interaction event ($\mathbf{e}$), the GNN encoder first computed latent representations of the participating nodes,

$$h_u = \phi(u, v, t, m), \qquad h_v = \phi(v, u, t, m), \\ h_u, h_v \in \mathbb{R}^d \tag{3.5}$$

where $\phi(\cdot)$ denotes the graph representation learning process, d denotes the embedding dimension, and h_u, h_v are the resulting node embeddings. These embeddings were subsequently passed through a classification head g, producing the predicted node labels:

$$\hat{\mathbf{y}} = g(h_u, h_v), \tag{3.6}$$

where

$$g : \mathbb{R}^d \times \mathbb{R}^d \to \{0, 1\}^2. \tag{3.7}$$

Node labels were derived from the known attacker identity during dataset generation and stored with each interaction event. Interactions involving the attacker node were labelled as compromised according to the labelling scheme described in Section 3.2.

Motivation for Binary Classification

Intrusion detection is commonly formulated either as anomaly detection or supervised classification.

An anomaly detection approach learns a representation of benign behaviour and identifies deviations from this baseline as potential attacks. Such approaches are particularly useful when labelled attack data is unavailable.

In this work, binary classification was selected instead. The synthetic data generation process enabled accurate labelling of attacker activity, making supervised learning feasible. Furthermore, Kubernetes-based 5GC environments exhibit highly dynamic communication patterns due to service discovery, container orchestration, autoscaling, and fluctuating user activity. These factors make it difficult to establish a stable definition of normal behaviour and may lead to elevated false-positive rates in anomaly-based systems.

By formulating the problem as binary classification, the model can directly learn discriminative patterns associated with attacker behaviour and establish a decision boundary between benign and malicious activity.

The task was formulated as node classification rather than edge classification because the primary objective was to identify compromised hosts within the cluster. Individual communication events may be benign even when originating from a compromised node, whereas identifying the compromised endpoint enables security operators to localize the attacker's foothold and take remediation actions.

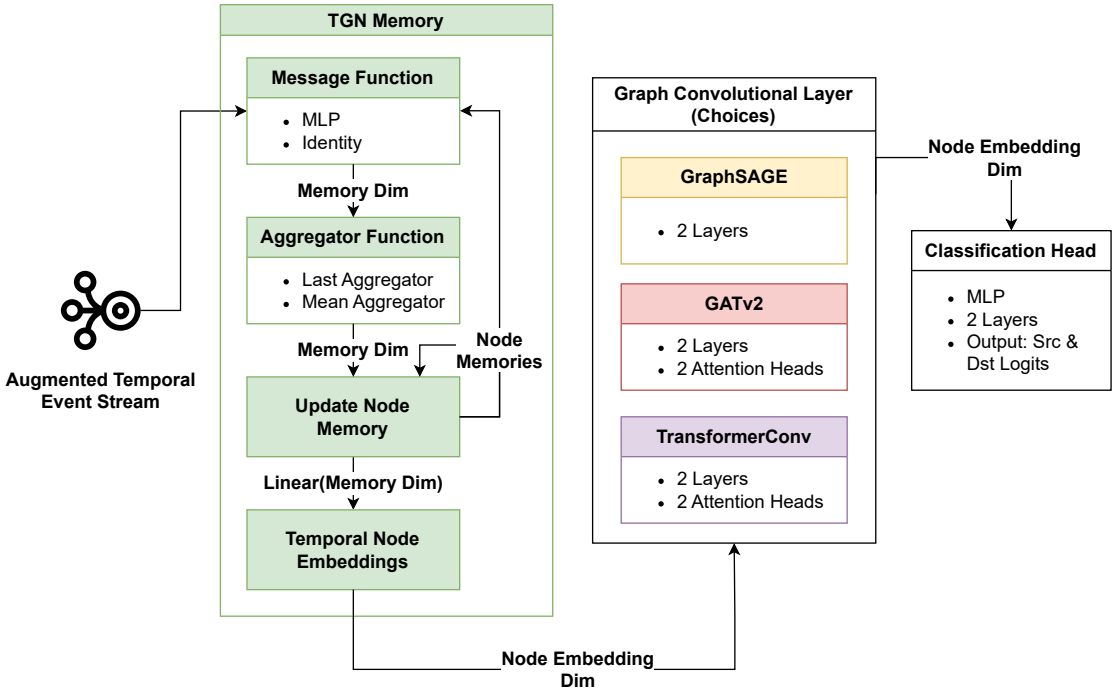


Figure 3.4: A high-level overview of the model architecture and tested variations. Varying sizes of embedding dimensions **Memory Dim** and **Node Embedding Dim** were explored, where the **Linear** function performed rescaling between them.

3.4 Model Architecture

The proposed model consisted of three sequential components designed to jointly capture temporal dynamics, structural neighbourhood context, and perform node-level classification in a streaming interaction graph. The architecture was built on a TGN memory backbone and extended with a message-passing GNN encoder and a classification head. Figure 3.4 gives a simple overview of the architecture.

Temporal Graph Memory Module (TGN)

A Temporal Graph Network (TGN) memory module was used to model the continuous-time interaction stream. PyTorch Geometric's implementation of `TGNMemory` was used along with its included classes. Each node maintained a persistent memory vector $m_v(t)$, which was updated whenever the node participated in an interaction event. The memory updates was based on the default implementation of `TGNMemory` with these variations during testing:

- A message function as a MLP and Identity function (concatenation of inputs).
- An aggregation function as *LastAggregator* and *MeanAggregator*.

Degrees of freedom included the choice of message function (identity-based or MLP-based), aggregation strategy and memory dimensionality.

Graph Convolution Layer

A graph convolution layer was applied on top of the learned temporal node memory representations to incorporate structural neighbourhood information. Multiple types of graph convolutional operators were tested:

- **GraphSAGE**: PyTorch Geometric's SAGEConv implementation was used.
- **GAT**: PyTorch Geometric's GATv2 implementation was used.
- **TransformerConv**: PyTorch Geometric's TransformerConv implementation was used.

This layer produced refined node embeddings h_v, which incorporated both temporal and structural context. **Degrees of freedom** included the choice of convolution operator and node embedding size.

Edge-level Classification Head

A MLP was used as the final classification head to perform node-level classification for both endpoints of each interaction. For each interaction (u, v, t, m), the model produced source and destination label predictions based on produced h_u, h_v. The classification head mapped node embeddings to logits over two classes (benign or malicious), independently for source and destination nodes. **Degrees of freedom** included the number of hidden layers, hidden dimensionality, dropout rate, and activation functions.

Model Tuning

Hyperparameter tuning was performed with the validation dataset using Optuna (Akiba et al. [2019]), an automated optimization framework that uses Bayesian search to efficiently explore the hyperparameter space. The hyperparameters subjected to tuning, along with their respective search ranges, are listed below.

1. **Learning rate** controls the step size during gradient-based optimisation. Searched over the range $[10^{-5}, 10^{-3}]$ on a logarithmic scale.
2. **Batch size** determines the number of edges processed per training step. The values tested were 12,000, 24,000, and 48,000.
3. **Neighbour size** determines the number of historical neighbours sampled per node during temporal graph aggregation. The values tested were 10, 20, 30, and 40.
4. **Class 1 weight** controls the penalty assigned to misclassified malicious events during training. Used to address the class imbalance between benign and malicious traffic. Searched over the continuous range $[4.0, 10.0]$. The weight of class 0 was kept at 0.2 after preparatory empirical testing.
5. **Memory dimension** defines the size of the size of nodes' hidden state memory in the TGN memory module. The values tested were 64, 128, 256, and 512.
6. **Message embedding dimension** defines the dimensionality of the projected edge feature representation prior to memory aggregation. The values tested were 64, 128, and 256.

7. **Node embedding dimension** defines the size of the outputted node embeddings produced by the TGN memory and Graph Convolutional Layer. The values tested were 64, 128, 256, and 512.
8. **Message aggregation function**. Tested with *LastAggregator* or *MeanAggregator*.
9. **Message function**. Tested with *Identity Function* and *MLP*.
10. **GNN module** determines the Graph Convolutional Layer used for enriching node embeddings with structural neighbourhood information. Options being *GraphSAGE*, *GAT* and *TransformerConv*.

An Optuna (Akiba et al. [2019]) study was performed to find the optimal hyperparameters and thus the best performing model architecture and training setup. In total 50 different trials (model configurations) was evaluated for 12 epochs each of training. Training was performed with the Adam optimizer with weight-decay (Kingma and Ba [2017]) and Cross-Entropy as loss function.

3.5 Evaluation Methodology

The performance of the model was evaluated on a held-out test dataset, containing both benign and malicious network traffic. The evaluation focused on the model's ability to correctly classify nodes as either benign or compromised based on observed temporal communication events. Since the task is formulated as binary node classification, all evaluation metrics were computed from the confusion matrix entries: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

3.5.1 Evaluation Protocol

The model was trained on the training split, while hyperparameter optimisation and model selection were performed using the validation split. Final performance figures were computed exclusively on the test split to provide an unbiased estimate of generalized performance.

Repeated Evaluation under Data Augmentation

The use of a data augmentation, in which the attacker node was remapped to a randomly selected node at the start of each epoch, caused a high degree of variance for measured performance metrics. This was most likely due to the varying benign traffic the assigned attacker node outputted, since the classification task's difficulty depended on the benign background traffic. In order to obtain accurate and representative performance metrics, the evaluation was performed with the validation and test datasets through multiple iterations with data augmentation.

During hyperparameter tuning, the validation dataset was iterated 5 times and averaged after each training epoch. For the final evaluation of the best performing models, both the validation and test datasets were iterated 100 times each, with the

test dataset being the definitive benchmark. Each evaluation round started with a freshly remapped attacker, a fully reset model memory and neighbourhood loader to prevent any information leakage between rounds.

Performance from the final evaluation with the test dataset, was therefore reported as a distribution over rounds. with the primary statistics reported being the *median* and *interquartile range* (IQR), as well as mean and standard deviation.

3.5.2 Evaluation Setup

Model training and evaluation were performed on an `g6.4xlarge` Elastic Compute Cloud (EC2) instance hosted on Amazon Web Services (AWS). The hardware specifications are summarized in Table 3.3, while the software frameworks used in the implementation are listed in Table 3.4.

G6.4xlarge Specification:	
Physical Processor	AMD EPYC 7R13
vCPUs (Threads)	16
Memory (GiB)	64
GPUs	1xNVIDIA L4
GPU Memory (GiB)	24

Table 3.3: Specification of `g6.4xlarge` EC2 instance.

Software Frameworks and Versions:	
CUDA	13.0
Python	3.10.12
Torch	2.4.0
Torch-Geometric	2.7.0
Scikit-Learn	1.7.2
Scapy	2.7.0

Table 3.4: Used software frameworks and their versions.

3.5.3 Performance Metrics

The following metrics were used to evaluate classification performance. Given the class imbalance, particular emphasis was placed on metrics that remain informative and would not bloat.

Precision Precision measures the proportion of flagged nodes that are truly malicious:

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{3.8}$$

Recall Recall measures the proportion of actual malicious nodes that are correctly identified:

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{3.9}$$

F1-score The F1-score is the harmonic mean of precision and recall, balancing the cost of missed detections against false alarms:

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{3.10}$$

Matthews Correlation Coefficient (MCC) The MCC is the primary metric used for model selection and comparison. Unlike F1, it accounts for all four confusion matrix cells and yields a high score only when the model performs well on both classes simultaneously, making it particularly appropriate under class imbalance:

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}. \tag{3.11}$$

ROC-AUC The ROC-AUC measures the model's ability to rank malicious nodes above benign ones across all possible decision thresholds. It is defined as the area under the Receiver Operating Characteristic curve:

$$\text{ROC-AUC} = \int_0^1 \text{TPR}\Big(\text{FPR}^{-1}(x)\Big)\, dx, \tag{3.12}$$

where $\text{TPR} = \frac{TP}{TP+FN}$ and $\text{FPR} = \frac{FP}{FP+TN}$.

Average Precision (AP) Average precision summarises the precision-recall curve as a weighted mean of precision values at each recall threshold, and is particularly informative when the positive class is rare:

$$\text{AP} = \sum_k (R_k - R_{k-1}) \cdot P_k, \tag{3.13}$$

where P_k and R_k are the precision and recall at the k-th threshold.

Balanced Accuracy Balanced accuracy is the arithmetic mean of per-class recall, correcting for class imbalance without requiring threshold tuning:

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right). \tag{3.14}$$

4
Results

In this chapter, the results of the thesis work is presented. This includes the results of the generation of datasets, preprocessing of data, as well as training and evaluation of the GNN model on the datasets.

4.1 Data Generation

Three datasets were generated, for training, testing and validation, containing benign and malicious data. The training data was generated on the running *Ericsson* stability test system for roughly 15 minutes. The set of attacks was executed 8 times to generate malicious data for the training dataset. For the testing and validation datasets, the stability tests were running for roughly 5 minutes, and the set of attacks executed one time for each dataset. Statistics of the datasets can be seen in Table 4.1, and benign traffic plotted from the validation dataset in Figure 4.1.

4.2 Data Preprocessing

Figure 4.2 illustrates the distribution of protocols in the dataset and highlights the protocols that were treated as separate classes during the parsing process.Protocols retained as distinct categories during feature encoding are highlighted in blue, reflecting those with sufficient prevalence to warrant independent representation in the model. The remaining protocols, shown in gray, occur at lower frequencies and were collapsed into a single residual class during preprocessing, following the encoding strategy described in Section 3.2. All protocols below the network layer (layer 3) were excluded from the dataset.

	Train	**Test**	**Validation**
Total Packet Count	17,176,430	4,100,342	4,723,181
Malicious Packet Count	1,905,542	238,652	239,027
Malicious Packet Share	11%	5.8%	5%
Pcap File Size	11.68 GB	2.81 GB	4.02 GB

Table 4.1: Overview of generated datasets. The training dataset has a ratio of roughly 1:9 for anomalous and benign data, while the validation and test datasets have a ratio of roughly 1:20.

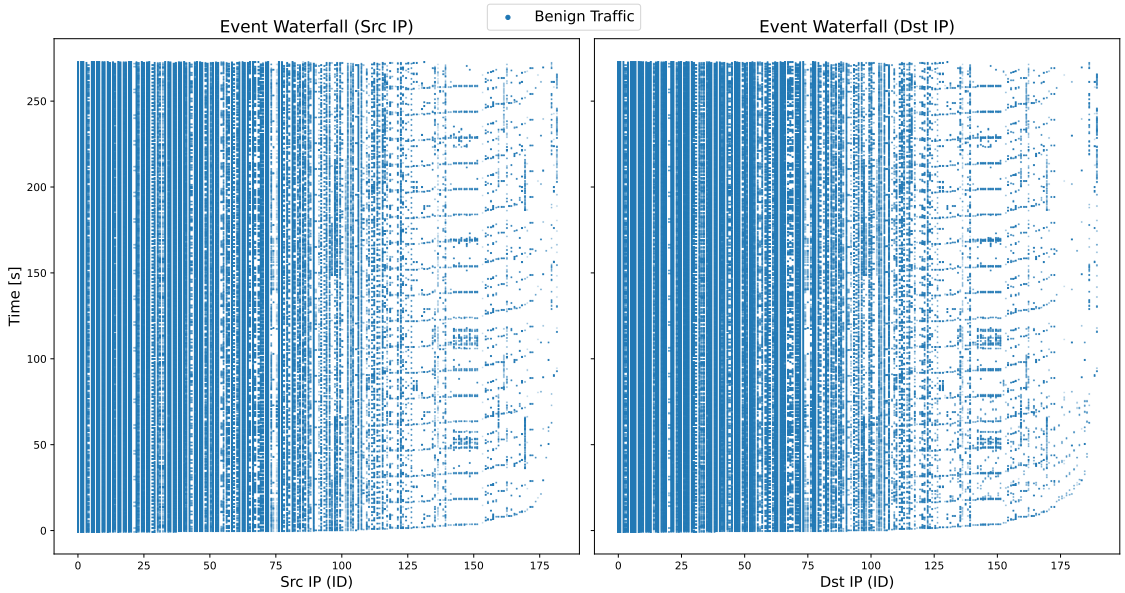


Figure 4.1: Visualization of the benign background traffic from the validation dataset. The vertical axis displays the relative time packets was captured by tcpdump, the horizontal axis represents the node ID (IP addresses). Packets are plotted both in **Src IP (ID)** and **Dst IP (ID)** representing source and destination nodes.

Figure 4.3 illustrates the port distribution of both source and destination of the captured network traffic. The port space is partitioned into three regions: well-known ports (0–1023), commonly associated with standard services such as SSH (22), DNS (53), HTTP (80), and HTTPS (443); registered ports (1024–49151), assigned to specific applications and services; and ephemeral ports (49152–65535), dynamically allocated for transient client-side connections. The y-axis is shown on a logarithmic scale to accommodate the several orders of magnitude difference in occurrence frequency between well-known service ports and the broader port space.

In Figure 4.4, the test dataset is visualized, with each stage of the attack marked, and each occurrence shown for the source and destination IPs. As the tcpdump collects data from each interface and combines them, there is duplication of the packets, which results in the diagram having doubles and triplets of certain flows, and the packets are not deduplicated. Packets marked in green correspond to the first step of the attack, the `nmap` scan. Packets marked in red correspond to the API endpoint search with `ffuf`, packets marked with purple and brown correspond to the curl get and put of the credentials, and pink corresponds to the `SSH` connection.

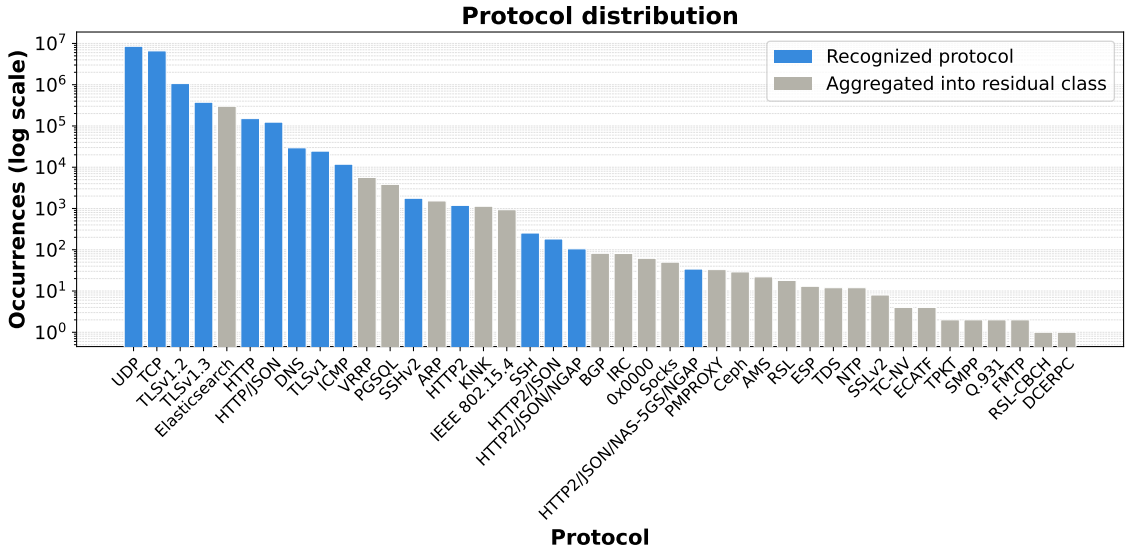


Figure 4.2: Distribution of network protocols observed across the training dataset, ranked by occurrence frequency on a logarithmic scale.

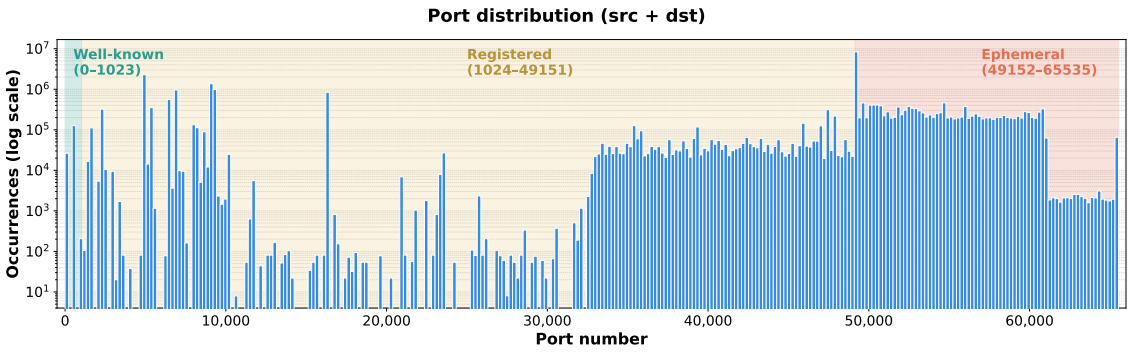


Figure 4.3: Distribution of source and destination port numbers observed in the training set, aggregated into bins of approximately 220 ports.

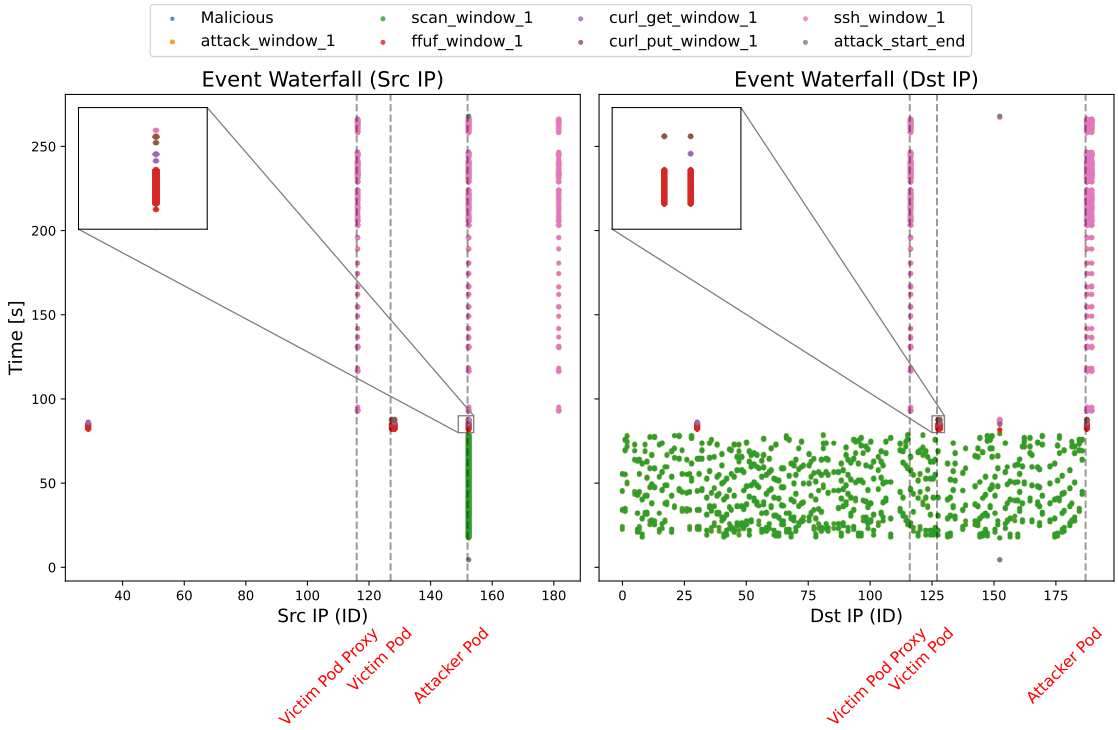


Figure 4.4: Visualization of test dataset. The vertical axis displays the relative time packets was captured by `tcpdump`, the horizontal axis represents the node ID (IP addresses)

4.3 Model Performance

After hyperparameter tuning, the top five best performing models were selected for final evaluation with the held-out test dataset. The five best performing architectures can be seen in Table 4.2 and Table 4.3.

Trial	Aggregator	GNN	Message Function
40	Mean	TransformerConv	id
45	Mean	GraphSAGE	id
36	Mean	GraphSAGE	id
33	Mean	GraphSAGE	id
17	Mean	GraphSAGE	id

Table 4.2: The five highest performing model architectures

Trial	memory_dim	msg_emb_dim	node_emb_dim	neighbour_size	C1 Weight
40	256	128	128	30	4.236
45	64	64	512	10	6.350
36	256	64	128	30	5.944
33	128	64	512	30	6.620
17	128	64	128	30	7.121

Table 4.3: Hyperparameters of the five highest performing model architectures

The Optuna study shows that the optimal aggregator is the Mean aggregator, with the five best performing architectures using the Mean aggregator, and the optimal message function being id as all five architectures use that message function. For GNN, four out of the five architectures use GraphSage, while one uses TransformerConv.

These five model architectures were evaluated based on their F1 Score, Precision, Recall, ROC-AUC, MCC and Accuracy, with the results shown in Figure 4.5, Figure 4.6 and Figure 4.7.

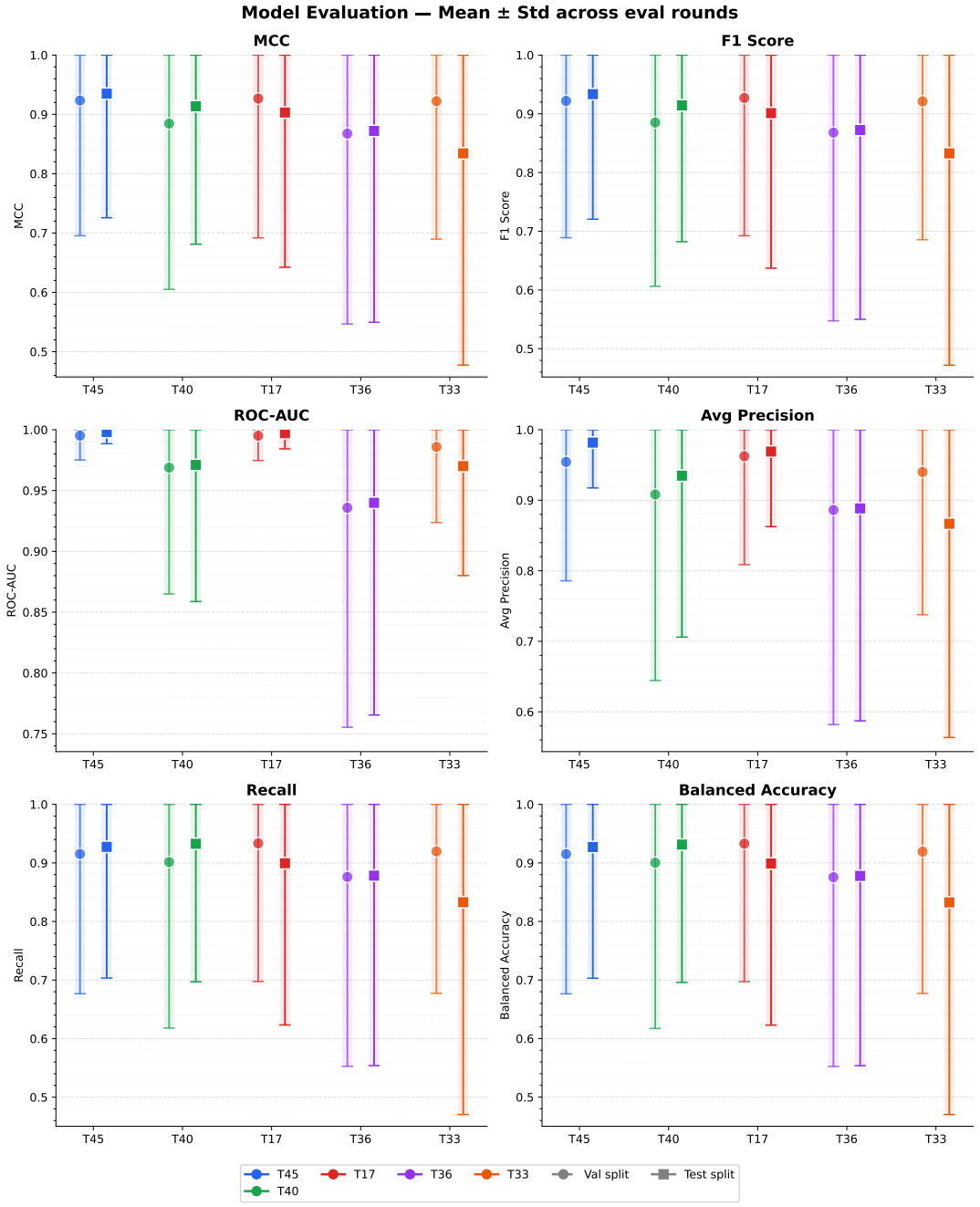


Figure 4.5: Distributions for performance metrics of top five performing models, with both validation and test dataset. Mean values and standard deviations (cropped between 0 and 1) are displayed.

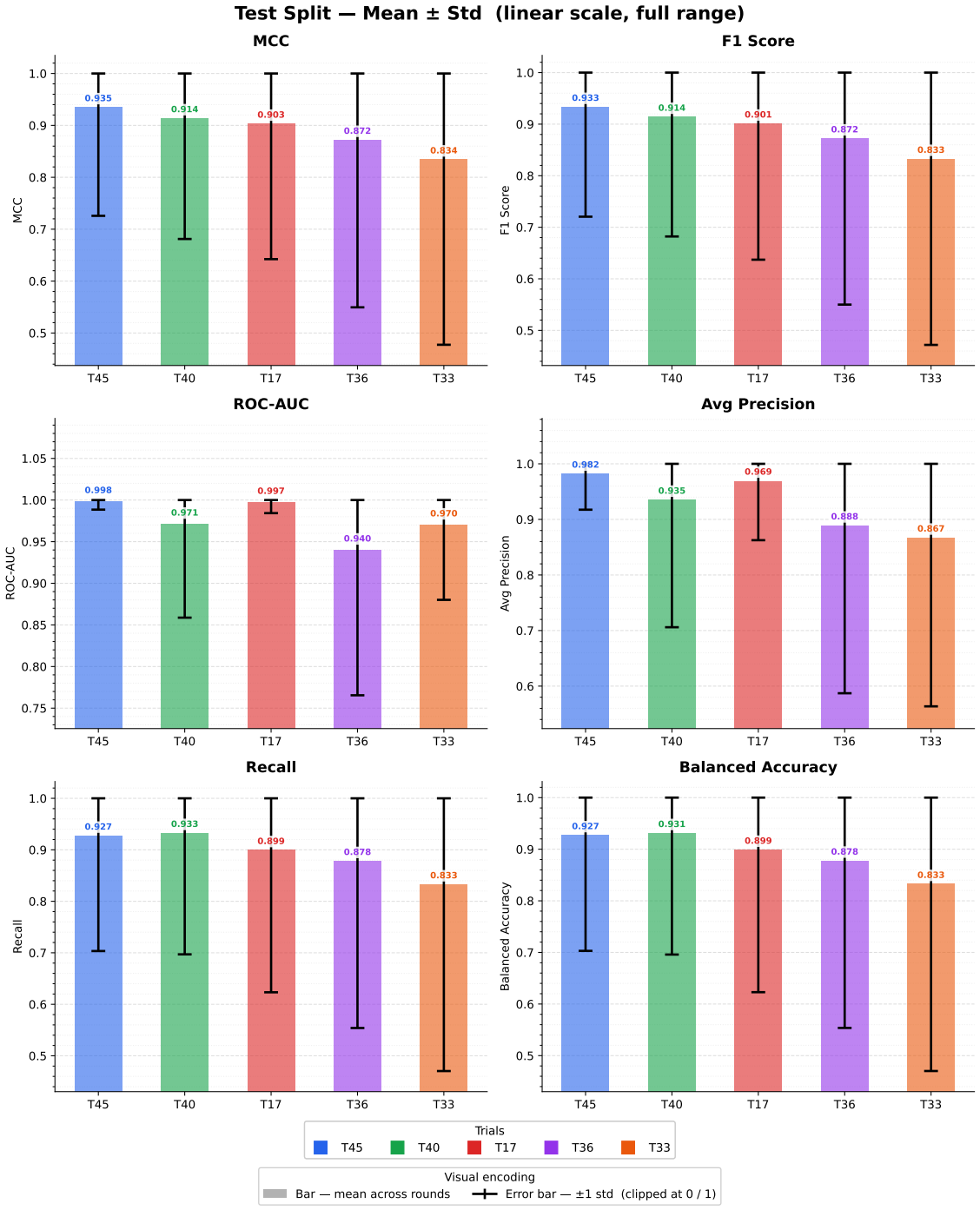


Figure 4.6: Distributions for performance metrics of top five performing models with test dataset. Mean values and standard deviations (cropped between 0 and 1) are displayed.

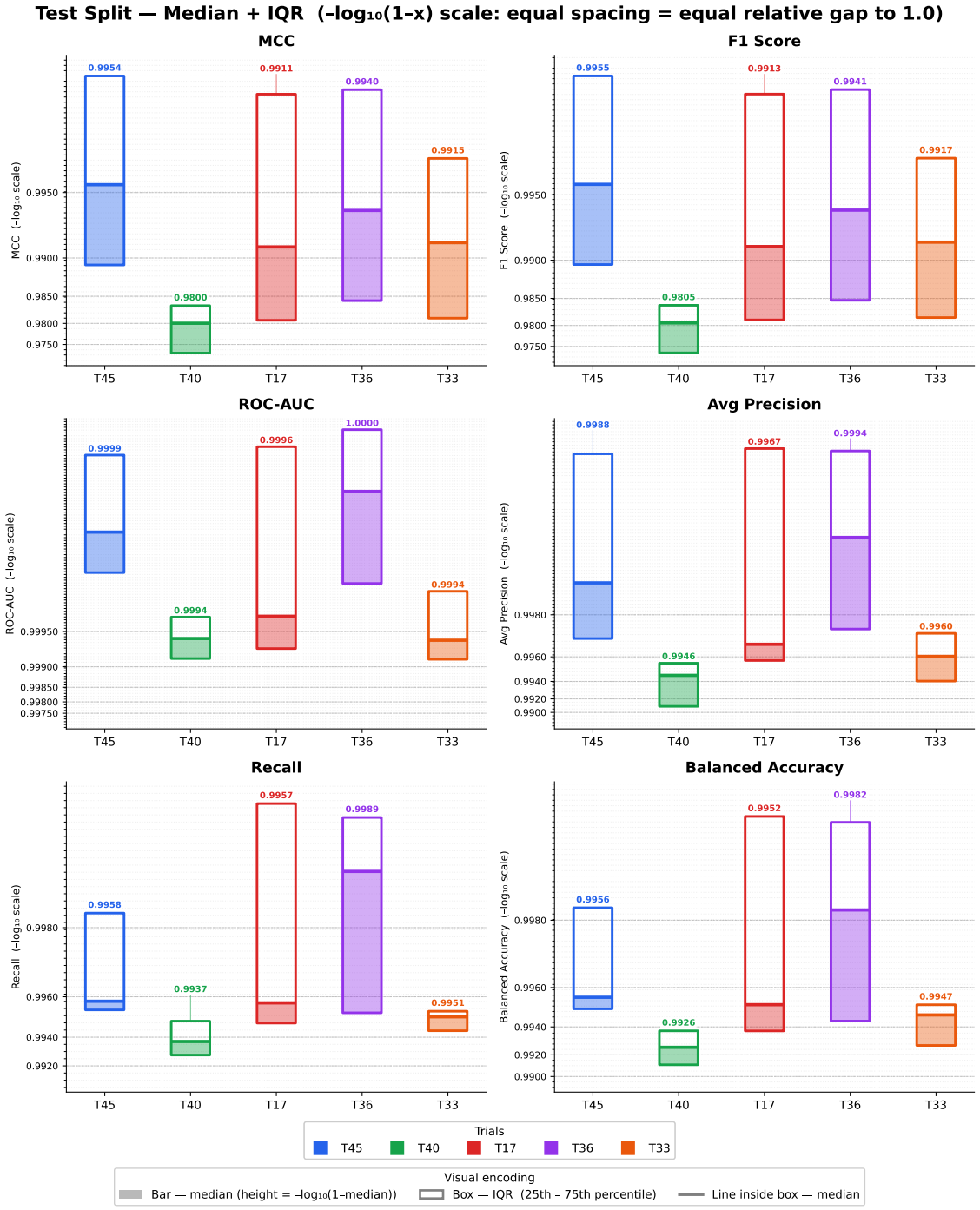


Figure 4.7: Distributions of median and interquartile range (IDR) for performance metrics of top five performing models with test dataset. Vertical axis uses complementary log scale. Values printed over each bar represents the median values.

The model architecture with the highest F1 Score is the model used in trial 45, which has a median F1 Score of 99.55%, median MCC score of 99.54%, median ROC-AUC score of 99.99%, median Precision of 99.88%, median Recall of 99.58% and median Balanced Accuracy of 99.56%. As the Recall score is lower than the Precision score for this architecture, this architecture is more susceptible to classifying false nega-

tives than false positives. The model used in trial 36 has a median Recall score of 99.89%, which means that this is the best model for preventing false negatives.

As seen in Figure 4.5 and Figure 4.6, the mean score for each model is significantly lower than the median scores for each model. This is due to the significant outliers that exist in the data, which occurs in evaluation rounds where the model fails to classify the data, and the scores are extremely low. The reason for the model occasionally failing is the data augmentation where source IP's are randomized for the malicious data, which sometimes causes the model to classify incorrectly. The boxes in Figure 4.7 represent the $25th$ to $75th$ percentile scores for each model on each metric, which shows that the majority of the scores are greater than 99% for each model except the one used in Trial 40.

5
Discussion

This chapter contains Discussion about the results of this thesis with regards to the GNN-model and the dataset, as well as potential Future Work and a potential industrial implementation.

5.1 Model Evaluation

Our model reached the most stable results from trial 45 (T45) with a median F1 score of 99.55%, for detecting the adversarial behaviour. This is a reasonably high accuracy considering the data augmentation process which is performed to make it more difficult for the model to distinguish malicious from benign packets due to the randomization of attacker identity. The class imbalance also affects the accuracy, as a more balanced dataset would let the model better learn from the malicious data. But this class imbalance was expected and is considered a common problem within the area of IDS modelling.

Without data augmentation, the model achieved higher performance when classifying malicious traffic. However, this improvement did not necessarily reflect a better understanding of network behaviour. Since all attacks in the dataset originated from a single designated attacker pod, the model could instead learn to associate malicious activity with specific source or destination IP addresses rather than the underlying behavioural patterns of an attack.

Increasing the weights of the malicious class could also reduce the amounts of false negatives, but it could also have the adverse effect of increasing the amount of false positives, which is a trade-off that would need to be considered. In a high-security environment, numerous false positives are also problematic as resources need to be allocated to security alerts, but it is less hazardous than potential threats or active intruders going unseen.

While we evaluated several model configurations, further testing with additional model configurations and more rigorous training and evaluation would further validate the produced results, especially given the high degree of variance of the performance metrics. Configurations that should be explored in future works would be different classification heads, usage of residual connection, and exchanging the GRU module in TGNMemory of a more capable sequential module.

Variants of the proposed model architecture with an added residual connection and MLP embedder, from the input event feature vector to the classification head, were shown to be both promising and hard to execute. While this improved predictive performance, further analysis indicated that the model became overly reliant on the residual connection pathway. This caused the remaining components of the architecture, including the temporal memory and graph convolution modules, to contribute less to the final predictions. As a result, much of the model's performance could be attributed to the direct edge-feature classifier rather than the learned temporal and structural representations. Solutions with residual connections should be further explored in future works. Due to lack of time, this research direction was discontinued for this project.

As our model was specifically tuned for our dataset, as well as being a combination of elements of different models such as TGN and different GNNs, comparing to other baseline models was not performed. Comparing to baseline models would validate our results, and future work creating models for IDS in 5GC would be useful to compare results of different approaches to training models on network data for threat detection.

5.2 Attack execution

The implemented attack was not a true APT, as it did contain noisy elements like the `nmap` scan as well as the `ffuf` to explore API endpoints. This does generate a large amount of malicious data for the model to learn from, which was necessary due to the class imbalance. A dataset combining true simulated APT attacks with benign 5GC network data would be ideal for future evaluation of the model, but this was out of scope for this thesis as the goal was to evaluate if a GNN model can distinguish between benign and malicious behaviour in 5GC Networks.

5.3 Industrial Implementation

If a version of our GNN-based IDS would be implemented in a production 5GC environment, there are several factors that would need to be considered. While our model uses exported static network data, a live model implementation would need to directly receive data from the network, either as periodic recordings of network data such as *tcpdumps* taken at regular intervals, or by directly listening to the network traffic. The choice also depends on where the model is deployed, if the model is deployed on a pod inside the cluster then it could directly passively listen to the network traffic, but if the pod is externally deployed outside the cluster then it would need either a listener pod directly streaming the internal network traffic to the model or periodically send chunked data.

While our proposed model uses supervised learning from a training dataset, one interesting variation for industrial use would be a model that continuously learns from network data. Labelling could be performed by the model flagging potentially

malicious packets and generating alerts for a security administrator, who could then investigate and confirm if the alert was a true positive or a false positive, and the model would then be retrained with this data added. This should allow to model to better differentiate between truly malicious or benign data with time, and therefore reduce the amount of false positives.

As our model was trained using a manufactured vulnerability as attack data, a model used in a production environment would preferably be trained on true vulnerabilities that could have been discovered by *Ericsson's* penetration testing team. The reason that our model is trained on a manufactured vulnerability is for integrity and confidentiality, as potential vulnerabilities are confidential data to protect from potential malicious actors discovering said vulnerabilities. This restriction does not exist for fully internal tools, as a production model would be, and therefore the training data could be tuned to train on true existing system vulnerabilities.

The constructed attack data was generalized in terms of attacker identity. Future implementations should consider generalizing all parts of the executed attack, including used attack vector, used phases, and randomized delays to the sequential events. This would enable increased training and potential learning with reduced risk of overfitting. If additional synthetic pods (such as the attacker pod running Kali Linux) were deployed, attacks with increased lateral movement could be emulated and randomly mapped to the ordinary production pods. You could thereby emulate complex scenarios of multi-step attacks without changing the internals of the ordinary production pods.

5.4 Limitations

This thesis had a set of limitations, including the choice of data type, the dataset as well as the chosen NF for data collection.

The data collection was only performed on the AMF, since it is the NF responsible for handling user traffic and establishing user connections, which makes it susceptible to attacks from a malicious user.
The data used was taken from the Kubernetes CNI which is the Network layer, meaning that the information the model is trained on is where packets are being sent, and what the packets consist of, but no OS layer data which would contain more details. This is a limiting factor since it is more difficult for the model to distinguish between benign and malicious behaviour without deeper information such as OS layer data.
Since no open-source datasets exist for multi-step attacks in 5G Core Networks, a new dataset was created. Creating a dataset is complex, and since the focus of the thesis is evaluating if a GNN model can detect threats in a 5GC environment, the dataset contains a simpler multi-step attack instead of very complex APT attacks.

For the attack execution, all attacks are executed from a separate pod deployed inside the Kubernetes Cluster, instead of an existing network pod being breached

by a potential attacker. The reason for this limitation is that the Kali Linux tooling used in the attack does not exist on the other network pods, as they have limited capabilities depending on their function, while our attack execution requires tools such as `nmap` and `ffuf` which exist in Kali Linux.

5.5 Future Work

This thesis models the effectiveness of a GNN when detecting multi-step attacks on 5GC networks, but the multi-step attack is not a traditional APT attack, so some future work could be creating a dataset in a 5GC environment where an APT attack is executed. Since the dataset used in this thesis was not being published due to the data being internal *Ericsson* data, and the lack of data in this field, creating an open-source dataset combining 5GC and APT is of high interest. This dataset could include examples of more attacks for model training and evaluation, since our dataset only includes one multi-step attack which already assumes that an attacker has gained access to the Kubernetes cluster and is in control of a pod. In real-world scenarios, many different types of threats are prevalent, and a reliable IDS needs to be able to detect these threats and be prepared to handle them.

The dataset created only contains a simple variant of a multi-step attack, and in a real-world scenario, a high-profile actor looking to target critical telecom infrastructure might perform highly complex APT attacks, which an IDS system would need to be able to detect. Therefore an open dataset combining complex APT attacks in a 5GC production environment would be a significant contribution for future research.

Our dataset contains markers generated for each phase of the attack, in the form of packets sent out by the attacking pod. These markers are used to visualize which phase the multi-step attack is currently in, but it is not something that the model currently takes into regard. Future work could be to use these markers as input to the model as well, and train the model to predict not only if a packet is malicious or benign, but also which phase of the attack that a malicious packet belongs to.

The threat model in this thesis also makes certain assumptions that are not prevalent in a real-world 5GC environment, since the attacker is not only assumed to have gained access to an application pod in the cluster, but it is also assumed that communication with the victim pod is not limited and the attacker also knows credentials for a user on said pod. These assumptions are necessary to collect data for research purposes, but they are less relevant for real-world use. This thesis investigates whether GNNs are able to detect a multi-step attack in a simulated 5GC environment, but for production deployment other assumptions would need to be considered.

6
Conclusion

In this thesis, we presented a graph-based model that is successfully able to identify multi-step attacks inside a 5GC environment, with a relatively stable median F1 score of 99.6% which indicates the strong ability of the model to differentiate between benign and malicious network behaviour, using only packet logs. We created a dataset, consisting of simulated benign network data with an attacker breaching a component of the 5GC network through a manufactured vulnerability. In the beginning of this thesis, we presented two research questions, with the first question successfully answered, while we leave the latter for future work.

Bibliography

Get to the core of 5g: 5g core (5gc) - explained, 2025. URL https://www.ericsson.com/en/core-network/5g-core. Accessed: Friday 10th July, 2026.

3GPP. 5g system overview, 2022. URL https://www.3gpp.org/technologies/5g-system-overview. Accessed: Friday 10th July, 2026.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Noha Alnazzawi, Fatima Asiri, Nazik Alturki, Zhumabayeva Laula, Saniya Zafar, Shahid Latif, and Jawad Ahmad. Mgnn-ids: a multi-graph neural network approach for robust intrusion detection in the internet of things. *Telecommunication Systems*, 88(4):1–20, 2025.

Nmap Authors. Nmap reference guide, 2026. https://nmap.org/book/man.html Accessed: Friday 10th July, 2026.

Atmane Ayoub Mansour Bahar, Kamel Soaid Ferrahi, Mohamed-Lamine Messai, Hamida Seba, and Karima Amrouche. Continuum: Detecting apt attacks through spatial-temporal graph neural networks, 2025. URL https://arxiv.org/abs/2501.02981.

P. Biondi, P. Lalet, G. Potter, G. Valadon, and N. Weiss. Scapy: Packet manipulation tool and network scanner. https://scapy.net/, 2025. Accessed: 2026-06-24.

Anshika Chaudhary, Himangi Mittal, and Anuja Arora. Anomaly detection using graph neural networks. *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 346–350, 2019. URL https://api.semanticscholar.org/CorpusID:204230429.

Zijun Cheng, Qiujian Lv, Jinyuan Liang, Yan Wang, Degang Sun, Thomas Pasquier, and Xueyuan Han. Kairos: Practical intrusion detection and investigation using whole-system provenance, 2023. URL https://arxiv.org/abs/2308.05034.

Matthias Fey, Jinu Sunil, Akihiro Nitta, Rishi Puri, Manan Shah, Blaž Stojanović, Ramona Bendias, Alexandria Barghi, Vid Kocijan, Zecheng Zhang, Xinwei He, Jan E. Lenssen, and Jure Leskovec. Pyg 2.0: Scalable learning on real world

graphs. In *Temporal Graph Learning Workshop @ KDD*, 2025. Accessed: Friday 10th July, 2026.

ffuf Authors. ffuf |kali linux tools, 2026. https://www.kali.org/tools/ffuf/ Accessed: Friday 10th July, 2026.

Hampus De Flon and Omar Sulaiman. Designing a proactive security solution for kubernetes clusters. Master's thesis, Chalmers University of Technology and Gothenburg University, 2025. URL http://hdl.handle.net/20.500.12380/310892.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018. URL https://arxiv.org/abs/1706.02216.

Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. Unicorn: Runtime provenance-based detector for advanced persistent threats. In *Proceedings 2020 Network and Distributed System Security Symposium*, NDSS 2020. Internet Society, 2020. doi: 10.14722/ndss.2020.24046. URL http://dx.doi.org/10.14722/ndss.2020.24046.

Emil Holmkvist Bergqvist. Graph neural network for early ddos detection: Evaluating the impact of progressive node reduction in flow graphs, 2025.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs, 2021. URL https://arxiv.org/abs/2005.00687.

Zian Jia, Yun Xiong, Yuhong Nan, Yao Zhang, Jinjing Zhao, and Mi Wen. MAGIC: Detecting advanced persistent threats via masked graph representation learning. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 5197–5214, Philadelphia, PA, August 2024. USENIX Association. ISBN 978-1-939133-44-1. URL https://www.usenix.org/conference/usenixsecurity24/presentation/jia-zian.

Abdalsamad Keramatfar, Mohadeseh Rafiee, and Hossein Amirkhani. Graph neural networks: a bibliometrics overview, 2022. URL https://arxiv.org/abs/2201.01188.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017. URL https://arxiv.org/abs/1609.02907.

Mikołaj Komisarek, Marek Pawlicki, Rafał Kozik, Witold Hołubowicz, and Michał Choraś. How to effectively collect and process network data for intrusion detection? *Entropy*, 23(11), 2021. ISSN 1099-4300. doi: 10.3390/e23111532. URL https://www.mdpi.com/1099-4300/23/11/1532.

Kubernetes Authors. About kubernetes, 2025. https://kubernetes.io/docs/concepts/ Accessed: Friday 10th July, 2026.

Shanxi Li, Qingguo Zhou, Rui Zhou, and Qingquan Lv. Intelligent malware detection based on graph convolutional network. *The Journal of Supercomputing*, 78:4182–4198, 08 2021. doi: 10.1007/s11227-021-04020-y.

Zhiyuan Liu and Jie Zhou. *Introduction to Graph Neural Networks*. Springer Cham, 2022. ISBN 978-3-031-01587-8. doi: https://doi.org/10.1007/978-3-031-01587-8.

Wai Weng Lo, Siamak Layeghy, Mohanad Sarhan, Marcus Gallagher, and Marius Portmann. E-graphsage: A graph neural network based intrusion detection system for iot. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, page 1–9. IEEE, April 2022. doi: 10.1109/noms54207.2022.9789878. URL `http://dx.doi.org/10.1109/NOMS54207.2022.9789878`.

Shaswata Mitra, Trisha Chakraborty, Subash Neupane, Aritran Piplai, and Sudip Mittal. Use of graph neural networks in aiding defensive cyber operations, 2024. URL `https://arxiv.org/abs/2401.05680`.

MITRE ATT&CK Authors. Exfiltration, 2025a. `https://attack.mitre.org/tactics/TA0010/` Accessed: Friday 10th July, 2026.

MITRE ATT&CK Authors. Lateral movement, 2025b. `https://attack.mitre.org/tactics/TA0008/` Accessed: Friday 10th July, 2026.

The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL `https://doi.org/10.5281/zenodo.3509134`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python, 2018. URL `https://arxiv.org/abs/1201.0490`.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '14, page 701–710. ACM, August 2014. doi: 10.1145/2623330.2623732. URL `http://dx.doi.org/10.1145/2623330.2623732`.

Panagiotis Radoglou Grammatikis, George Nakas, George Amponis, Sofia Giannakidou, Thomas Lagkas, Vasileios Argyriou, Sotirios Goudos, and Panagiotis Sarigiannidis. 5gcids: An intrusion detection system for 5g core with ai and explainability mechanisms. pages 353–358, 12 2023. doi: 10.1109/GCWkshps58843.2023.10464667.

Stefan Rommer, Peter Hedman, Magnus Olsson, Lars Frid, Shabnam Sultana, and Catherine Mulligan. *5G core networks: powering digitalization*. Academic Press, 2019.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs, 2020. URL `https://arxiv.org/abs/2006.10637`.

Sehan Samarakoon, Yushan Siriwardhana, Pawani Porambage, Madhusanka Liyanage, Sang-Yoon Chang, Jinoh Kim, Jonghyun Kim, and Mika Ylianttila. 5g-nidd: A comprehensive network intrusion detection dataset generated over 5g wireless network. *arXiv preprint arXiv:2212.01298*, 2022a.

Sehan Samarakoon, Yushan Siriwardhana, Pawani Porambage, Madhusanka Liyanage, Sang-Yoon Chang, Jinoh Kim, Jonghyun Kim, and Mika Ylianttila. 5g-nidd: A comprehensive network intrusion detection dataset generated over 5g wireless network, 2022b. URL `https://dx.doi.org/10.21227/xtep-hv36`.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification, 2021. URL `https://arxiv.org/abs/2009.03509`.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018. URL `https://arxiv.org/abs/1710.10903`.

Huanting Wang, Guixin Ye, Zhanyong Tang, Shin Hwei Tan, Songfang Huang, Dingyi Fang, Yansong Feng, Lizhong Bian, and Zheng Wang. Combining graph-based learning with automated data collection for code vulnerability detection. *IEEE Transactions on Information Forensics and Security*, 16:1943–1958, 2021. doi: 10.1109/TIFS.2020.3044773.

Shen Wang, Zhengzhang Chen, Xiao Yu, Ding Li, Jingchao Ni, Lu-An Tang, Jiaping Gui, Zhichun Li, Haifeng Chen, and Philip S. Yu. Heterogeneous graph matching networks, 2019. URL `https://arxiv.org/abs/1910.08074`.

Shakhzod Yuldoshkhujaev, Mijin Jeon, Doowon Kim, Nick Nikiforakis, and Hyungjoon Koo. A decade-long landscape of advanced persistent threats: Longitudinal analysis and global trends. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, CCS '25, page 3206–3220. ACM, November 2025. doi: 10.1145/3719027.3765085. URL `http://dx.doi.org/10.1145/3719027.3765085`.

Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks, 2019. URL `https://arxiv.org/abs/1909.03496`.