electronics

MDPI

Article

Graphical Representation of UWF-ZeekData22 Using Memgraph

Sikha S. Bagui [1,*], Dustin Mink [1], Subhash C. Bagui [2], Dae Hyun Sung [1] and Farooq Mahmud [1]

[1] Department of Computer Science, University of West Florida, Pensacola, FL 32514, USA; dmink@uwf.edu (D.M.); ds83@students.uwf.edu (D.H.S.); fm37@students.uwf.edu (F.M.)
[2] Department of Mathematics and Statistics, University of West Florida, Pensacola, FL 32514, USA; sbagui@uwf.edu
* Correspondence: bagui@uwf.edu

Abstract: This work uses Memgraph, an open-source graph data platform, to analyze, visualize, and apply graph machine learning techniques to detect cybersecurity attack tactics in a newly created Zeek Conn log dataset, UWF-ZeekData22, generated in The University of West Florida's cyber simulation environment. The dataset is transformed to a representative graph, and the graph's properties studied in this paper are PageRank, degree, bridge, weakly connected components, node and edge cardinality, and path length. Node classification is used to predict the connection between IP addresses and ports as a form of attack tactic or non-attack tactic in the MITRE framework, implemented using Memgraph's graph neural networks. Multi-classification is performed using the attack tactics, and three different graph neural network models are compared. Using only three graph features, in-degree, out-degree, and PageRank, Memgraph's GATJK model performs the best, with source node classification accuracy of 98.51% and destination node classification accuracy of 97.85%.

Keywords: graph data platform; graph database; Memgraph; graph machine learning; graph neural networks; node classification; MITRE ATT&CK framework

Citation: Bagui, S.S.; Mink, D.; Bagui, S.C.; Sung, D.H.; Mahmud, F. Graphical Representation of UWF-ZeekData22 Using Memgraph. *Electronics* **2024**, *13*, 1015. https://doi.org/10.3390/electronics13061015

Academic Editor: Kamal Berahmand

Received: 24 January 2024
Revised: 29 February 2024
Accepted: 3 March 2024
Published: 7 March 2024

1. Introduction

Addressing cybersecurity attacks in corporate networks is crucial, particularly considering the limitations of corporate resources in terms of manpower and budgetary constraints. Many organizations, especially smaller ones, may not have dedicated cybersecurity teams or the financial means to invest in robust security systems. A cyberattack occurs every 39 s [1], and in 2021, 43% of the attacks impacted small businesses [1], demonstrating the critical need for efficient methods to help resource constrained organizations. Machine learning, specifically graph-based machine learning, is a powerful tool for identifying potential attacks. However, many organizations lack the needed machine learning skills.

The uniqueness of this work lies in its using of Memgraph [2], an open-source graph database, to analyze, visualize, and detect cybersecurity attack tactics in a newly created dataset, UWF-ZeekData22. This dataset, composed of network logs, i.e., Zeek Conn logs, was generated in The University of West Florida's cyber simulation environment [3,4]. Graph characterization of this dataset was studied using PageRank, degree, bridge, weakly connected components, node and edge cardinality, and path length. Finally, node classification was performed on UWF-ZeekData22 using Memgraphs's graph neural networks (GNN). Node classification was used to predict the connection between IP addresses and ports as a form of attack or non-attack in the MITRE framework. Multi-classification was performed using three different graph neural networks models: GATJK, GraphSAGE, and GATV2. Results were compared and finally, three graph features were identified as attaining high source node and destination node classification accuracy.

Memgraph 2.8.0 is a freely available graph analytics software used to characterize and visualize graphs and perform graph machine learning through its various imported

libraries [2]. Memgraph is very similar in function to Neo4j [5,6], with a key difference being that Neo4j is written in Java and stores data on-disk, whereas Memgraph is written in C/C++ and stores data in-memory. This makes Memgraph more performant but limits the size of data loaded to the amount of available RAM on the local machine. Both software have built-in data science libraries, GDS for Neo4j, and MAGE for Memgraph. This research uses MAGE's built-in algorithms to create GNNs implemented in Torch and the Deep Graph Library (DGL) to classify network attack tactics, create visualizations of connections, and query the graph for supplemental information.

The rest of this paper is organized as follows. Section 2 presents the related works; Section 3 presents the data; Section 4 explains the data preprocessing; Section 5 explains Memgraph; Section 6 presents graph characterization or properties of UWF-ZeekData22; Section 7 presents graph visualizations of UWF-ZeekData22; Section 8 presents node classification results using graph neural networks; Section 9 presents the conclusions; and finally, Section 10 outlines future works.

2. Related Works

Javorník et al. [7] introduce the concepts of multi-step and multi-target attacks through Complex Privilege-Exploit Attack Graphs and Bayesian Privilege Attack Graphs. These structures model causal relationships among trust levels and guide decision-making. The system calculates the resilience of mission configurations using Bayesian networks, considering the distribution of critical privileges and the attacker's position. A case study of a medical information system demonstrates the use of the decision support system, highlighting the challenges of decision-making when most resilient configurations conflict with user demands.

Jacob et al. [8] discuss the problem of cyber security and the increasing need for improved detection of cyber-attacks on software applications using microservice architectures. The authors propose a graph-based anomaly detection approach to identify irregular microservice traffic caused by cyber-attacks. They use a graph convolutional neural network to capture spatial and temporal dynamics within the application's tracing data, which consists of the sequence of API calls between microservices. The authors also introduce a diffusion convolutional recurrent neural network for traffic forecasting and the detection of anomalies in microservice traffic.

Wei et al. [9] approach the problem of cyber threats by way of "proposed DeepHunter", a GNN-based approach that can match provenance data against known attack behaviors in a robust way. This approach aims to actively search for attack behavior in an organization's information system, using indicators of compromise (IOC). The DeepHunter GNN is able to capture the relationship between IOCs to provide better detection of advanced persistent threats for attack behaviors different from previously known attacks. This solution is determined to perform better than the comparable Poirot.

Hagheshenas et al. [10] frame the problem for application in smart grids, using a "Temporal graph neural network framework" for the detection and localization of false data injection and ramp attacks on the system state in smart grids. These forms of attack both manipulate sensor data to disrupt a grid's functionality. Examples of such data could be "voltage, current, power injections. . . status information of breakers and switches". Researchers have found promising results in classifying these forms of attack in a TGNN that models both the topological structure of the smart grids, and the "temporal measurements at each bus of the system".

Other works on graph databases have been conducted by [11–14]. Ref. [11] looked at how anomalies could be detected in node-labeled directed weighted graphs. Ref. [12] looked at the graph similarity model using the minimum description length principle. Ref. [13] presented the graph database analysis using Neo4j. Ref. [14] looked at attack graph analysis using temporal factors associated with vulnerabilities.

Although different forms of graph architectures have been used in the past, the uniqueness of our work lies in the use of, Memgraph characterizations, visualizations, and

node classification using multi-class graph neural networks. Moreover, this work uses Zeek Conn logs from the newly created UWF-ZeekData22 dataset.

3. The Data

The dataset used in this work, UWF-ZeekData22 [3], was generated in the cyber simulation environment at The University of West Florida (UWF). The Zeek Conn log dataset is labeled using the MITRE Adversarial Tactics, Techniques, and Common knowledge (ATT&CK) framework [15]. The MITRE ATT&CK framework, based on a foundation of threat models that determine adversary tactics, contains 14 tactics to date, and several techniques and sub-techniques. UWF-ZeekData22 contains 10 tactics, as shown in Table 1, with a total of 9,280,869 attack tactic records and 9,281,599 benign records [4]. A breakdown of this dataset's tactics is also presented in [3]. This research, however, focuses on the top three tactics by count: Reconnaissance (TA0043) [16], Discovery (TA0007) [17], and Credential Access (TA0006) [18].

Table 1. Count by Tactic: UWF-ZeekData22.

Tactic	UWF-ZeekData22 Count
None	9,281,599
Reconnaissance	9,278,722
Discovery	2086
Credential Access	31
Privilege Escalation	13
Exfiltration	7
Lateral Movement	4
Resource Development	3
Persistence, Initial Access, Defense Evasion	1
Execution	0
Command and Control	0
Defense Evasion	0
Initial Access	0
Initial Access, Persistence	0

Adversaries using Reconnaissance are actively and passively looking for ways to gather information about the system as a whole so that they can plan attacks. Adversaries performing Discovery tactics are gathering information about the internals of the system and environment so that they can plan their attacks. Adversaries using Credential Access tactics are looking for ways to steal credentials like user IDs and passwords.

4. Data Preprocessing

To create this graph network in Memgraph, the dataset was preprocessed. This preprocessing flow is presented in Figure 1. The goal of the preprocessing was to transform log data to representative graph nodes and edges CSV files which could be ingested into Memgraph.

The first preprocessing step was to remove the tactics that had very few occurrences. There is too little data in these low frequency tactics to do any meaningful graph analysis. Only the Reconnaissance, Discovery, and Credential Access tactics were kept from the UWF-ZeekData22 dataset.

The second preprocessing step was to combine the four source/destination IP address/port columns into two address columns. The address value would become the node labels formatted as IP address–port. The ports should not be nodes because port 80 associated with an IP address is not the same port 80 associated with a different IP address.

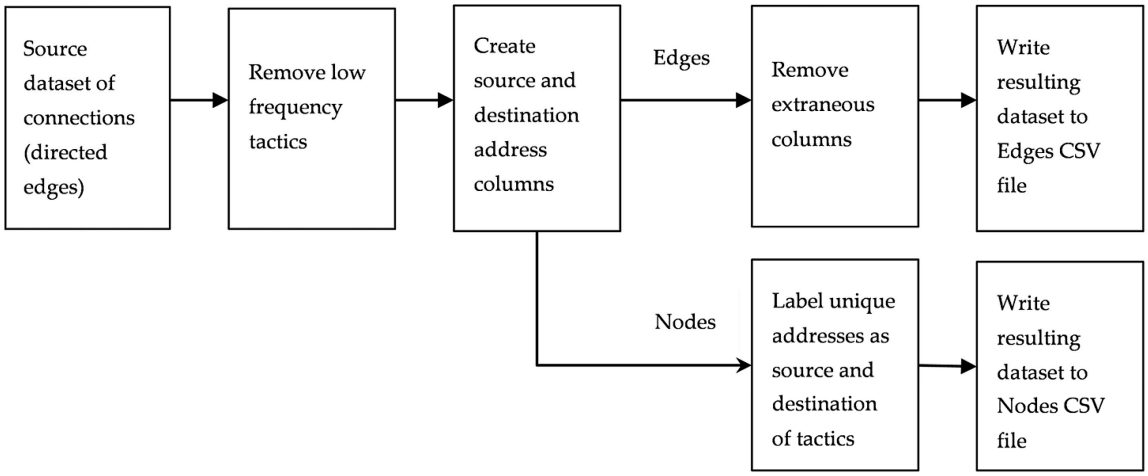

Figure 1. Preprocessing flow chart.

Once the IP addresses were in the proper format, distinct addresses were collected and joined on the address to the edges. The tactics_src and tactics_dest column values were parsed from the edges and added as labels to the nodes CSV file.

The source dataset contained 24 columns. Since we are concerned with the structure and pattern of connections made to identify tactics of attacks under the MITRE ATT&CK framework, we removed all columns except the addresses and labels of tactics which define the graph structure. Therefore, only the datetime, source address, destination address, and tactic columns were kept. This significantly reduced the size of the dataset to be processed. The resulting dataset was written to the edges CSV, and the nodes and edges CSV files were ready to be ingested into Memgraph.

The columns in the original dataset and the nodes and edges datasets are presented in Figure 2. Note the presence of address ID columns in the nodes and edges dataset. This creates a relational dataset which Memgraph uses to define the edges between the source and destination nodes.

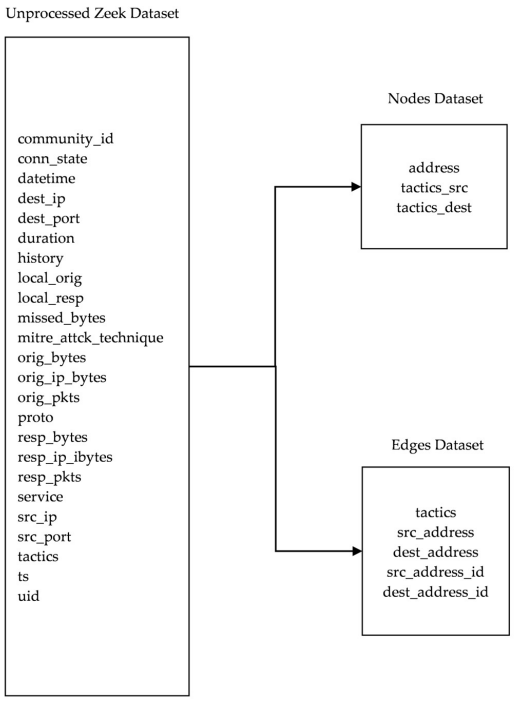

Figure 2. Attributes before and after preprocessing.

5. Memgraph

Memgraph is a high-performance, in-memory graph data platform designed for ingesting, querying, and visualizing large-scale graph data. Memgraph efficiently processes graph queries and traverses' large networks of nodes and edges using the cypher query language [2]. It is freely available and can be run inside a docker container, thus making setup easy.

Ingestion into Memgraph

The first step is to create the graph database. This is accomplished using the code below. The code does the following:

1. Loads the nodes dataset from disk (Figure 3).
2. For each row in the dataset, a Memgraph node is created. The node contains an address attribute.
3. An index is created on the address attribute. This speeds up edge creation and creates an edges dataset.
4. Load the edges dataset from disk.
5. For each row in the edges dataset:
    a. Obtain the source and destination address nodes using the MATCH clause.
    b. Create an edge between the two nodes.
    c. Set the tactic attribute on the edge.

```
LOAD CSV FROM "/nodes.csv" WITH HEADER AS row
CREATE (n:NetworkNode {address: row.address});
CREATE INDEX ON:NetworkNode(address);
LOAD CSV FROM "/edges.csv" WITH HEADER AS row
MATCH (src:NetworkNode {address: row.src_address})
MATCH (dest:NetworkNode {address: row.dest_address})
CREATE (src)-[e:COMMUNICATES_WITH]->(dest)
SET e.tactic = row.tactics;
```

Figure 3. Cypher query: creating the nodes and edges.

Once the nodes and edges have been created (Figure 3), the networks can be created by variations shown in Figure 4. The query, Figure 4, uses two useful functions:

1. A WHERE clause for filtering by tactic.
2. A LIMIT clause for returning a subset of nodes. Since the Reconnaissance tactic has many nodes, obtaining a full result set not only takes time but also does not offer any additional insights into the structure. Experimentation showed that setting the limit to about 5000 nodes gave an adequate representation of the graph's structure.

```
MATCH (src)-[comm]->(dest)
WHERE comm.tactic = "[name of tactic]"
RETURN src, comm, dest
LIMIT [number of nodes to return]
```

Figure 4. Cypher query: creating the network.

6. Graph Characterization

Graphs are complex structures whose topology can vary depending on the number of nodes, edges, and whether they are directed or not. Mathematically, a graph is a representation of a set of objects (nodes) connected by links (edges). In the context of

this work, a node represents an individual IP address. An edge represents the connection between two nodes, a source IP address, and destination IP address. The edge is the attack tactic that is used. In other words, the graph represents an attack modality from a source computer (the attack initiator) to the target computer (the attack victim). Having a source and target node implies direction; hence, the edge is "directed", and the graph is a "directed graph". The degree of a node in a graph refers to the number of edges that are incident to that node. In a directed graph, the degree can be split into two separate measures: the in-degree (number of incoming edges) and the out-degree (number of outgoing edges).

Hence, when analyzing a graph, the following characteristics or properties are of interest: (i) PageRank; (ii) degree; (iii) bridges; (iv) weakly connected components; (v) node and edge cardinality; (vi) path Length. These characteristics, obtained using Cypher queries, are discussed next.

6.1. PageRank

6.1.1. PageRank

Google created the PageRank algorithm to distinguish recognizable and relevant web pages from lesser known pages. The algorithm uses the web's link structure to calculate PageRank scores for each document on the web [19,20]. The algorithm incorporates the concept of a "random user" in the following fashion:

1. The random web surfer starts on a random web page. This surfer follows the links on the page and clicks randomly on one of the links.
2. After clicking a link, the surfer is now on the new page and repeats the process by randomly clicking on one of the links on that page.
3. This process continues indefinitely, with the random surfer randomly clicking on links and moving from page to page.

The PageRank algorithm calculates the probabilities of the random surfer being on a specific page at any given time. These probabilities are represented as the PageRank scores for each page. Pages that are frequently visited or are linked to by other important pages tend to have higher probabilities and therefore higher PageRank scores.

The random surfer concept can be extended to model the behavior of a cyber attacker targeting machines on a network. Using the PageRank algorithm, one can calculate the probability that a random cyber attacker will attack a particular machine by randomly choosing machines on the network. The graph data are a historical record of attacks. The PageRank score evaluates the likelihood of another attack using past attacks.

Memgraph natively implements PageRank using a straightforward calculation. The iterative nature of the algorithm allows Memgraph to parallelize the score computation across multiple processor cores [20]. The execution times of running PageRank for the benign data (None), Reconnaissance, and Discovery using Memgraph are presented in Table 2.

Table 2. PageRank execution time.

Tactic	Row Count	Execution Time (ms)
None	9,281,599	13,554
Reconnaissance	9,278,722	13,198
Discovery	2086	47
Tactic	Row Count	Execution time (ms)
None	9,281,599	13,554

The cypher query, Figure 5, was used to generate the PageRank scores for the tactics of interest shown in subsequent sections.

```
MATCH n = ()-[e]->()
WHERE e.tactic = "[Tactic]"
WITH project(n) AS proj
CALL pagerank.get(proj) YIELD node, rank
SET node.rank = rank
RETURN node.address AS address, rank
ORDER BY rank DESC
LIMIT 10;
```

Figure 5. Cypher query: generating the PageRank scores.

6.1.2. PageRank Scores for the Reconnaissance Tactic

Table 3 presents the PageRank scores for the top 10 IP addresses for the Reconnaissance tactic. As per Table 3, the IP address 143.88.5.1:53 in UWF-ZeekData22 is the most likely to be attacked using the Reconnaissance tactic. Based on these results, more resources should be allocated to protecting 143.88.5.1:53.

Table 3. Reconnaissance: top 10 PageRank scores.

Address	PageRank Score
143.88.5.1:53	0.036933
143.88.7.1:443	0.002461
143.88.7.12:8080	0.001996
143.88.7.11:631	0.001677
143.88.7.15:135	0.001384
143.88.2.10:80	0.000762
143.88.7.12:22	0.000697
143.88.2.10:443	0.000673
143.88.7.12:80	0.000621
143.88.7.11:21	0.000617

6.1.3. PageRank Scores for the Discovery Tactic

Table 4 shows that IP address 143.88.2.12:22 in UWF-ZeekData22 is the most likely to be attacked using the Discovery tactic. Based on these results, more resources should be allocated to protecting 143.88.2.12:22.

Table 4. Discovery: top 10 PageRank scores.

Address	PageRank Score
143.88.2.12:22	0.003656
143.88.2.12:1	0.001554
143.88.2.12:443	0.000954
143.88.2.12:80	0.000954
143.88.7.10:3	0.000909
143.88.2.12:1999	0.000654
143.88.2.12:5907	0.000654
143.88.2.12:8443	0.000654
143.88.2.12:3945	0.000654
143.88.2.12:8001	0.000654

6.2. Degree

Degree is the second category of interest when analyzing the graph. We analyze the degree centrality, degree distribution as well as the degree averages [21,22].

6.2.1. Degree Centrality

Degree centrality measures the number of edges connected to a node, normalized by the number of nodes in the graph, such that degree centrality = degree/(number of nodes −1). This measure is useful to view the connectedness of nodes based on the tactic being used. For directed graphs, the user specifies the "in" or "out" parameter for the type of degree, or the method defaults to "undirected." In-degree centrality is the degree centrality calculated value, where degree is the number of incoming edges. Likewise, out-degree centrality is the degree centrality calculated value where degree is the number of outgoing edges. Figure 6 sets the in and out degrees as a property of each node.

```
CALL degree_centrality.get("in") YIELD node, degree
SET node.in_degree = degree;

CALL degree_centrality.get("out") YIELD node, degree
SET node.out_degree = degree;
```

Figure 6. Cypher query: setting in- and out-degree.

In-degree centrality and out-degree centrality are shown for the Reconnaissance tactics. Table 5 shows that for UWF-ZeekData22, there is no significant difference in the in-degree values between the Reconnaissance and None (benign data) tactics. They appear to be extremely similar in distribution.

Table 5. In-degree centrality as tactics subgraphs.

Reconnaissance	Reconnaissance	None	None
Address–Port	**In-Degree**	**Address–Port**	**In-Degree**
143.88.5.1:53	27.50483817	10.0.10.1:53	27.6587867
143.88.7.15:135	3.174730286	143.88.11.1:53	2.40422696
143.88.7.1:443	1.936158381	8.8.8.8:53	1.87328573
143.88.7.12:8080	1.665665666	8.8.4.4:53	1.86731302
143.88.7.11:631	1.352463575	143.88.1.1:53	1.65291272
143.88.7.12:22	0.654877099	ff02::1:2:547	0.27426719
143.88.7.15:1	0.654877099	143.88.255.10:53	0.22335524
143.88.7.11:80	0.512512513	143.88.0.41:53	0.10713022
143.88.7.11:21	0.498276054	172.28.128.255:138	0.08518095
143.88.7.12:80	0.441330219	172.28.128.255:137	0.06999827

Table 6 shows the out-degree centrality for the Reconnaissance and None (benign) nodes. These results show that the connectedness of the Reconnaissance nodes are significantly greater than the normal "none" tactic nodes, even though the number of edges are similarly in the millions. This outlier may be a strong indicator of an attack using the Reconnaissance tactic.

6.2.2. Degree Averages

Degree averages show the differences between tactics and their connectedness. These values are highly correlated with the degree centrality and provide more raw data but do not divide by the number of nodes. Table 7 shows the out-degree averages for each of the tactics—Reconnaissance, Credential Access, Discovery—as well as for benign data.

Table 6. Out-degree centrality as tactics subgraphs.

Reconnaissance	Reconnaissance	None	None
Address–Port	**Out-Degree**	**Address–Port**	**Out-Degree**
143.88.2.10:53565	14.69202536	fe80::250:56ff:fe9e:5457:546	0.17432725
143.88.2.10:41562	14.46424202	172.28.128.3:138	0.08518095
143.88.2.10:58517	14.40729619	172.28.128.3:137	0.06999827
143.88.2.10:51130	14.33611389	143.88.11.10:3	0.05328951
143.88.2.10:38774	14.29340452	143.88.1.50:138	0.05276711
143.88.2.10:54736	14.29340452	143.88.11.10:68	0.05139322
143.88.2.10:35962	14.25069514	143.88.11.14:3	0.04899509
143.88.2.10:43921	14.23645868	143.88.1.50:137	0.04237246
143.88.2.10:62815	14.23645868	143.88.11.10:50888	0.0398427
143.88.2.10:44715	14.23645868	143.88.11.10:53887	0.03973987

Table 7. Out-degree averages for UWF-ZeekData22.

Tactic	Source Out-Degree Average	Destination In-Degree Average
Reconnaissance	743.321	1673.07
None (benign)	40.031	814.657
Credential Access	1.292	4.429
Discovery	43.31	2.078

6.3. Bridges

Bridge is the third category of interest when analyzing the graph. A bridge is a single edge that connects subgraphs together, which when removed, would result in the two subgraphs losing connection. Looking at bridges can provide additional information about the structure of our graphs. Figure 7 yields the bridge count, and Table 8 shows the bridge count by tactic for UWF-ZeekData22.

```
MATCH n = ()-[e]->()
WHERE e.tactic = "[Tactic]"
WITH project(n) AS proj
CALL bridges.get(proj) YIELD node_from, node_to
RETURN count(*);
```

Figure 7. Cypher query for count of bridges.

Table 8. Bridge counts by tactic.

Tactic	Bridge Count
Reconnaissance	2
None (Benign)	5742
Credential Access	18
Discovery	1866

The number of bridges in None compared to Reconnaissance supports the idea that the Reconnaissance tactic is heavily connected between nodes.

6.4. Weakly Connected Components

Weakly connected components is the fourth category of interest when analyzing the graph. Given a directed graph, a weakly connected component (WCC) is a subgraph of the original graph where all vertices are connected to each other by some path, ignoring the direction of edges [23]. These values give an idea of how many disparate subgraphs exist by tactic. It shows that the Reconnaissance tactic has many disparate components,

whereas None (the benign data) has very few disparate components, even though both have a similar number of connections in the millions (Table 9).

Table 9. Weakly connected components.

Tactic	Components Count
Reconnaissance	930
None (Benign)	34
Credential Access	7
Discovery	4

6.5. Node and Edge Cardinality

Node and edge cardinality is the fifth category of interest when analyzing the graph. Figure 8 shows the code snippet of the cypher queries returning the number of nodes and edges, respectively. The data for UWF-ZeekData22 resulted in a graph with 262,963 nodes and 18,562,438 directed edges (edges from the source to target address). Table 10 shows the node and edge cardinality for the tactics of interest as given by Figure 9.

```
MATCH (n)
RETURN count(n);

MATCH ()-[e]->()
RETURN count(e);
```

Figure 8. Cypher query returning the number of nodes and edges.

Table 10. Path length.

Tactic	Edges
None (Benign)	9,281,599
Reconnaissance	9,278,722
Discovery	2086
Credential Access	31
Resource Development	0

```
MATCH n = ()-[e]->()
RETURN e.tactic, count(e) as cnt
ORDER BY cnt DESC;
```

Figure 9. Node and edge cardinality for tactics of interest.

6.6. Path Length

Path length is the sixth category of interest when analyzing the graph. It is valuable to know if there are nodes that are more than one hop away from a source node. The query in Figure 10 returns paths that are of length 2 to 5 edges away from source to destination, where the source is a source of the Reconnaissance tactic, limited to 1000 paths.

```
MATCH path = (src)-[e * 2..5]->(dest)
WHERE src.src_reconnaissance = 1
RETURN path
LIMIT 1000
```

Figure 10. Returns paths of length 2 to 5 edges away.

Figure 11 shows a source node of the Reconnaissance tactic, with address "143.88.2.10:41562" that connects to intermediate address "143.88.7.12:3", then to address "143.88.2.10:3", and then to many destination addresses.

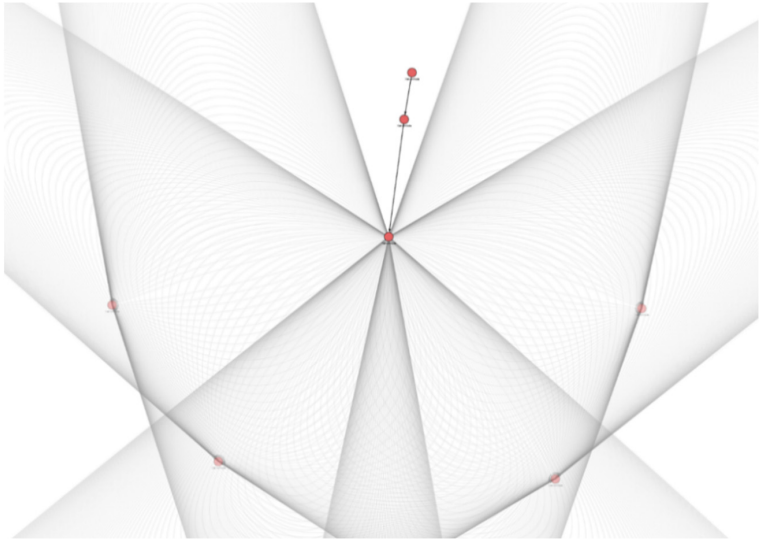

Figure 11. A node of the Reconnaissance tactic to destination addresses.

Figure 12 shows a source node of None (benign traffic), with path lengths of 2.

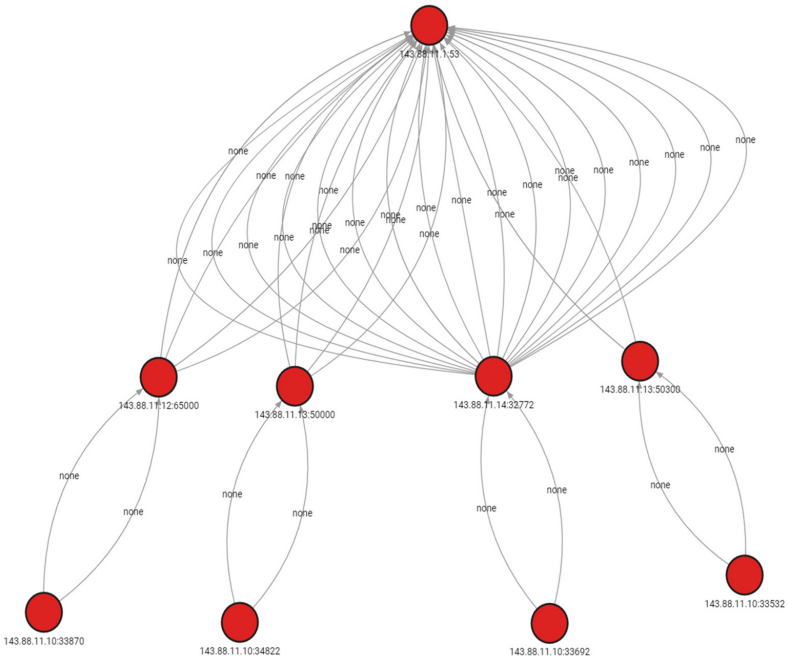

Figure 12. Node of None (benign traffic) with path length of 2.

Figure 13 shows a source node of the Discovery tactic with address "143.88.7.10:55262" that connects to intermediate address "143.88.2.10:3" and then to many destination addresses for dataset UWF-ZeekData22.

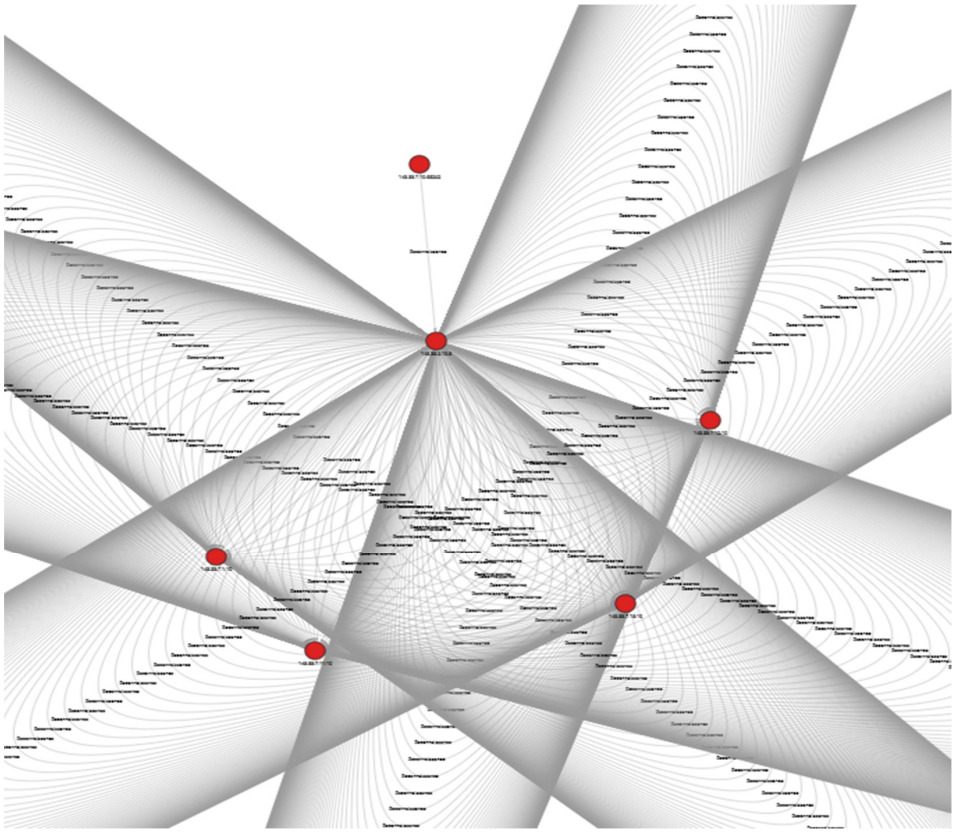

Figure 13. Nodes of the Discovery tactic to destination addresses.

The paths shown in Figures 11–13 are the result of running the query in Figure 10 and represent the way that different tactic types connect to addresses, with a focus on connections that pass through an intermediate IP address/port number before connecting to their destination nodes. The source nodes of Credential Access do not have any paths of length greater than 1 for dataset UWF-ZeekData22; hence, they have not been displayed.

7. Graph Visualizations

Figure 14 presents graphs representing the whole graph or sub-graphs of the three major attack tactics as well as benign data from the UWF-ZeekData22 dataset. The graphs help visualize the way different tactics connect between nodes. For instance, the graph with connections using the tactic "Discovery" shows a central node connecting to most other nodes, and some nodes connecting over an intermediate node in the path. Looking at the "none" data, connections show a node connected to thousands of other nodes, with a low degree of connections. In contrast, the nodes for the Reconnaissance tactic show a very high degree of interconnectedness. The degree of connections for Reconnaissance appears to be a strong indicator that the node belongs to that tactic.

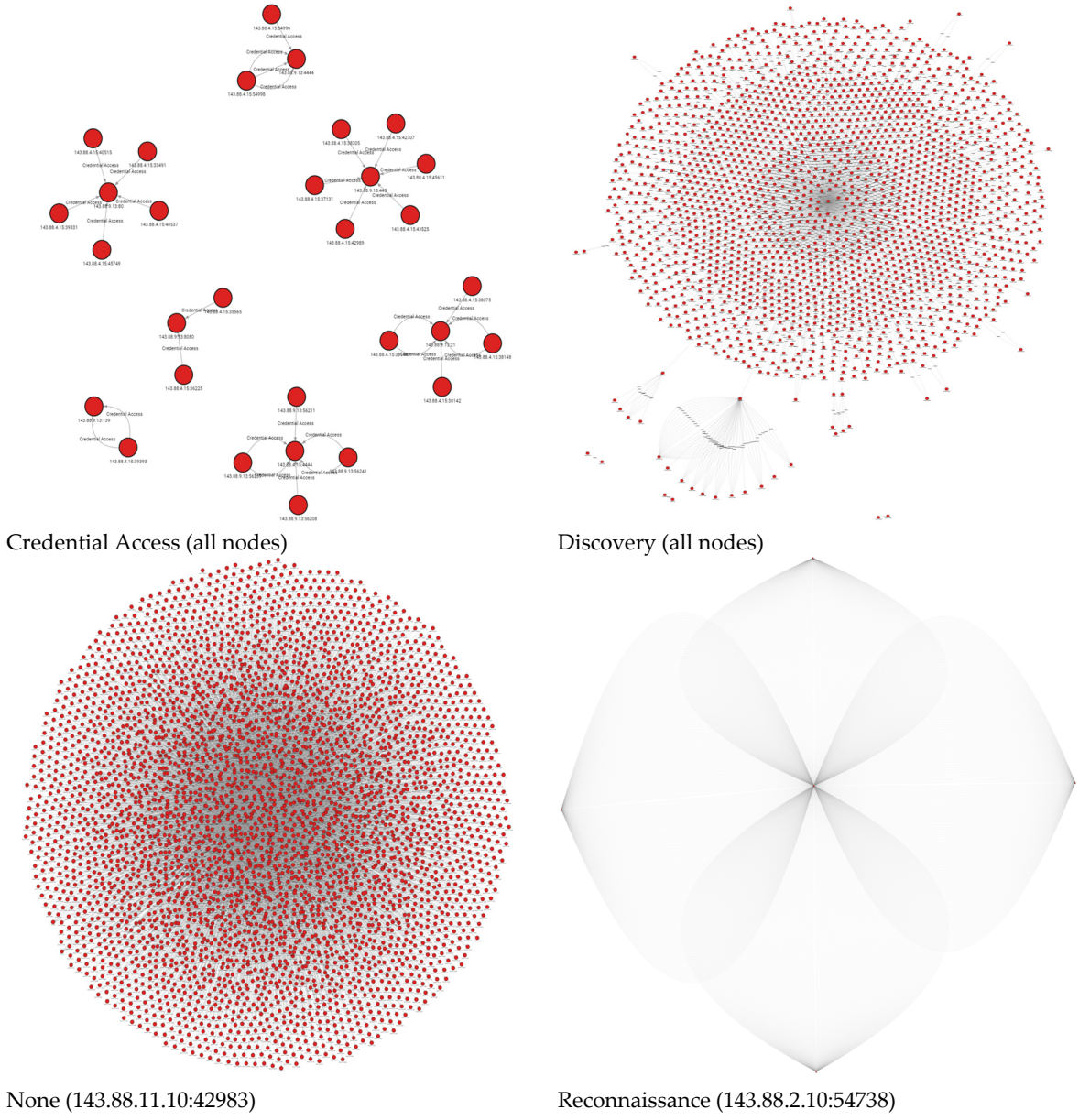

Figure 14. UWF-ZeekData22 visualization.

The code snippet of Figure 15 was used to create Figure 16, showing 10,000 node connections. The graph shows a large collection of nodes that connect among a set of central nodes. These central nodes appear to connect either directly or through intermediate nodes, with a low degree of connection from the central nodes to the outer nodes.

```
MATCH n = ()-[e]-()
WHERE e.tactic = "none"
RETURN * LIMIT 10000;
```

Figure 15. Showing 10,000 nodes of the benign data.

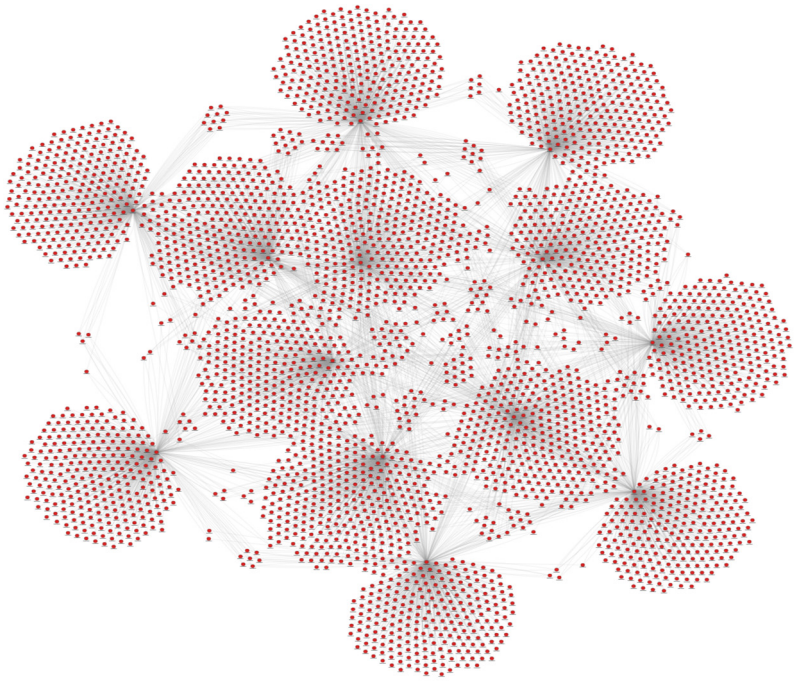

Figure 16. Showing 10,000 nodes of "none" tactic.

Figure 17 is a close up of Figure 16, offering a view of the connections at node address "199.7.91.13:53". The node is connected to many other nodes, with one or two edges for most connections.

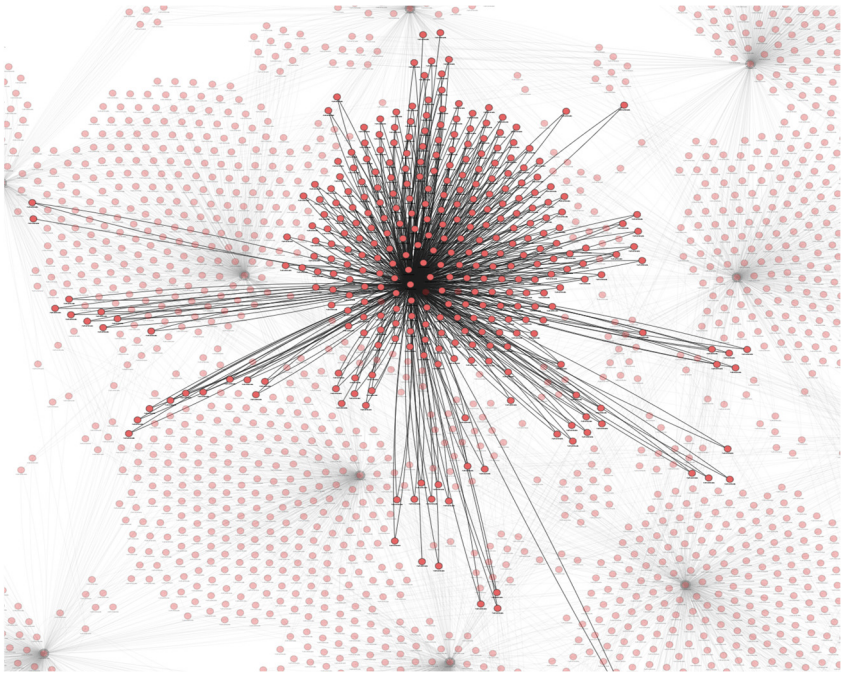

Figure 17. Highlight of a highly connected node of "none" (benign data).

8. Node Classification Using Graph Neural Networks

In this work, node classification, performed using graph neural networks (GNNs), is used to predict the connection between IP addresses and ports as a form of attack or non-attack in the MITRE framework of UWF-ZeekData22; that is, node classification is used to classify a node as a source or a destination of an attack tactic and is used to identify

the IP address–port combination that is the source of an attack for the different attack tactics (hence, multi-classification). In Memgraph, since node classification is conducted using GNNs, there are three different main layer types available for selection [24]:

1. Graph attention networks jumping knowledge (GATJK).
2. Graph attention networks (GAT, GATv2).
3. "GraphSAGE" (inductive representation learning on large graphs).

The following parameters are provided to train the models: the hidden features layers, layer type, learning rate, number of epochs, training testing split ratio, and weight decay. In this work, node classification is performed using the torch open-source machine learning library.

The layer type defines the type of layers in GNN, such as GATJK, GAT, GraphSAGE, etc. The learning rate defines the amount that the GNN will adjust during learning to correct its weights and minimize loss. The number of epochs defines the number of times that a model passes through the training dataset; that is, if the number of epochs equals five, the model will pass the whole dataset five times. Split ratio defines the ratio between training and testing data. Weight decay defines an additional loss to weights that grow too big, such that a single weight does not monopolize the result. Table 11 presents an example of node classification training parameters.

Table 11. Example of node classification training parameters.

Aggregator	Mean
checkpoint_freq	5
console_log_freq	5
device_type	cuda
hidden_features_size	[16, 16]
layer_type	GATJK
learning_rate	0.1
Metrics	["loss","accuracy","f1_score","precision","recall", "num_wrong_examples"]
node_id_property	id
num_epochs	100
path_to_model	/tmp/torch_models/model_GATJK_
split_ratio	0.8
weight_decay	0.0005

GAT and GATJK use the torch geometric library to apply graph attention network layers in addition to jumping knowledge. GAT creates a matrix of weights embedded on each node for the neighbors of the node and to itself. These standard weight values are supplemented by the "attention coefficient." This attention coefficient characterizes the importance of the relationship between a node to its neighboring node, or how much attention to give the neighboring node. This concept is used in language learning models as well. The attention coefficient, as conducted in [25], is calculated as an additional single-layer neural network that takes in the weights of two nodes, performs a "LeakyReLU" activation function, and normalizes them performing "softmax." The final weights are then multiplied against the attention coefficient, adjusting the weights for which neighboring nodes require the most attention per the additional single-layer neural network [26]. These calculations are performed for pairs in parallel, allowing for additional performance gain [25]. In the context of Memgraph's "GATJK" model, the torch GAT model is the given parameter jk ("Jumping Knowledge") with a string value "max", which performs a final linear transformation to transform node embeddings to the expected output feature dimensionality [27].

Jumping knowledge can be applied on top of GAT, GraphSAGE, and other graph neural network types. This is a way to be more agnostic with respect to specific graph structures such that a trained neural network for one graph might work just as well on

another graph with a different structure via the way jumping knowledge influences the aggregation of neighbor information for its embeddings [28].

GraphSAGE is another graph neural network model, which focuses on creating a general inductive framework that leverages node feature information [29]. The general inductive approach allows for GraphSAGE to be used across different graphs without losing as much information because instead of training individual embeddings for each node, we learn a function that generates embeddings by sampling and aggregating features from a node's local neighborhood [29].

Node classification required additional preprocessing as well as feature selection, which are presented next.

8.1. Preprocessing for Node Classification

Additional preprocessing was required for node classification on the UWF-ZeekData22 dataset. First, an attack tactic label was required for each node in the graph. The labeling was separated between two classifications: source and destination. This was undertaken to classify a node as a source of an attack or a destination of an attack based on its features, as predicted by the trained model.

Table 12 shows the tactic labels that were assigned to each node for both the src_class (source) and dest_class (destination) node properties, converting categorical strings to integer values for the node classification algorithm. "None" is a benign connection, whereas "no_conn" means there is no connection for this node. Hence, a node that is a source of a connection but has no incoming connection will have dest_class = 7 as it is not a destination of any connections. A node can be a source or destination of multiple tactics, and labels 5 and 6 denote such values.

Table 12. Tactic label values.

Tactic	Label
None	1
Reconnaissance	2
Discovery	3
Credential Access	4
Discovery, Reconnaissance	5
Reconnaissance, none	6
no_conn	7

Table 12's labels would be considered "multi-class" labels. In addition to this, binary labels were created for each tactic for both source and destination. Table 13 shows the node properties.

8.2. Feature Selection for Node Classification

As per the earlier analysis and visualizations that show the differences in degree and centrality of the various nodes by tactic, in-degree, out-degree, and PageRank were selected as graph features for node classification. The benefit of choosing these three features is that these values model the structure and layout of the graph without requiring highly specific network log data, while providing information about the amount of traffic and unique connections being made from each node. Node classification is performed purely based on the connectivity characteristics of the nodes to determine the underlying intention. In-degree is most relevant to nodes that receive network traffic (destination nodes), out-degree is most relevant to nodes that send traffic (source nodes), and PageRank provides data by scoring nodes that are most relevant and central to the graph. These graph features were strongly correlated to the classification of a node. Figure 18 presents the code snippet used for feature selection.

Table 13. Node property descriptions.

Node Property	Description	Example Value
address	Concatenation of IP address and port	143.88.7.10:54124
dest_class	Numeric label of node as a destination of a tactic	7
dest_credential_access	Binary label of node as destination of Credential Access tactic	0
dest_discovery	Binary label of node as destination of Discovery tactic	0
dest_no_conn	Binary label of node as destination of no connections	1
dest_none	Binary label of node as destination of None tactic (benign, normal connection)	0
dest_reconnaissance	Binary label of node as destination of Reconnaissance tactic	0
features	Array of feature values {in-degree, out-degree, PageRank}	Array [3]
Discovery, Reconnaissance	5	[0,
Reconnaissance, none	6	0.0009735246917805615,
no_conn	7	0.000002120804881073081]
in_degree	In-degree of node	0
out_degree	Out-degree of node	0.000973525
rank	PageRank of node	2.1208×10^6
src_class	Numeric label of node as a source of a tactic	2
src_credential_access	Binary label of node as a source of Credential Access tactic	0
src_discovery	Binary label of node as a source of Discovery tactic	0
src_no_conn	Binary label of node as a source of no connections	0
src_none	Binary label of node as a source of None tactic (benign, normal connection)	0

```
CALL degree_centrality.get("in") YIELD node, degree
SET node.in_degree = degree;

CALL degree_centrality.get("out") YIELD node, degree
SET node.out_degree = degree;

CALL pagerank.get() YIELD node, rank
SET node.rank = rank;
```

Figure 18. Cypher query for feature selection.

8.3. Node Classification Results and Discussion

Node classification was performed for the different layer types: GATJK, GraphSAGE, and GATv2. GATJK was initially performed using the default parameters. Then, experimentation with different learning rates as well as different epochs produced a set of optimal parameters that were used to then run the other two layer types: GraphSAGE and GATv2. Eventually the three models were compared using the optimal parameters. Being a multi-class model, the results presented reflect all classes; that is, classes 1–7 of Table 12.

8.3.1. GATJK

GATJK was performed using the default model. Based on the results of the default model, the learning rates and epochs were adjusted in order to obtain the best results.

Default Model

The initial set of parameters for creating the model for source node classification were as follows: {hidden features: [16, 16]; layer type: "GATJK"; learning rate: 0.1; weight decay 0.0005; split ratio: 0.8; epochs: 100}. The results are shown in Figure 19.

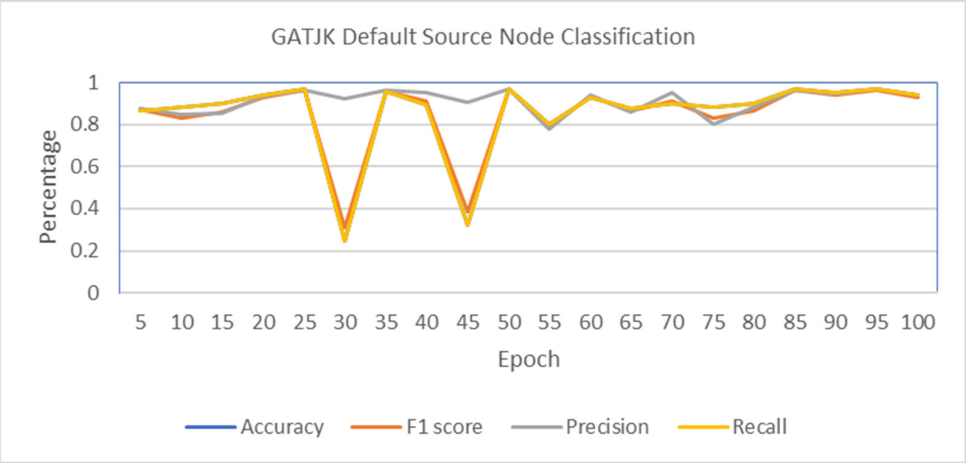

Figure 19. GATJK source node classification metrics.

As can be seen from Figure 19, the validation accuracy, F1 score, precision, and recall have a similar trajectory. Of note, the accuracy values are very close to recall; hence, they are difficult to see. For source node classification, throughout the epochs, the accuracy rises steadily until it plummets and cycles through this process when the learning rate is at 0.1. The chart visualizes how the accuracy, F1 score, and recall fall dramatically in very select training epochs, although generally, the performance is very high. This may be a sign that the learning rate is too high. It also might be because there are multiple attack tactics (since this is being run as a multi-classifier), and the distribution of the attack tactics is not even. The performance of the model continues to rise until the 25th epoch, after which, it drops and rises dramatically.

Figure 20 shows the GATJK default source node classification loss. The objective of the model is to minimize loss. The training loss was low and more or less stable with the default parameters; however, the validation loss was not very consistent and pretty high in many cases. Next, experimentation was conducted with the learning rate.

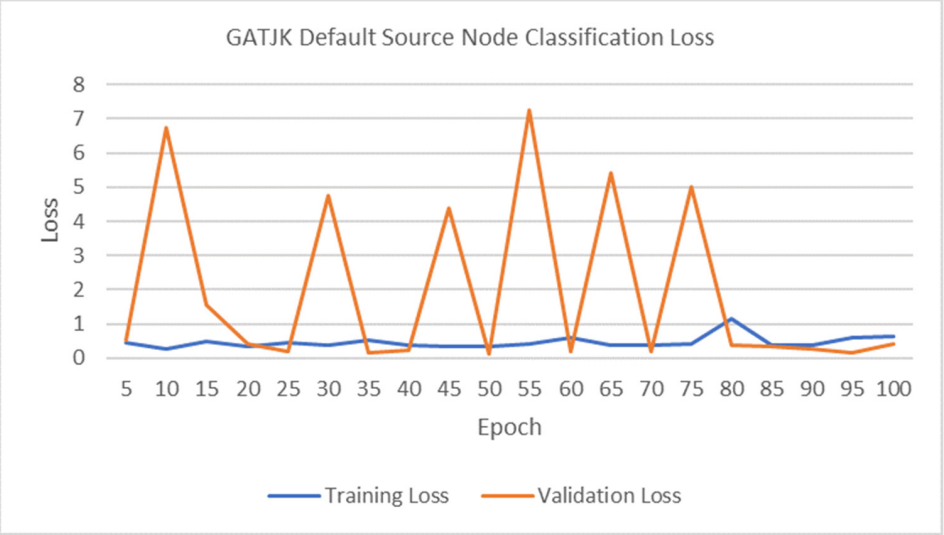

Figure 20. GATJK default source node classification loss.

Varying the Learning Rates

In the source and destination classification models, the learning rate was reduced to 0.001, and the layers were adjusted to [8, 8]. The following parameters were used: {hidden features: [8, 8]; layer type: "GATJK"; learning rate: 0.001; weight decay 0.0005; split ratio: 0.8; epochs: 100}. This is shown in Figure 21.

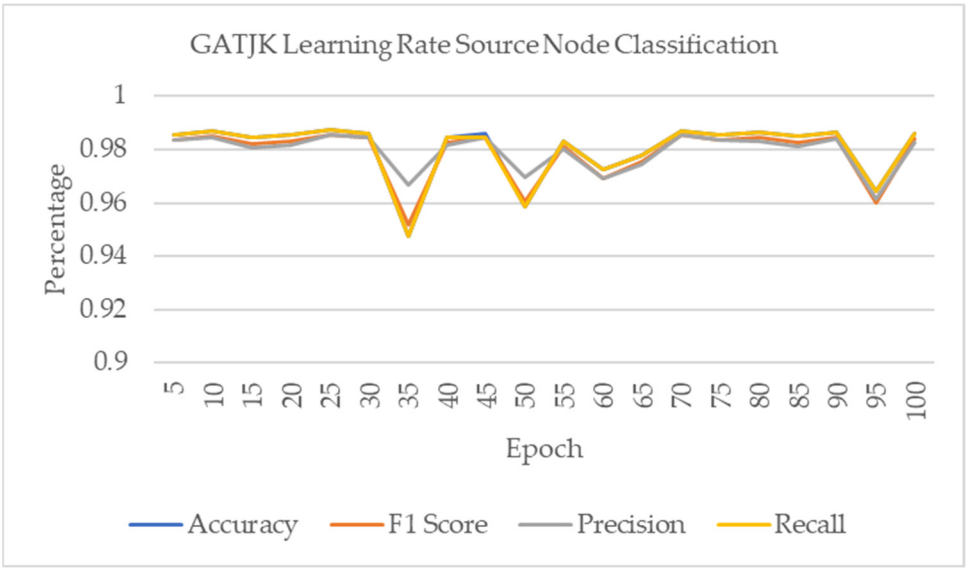

Figure 21. GATJK learning rate source node classification metric.

As noted from Figure 21, the range of performance is much reduced such that the peak stays high and the lows only fall to greater than 0.945 for any measured value as opposed to approaching 0.2 in our previous attempt. By lowering the learning rate, we can lower losses by an order of magnitude and maintain high metrics across all measures. The performance stops improving at around 10 epochs, so that is likely the optimum epoch to avoid overfitting the model to our dataset.

Figure 22 shows that training loss is lowered to the 0.05–0.06 range. The validation loss fluctuates between 0.04 and 0.12.

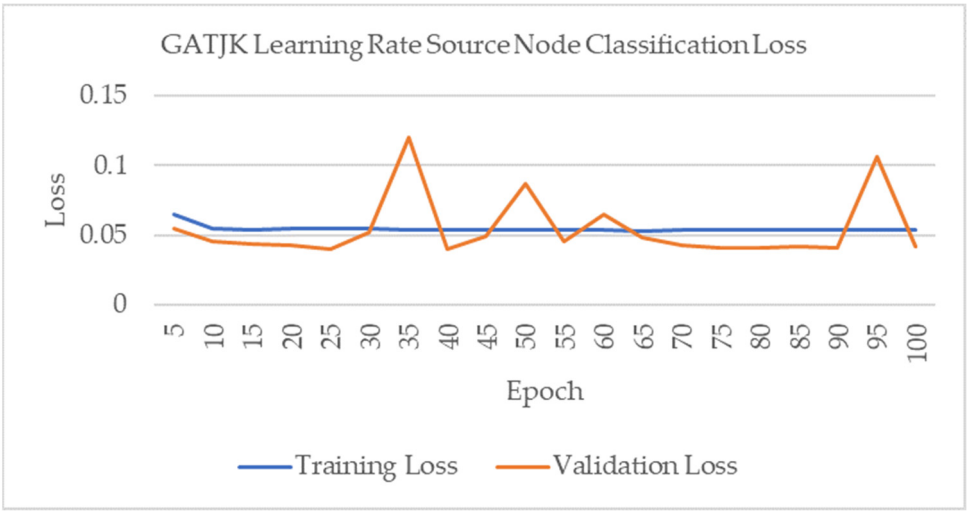

Figure 22. GATJK learning rate source node classification loss.

Figure 23 presents a lower learning rate of 0.001. There is significant improvement in loss over epochs with a lower learning rate of 0.001. Interestingly, the dips in metrics occur at approximately the same points, i.e., 30 to 35 epochs and 45 to 50 epochs, as shown in Figures 22 and 23.

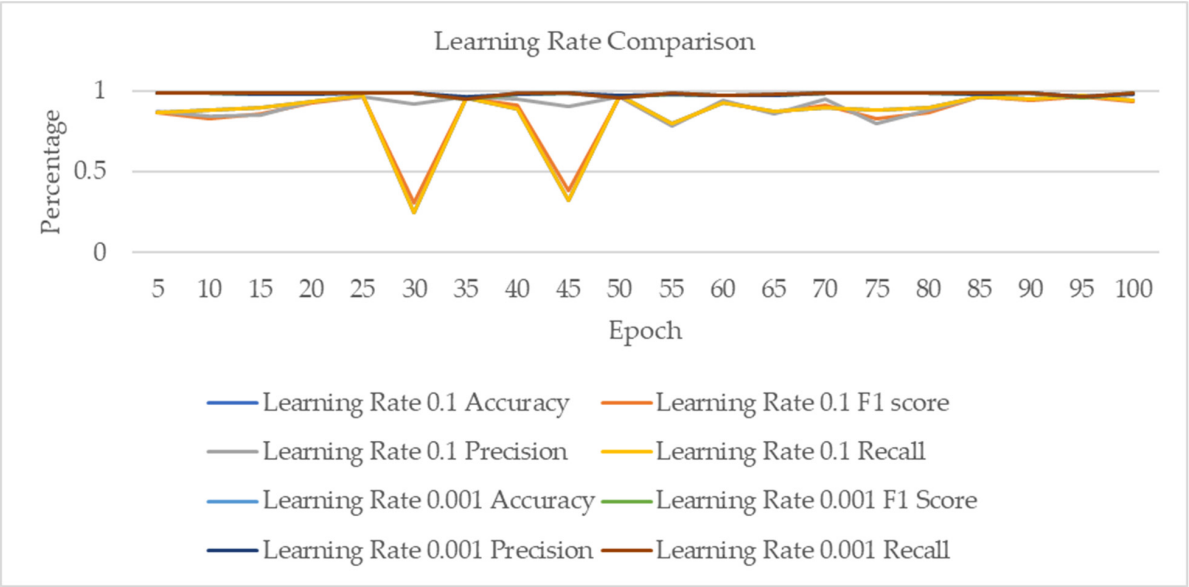

Figure 23. GATJK source classification learning rate comparison.

Varying Epochs

The training parameters were adjusted to the following: {hidden features: [8, 8]; layer type: "GATJK"; learning rate: 0.001; weight decay 0.0005; split ratio: 0.8; epochs: 5}. Reducing the epochs from 100 to 5 stops the oscillation around the optimum metrics, and this stops once the metrics have arrived or are close to the local maximum. This avoids overfitting the model to the data. This is reflected in the loss charts, which continually drop and flatten out, no longer rising again.

Figure 24 shows the results for the following source node classification metrics: accuracy, F1 score, precision, and recall. The initial epoch produces a high starting correct classification for all metrics. As epochs continue, the metrics flatten out to a plateau between 0.98 and 0.99.

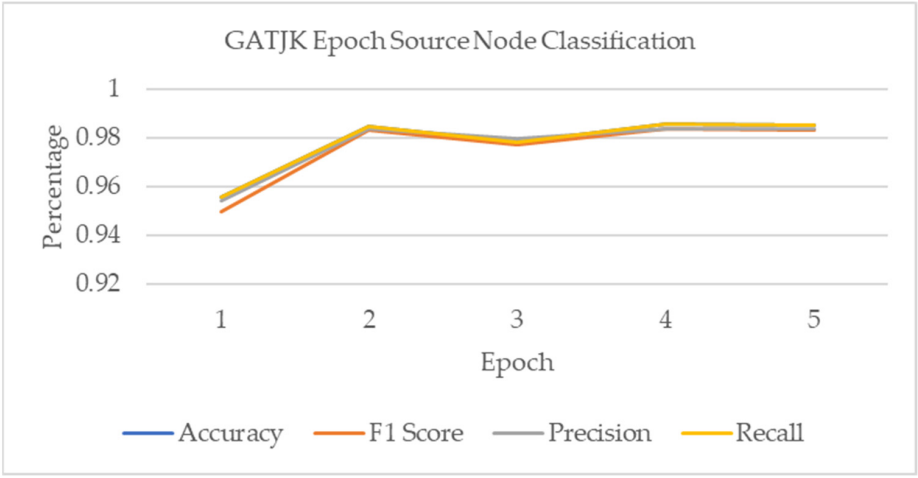

Figure 24. GATJK epoch source node classification metrics.

Figure 25 shows the loss from source node classification using GATJK. The loss from training data continually slopes down, approaching 0.05. Validation loss also follows the same trend and shows that five epochs is at, or close to, the optimal local minimum of loss.

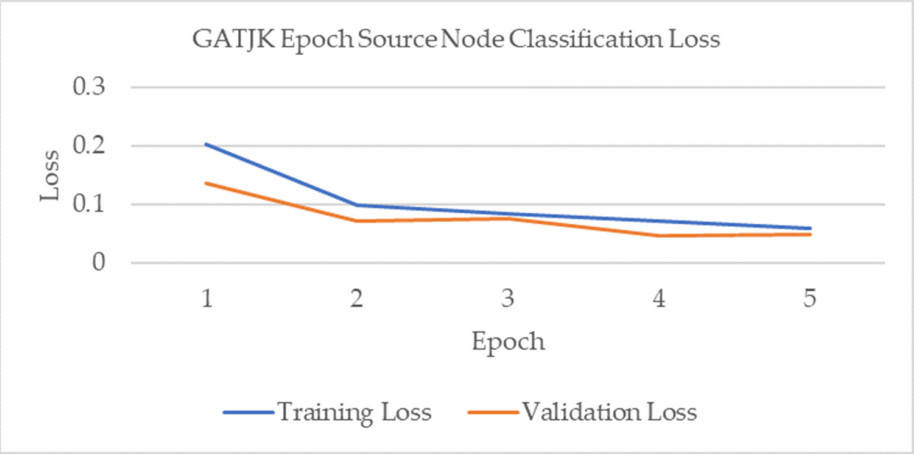

Figure 25. GATJK epoch source node classification loss.

Figure 26 shows accuracy, precision, recall, and F1 score for destination node classification. All metrics hit a local maximum at epoch 3, before slowly falling again. This contrasts with the source classification, which continued to improve beyond epoch 3. In terms of performance, destination node classification performed quite close to the source node classification, with both values ranging between 0.98 and 0.99 at their highest metric values.

Figure 27 shows the training and validation loss for destination node classification. Once again, loss is hitting a plateau at epoch 5, showing that at that point, we approach the local minimum. The loss is slightly higher compared to the source classification losses, but are comparably low, both being less than 0.1.

Table 14 summarizes the best results of accuracy, F1, precision, and recall for the source node classification as well as destination node classification using GATJK by adjusting learning rates and epochs for the respective nodes.

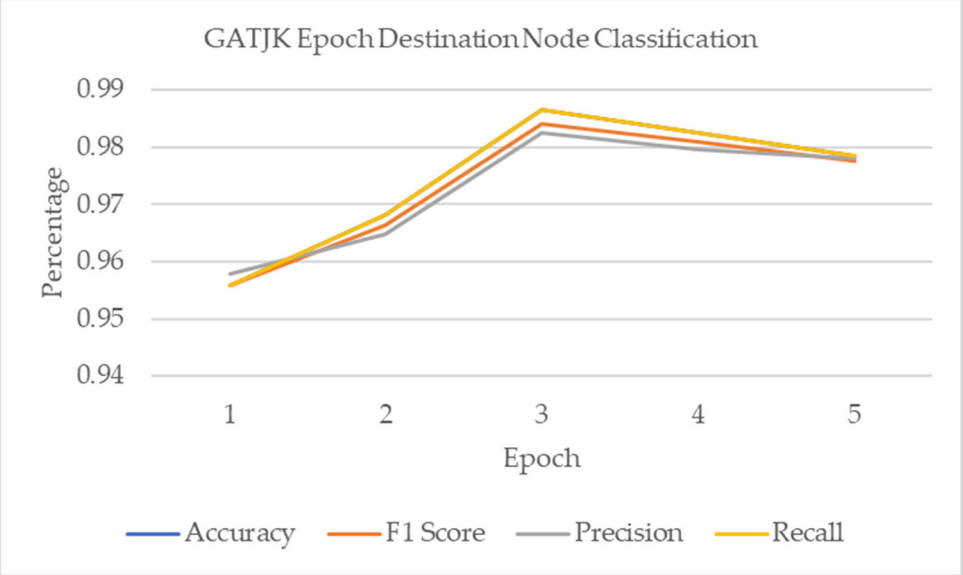

Figure 26. Destination epoch node classification metrics.

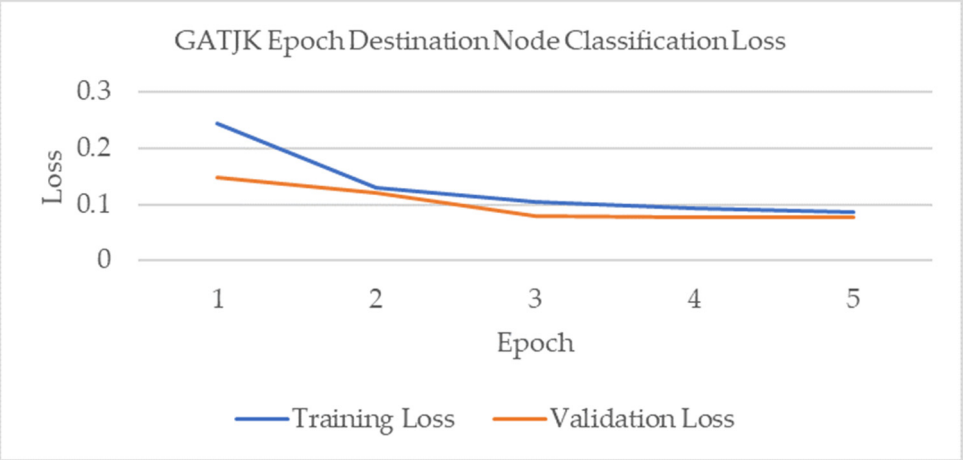

Figure 27. GATJK epoch destination node classification loss.

Table 14. Best GATJK results.

	Source Node Classification	Destination Node Classification
Accuracy	0.9851	0.9785
F1 Score	0.9834	0.9775
Precision	0.984	0.978
Recall	0.9851	0.9785

8.3.2. GraphSAGE

GraphSAGE was performed with the optimal parameters obtained from the GATJK model. The GraphSAGE model used the following parameters: {hidden features: [8, 8]; layer type: "SAGE"; learning rate: 0.0005; weight decay 0.0005; split ratio: 0.8; epochs: 10}. With similar parameters set for GraphSAGE, its performance was very comparable to GATJK, with metrics all lying within the 0.95 to 0.99 range for both source node classification and destination node classification, as shown in Figure 28. After epoch 6, the metrics stop trending higher and oscillates close to 0.96, with the highest metrics across accuracy, F1 score, precision, and recall falling between 0.96 and 0.98.

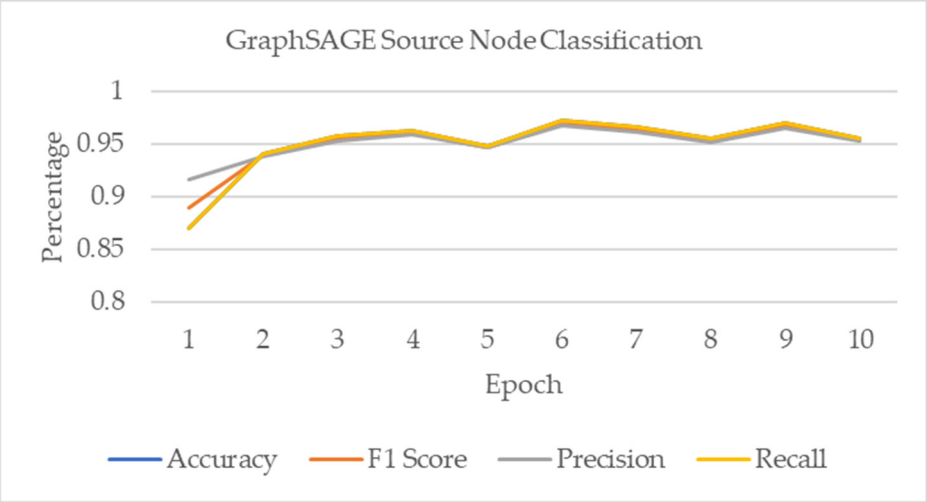

Figure 28. GraphSAGE source node classification metrics.

Figure 29 shows GraphSAGE source node classification loss. The loss plateaus starting from epoch 3 until training loss is almost flat at a value close to 0.1. The validation loss similarly plateaus, starting from epoch 3, and then peaks again at epoch 7. The validation loss is higher than the training loss, maintaining a value closer to 0.2.

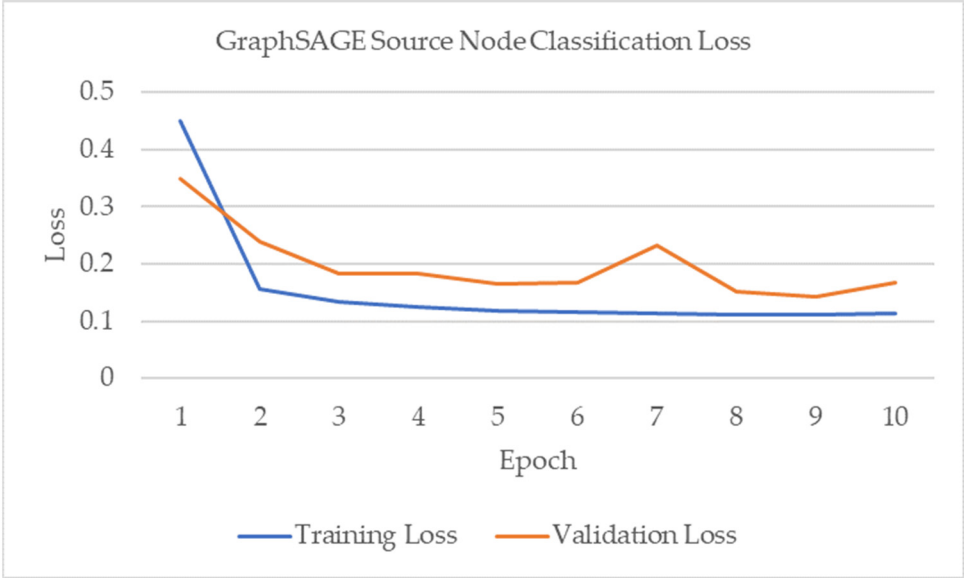

Figure 29. GraphSAGE source node classification loss.

Figure 30 shows the metrics for GraphSAGE destination node classification. The values peak at around epoch 7, nearing 0.99, before falling again. The peak performance appears to be slightly higher than GATJK, which peaked at around 0.985. The changes between epochs are more unpredictable as compared to source node classification.

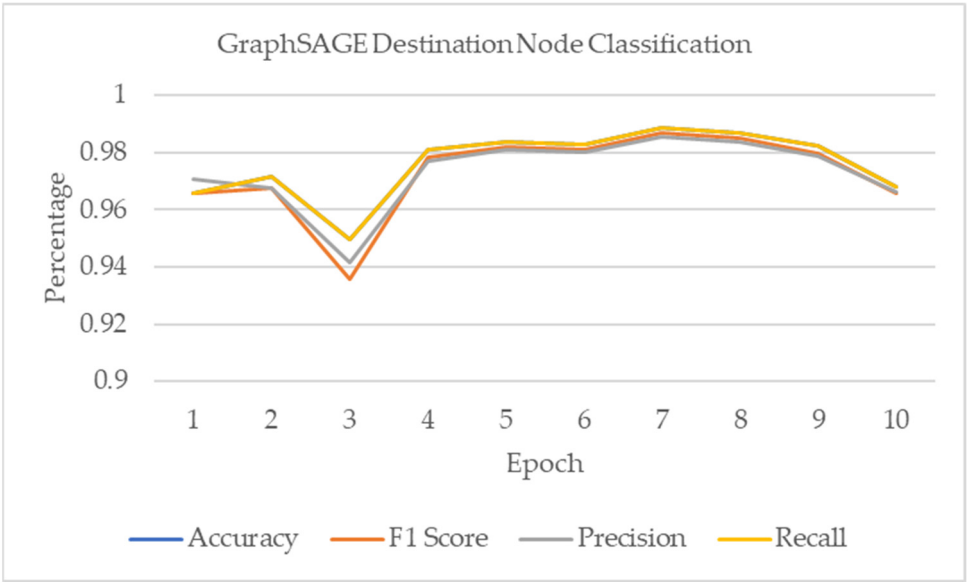

Figure 30. GraphSAGE destination node classification metrics.

Figure 31 shows GraphSAGE destination classification losses. The training losses continually trend downwards in the training loss line. The validation loss peaks at epoch 3 and then rises again at epoch 8. The overall loss is slightly higher in destination node classification loss compared to source node classification loss.

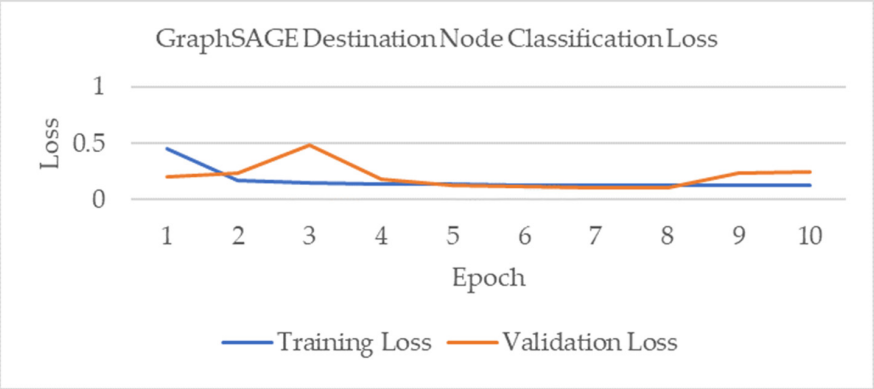

Figure 31. GraphSAGE destination node classification loss.

Table 15 summarizes the best results of accuracy, F1, precision, and recall for the source node classification as well as the destination node classification using GraphSAGE by adjusting learning rates and epochs for the respective nodes.

Table 15. Best GraphSAGE results.

	Source Node Classification	Destination Node Classification
Accuracy	0.9553	0.9681
F1 Score	0.9541	0.966
Precision	0.9536	0.9662
Recall	0.9553	0.9681

8.3.3. GATv2

GATv2 was also started with the optimal parameters obtained from the GATJK and GraphSAGE models. Using the same parameters as GATJK and GraphSAGE, the parameters were set to the following values: {hidden features: [8, 8]; layer type: "GATv2"; learning rate: 0.0005; weight decay 0.0005; split ratio: 0.8; epochs: 10}.

Figure 32 shows the resulting metrics for source node classification using GATv2. The results are very poor except for precision, which stays in the 0.8 to 0.9 range. Accuracy, F1 score, and recall stay below 0.6 after the first epoch. The results indicate the other GNNs are better solutions for node classification.

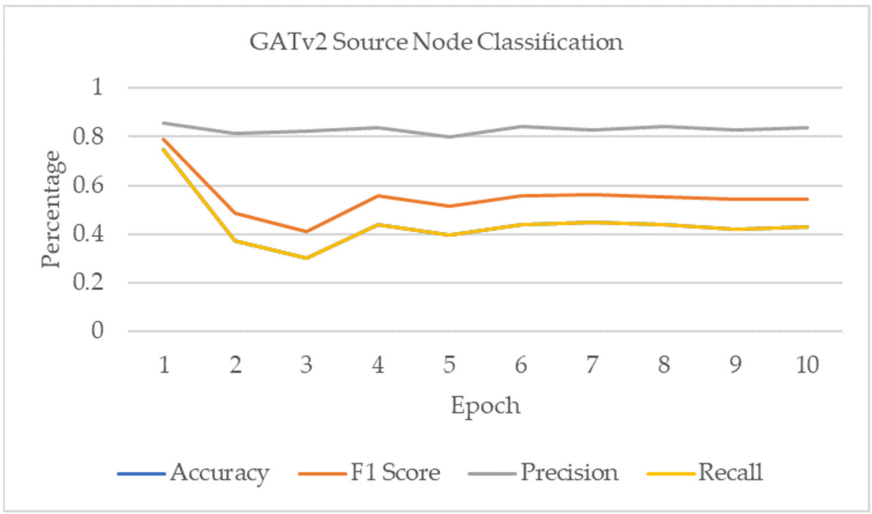

Figure 32. GATv2 source node classification metrics.

Figure 33 shows that although the losses trend downwards, the minimum losses are still much higher than the GATJK and GraphSAGE source node classification losses. The minimum training loss is still higher than 0.3.

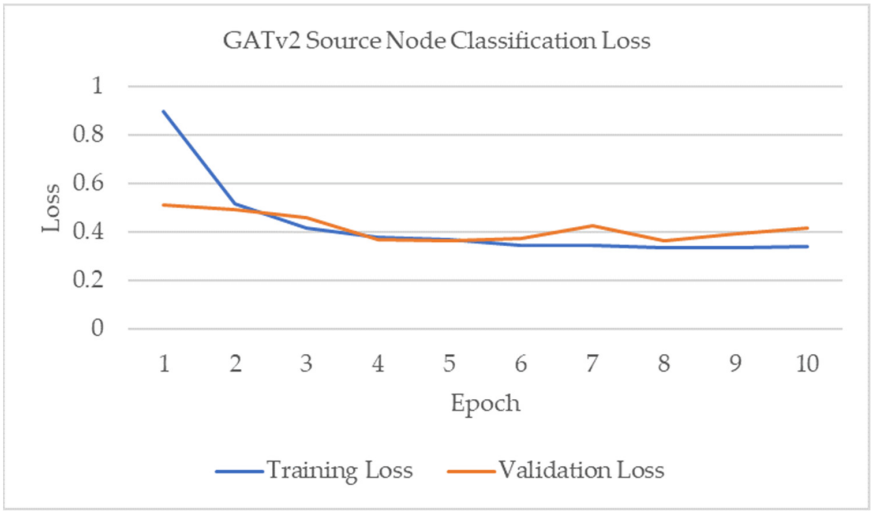

Figure 33. GATv2 source node classification loss.

Figure 34 shows that Gatv2 destination node classification performs relatively well compared to source node classification. The metrics stay close to 0.9 throughout, with no drastic changes over epochs.

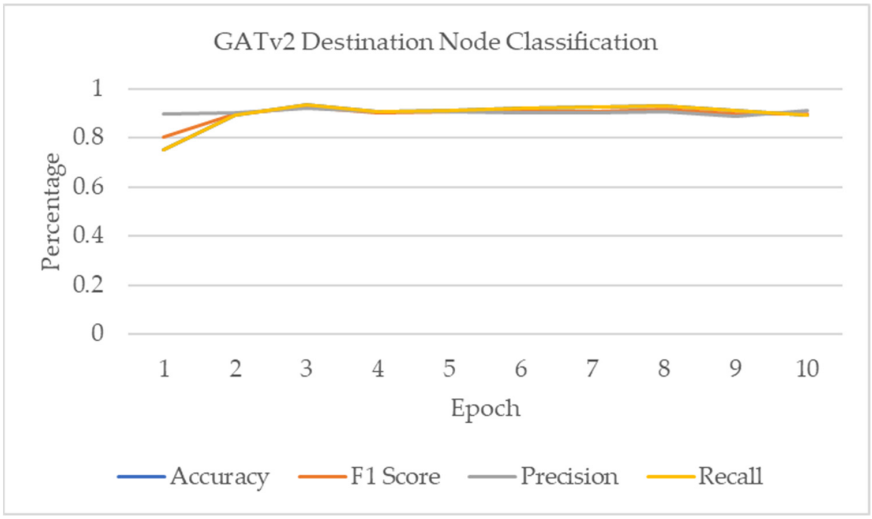

Figure 34. GATv2 destination node classification metrics.

Figure 35 shows that GATv2 destination node classification training loss and validation loss trend towards 0.2. Again, the destination node classification loss minimums are better than the source node classification losses.

Table 16 summarizes the best results of accuracy, F1, precision, and recall for the source node classification as well as the destination node classification using GATv2 by adjusting learning rates and epochs for the respective nodes.

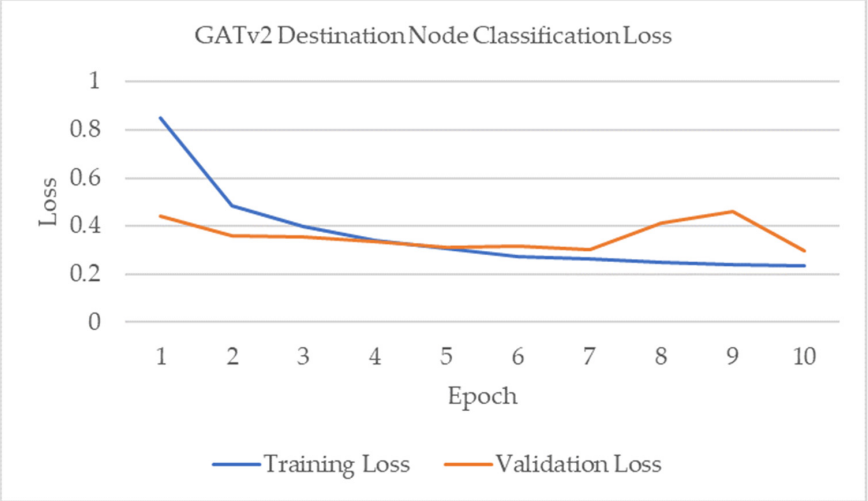

Figure 35. GATv2 destination node classification loss.

Table 16. Best GATv2 results.

	Source Node Classification	Destination Node Classification
Accuracy	0.4297	0.8928
F1 Score	0.5442	0.8992
Precision	0.8399	0.9113
Recall	0.4297	0.8928

8.4. Summary for Node Classification

The GNN models, for the purpose of classifying an IP address–port combination as a source or destination of an attack or benign connection tactic type based on graph-related features—in-degree, out-degree, and PageRank—proved to perform well depending on the selection of the algorithm or the layer type that was chosen. Except for GATv2, the other GNN models, GraphSAGE and GATJK, attained favorable results for both source and destination node classification with optimized models' lowest metrics being greater than 0.95. This supports the idea that the graph structure and graph-related features representing network connections can indicate the tactic of attacks being conducted as replicated in a controlled cyber environment.

8.5. Limitations of this Study

A limiting factor in our dataset is that the number of tactics for Reconnaissance far outweigh the other tactic types. That said, GNNs are a promising candidate for identifying bad actors in a network based on the results.

9. Conclusions

In this research, the UWF-ZeekData22 dataset, network logs generated by Zeek, a passive open-source network traffic analyzer, were viewed within a graph framework to explore and describe the graph structure of different MITRE ATT&CK tactics, and machine learning was performed using graph neural networks to classify tactics as source or destination nodes of an attack tactic.

Preprocessing to prepare the data for ingestion into Memgraph was conducted in Jupyter Notebooks. Unnecessary columns were removed, and additional node labels and addresses were edited to fit the format of a graph. The connections were set as edges with concatenated IP addresses and ports as the single address for each node. Both the nodes and edges were labeled for the attack tactic.

Memgraph generated a graph representation of UWF-ZeekData22, with which several graph properties were extracted, such as PageRank, degree, bridges, weakly connected

components, node and edge cardinality, and path length. These properties further described the graphs that were also visualized for different tactics, providing a different view.

Graph neural network models were generated for the UWF-ZeekData22 dataset to perform node classification; that is, to label a node as a source or destination node for the correct tactic under the MITRE ATT&CK framework. Through various means of testing and tweaking the models, favorable results were obtained with training parameters: {hidden features: [8, 8]; layer type: "GATJK"; learning rate: 0.001; weight decay 0.0005; split ratio: 0.8; epochs: 5}. Reducing the epochs to 5 stopped the oscillation around the optimum metrics and this prevented overfitting the model. So, of Memgraph's three GNN classifiers, GATJK gave the best results for both source node classification and destination node classification using only three graph features: in-degree, out-degree, and PageRank. For GATJK, all metrics, accuracy, F1-score, precision, and recall, produced above 98.3% and 97.7% for source and destination node classification, respectively. There was a significant improvement in loss over epochs with a lower learning rate of 0.001. The performance of GATv2 was the weakest of the three models.

Multi-classification had not been conducted on this set of data previously, but these results are better than some previous results obtained for binary classification using classical machine learning classifiers like SVM, naïve Bayes, and logistic regression in [30].

10. Future Works

The use of graph neural networks as an AI/ML intrusion detection system using live, real-time, "temporal" GNNs presents an exciting potential use-case that requires further research and exploration. As it currently stands, the Zeek logs provide a recording of events that have already happened and are useful for exploring past incidents. GNNs provide unique insight by framing connectivity from a graph perspective, and as research in GNNs is still ongoing, the use of GNNs as a solution for cybersecurity will continue to evolve and improve.

Author Contributions: Conceptualization, S.S.B., D.M., S.C.B., D.H.S. and F.M.; methodology, D.H.S. and F.M.; software, D.H.S. and F.M.; validation, S.S.B., D.M., S.C.B., D.H.S. and F.M.; formal analysis, D.H.S. and F.M.; investigation, D.H.S. and F.M.; resources, S.S.B., D.M. and S.C.B.; data curation, D.H.S. and F.M.; writing—D.H.S.; writing—review and editing, S.S.B., D.M., S.C.B. and D.H.S.; visualization, D.H.S. and F.M.; supervision, S.S.B., D.M. and S.C.B.; project administration, S.S.B., D.M. and S.C.B.; funding acquisition, S.S.B., D.M. and S.C.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Centers of Academic Excellence in Cybersecurity, 2021 NCAE-C-002: Cyber Research Innovation Grant Program, Grant Number: H98230-21-1-0170.

Data Availability Statement: The datasets are available at datasets.uwf.edu (accessed on 2 August 2023).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Howarth, J. The Ultimate List of Cyber Attack Stats. Exploding Topics. 2023. Available online: https://explodingtopics.com/blog/cybersecurity-stats (accessed on 13 June 2023).
2. Memgraph. "Memgraph Documentation". Memgraph Docs. Available online: https://memgraph.com/docs (accessed on 10 June 2023).
3. UWF-ZeekData22 Dataset. Available online: https://datasets.uwf.edu (accessed on 2 August 2023).
4. Bagui, S.S.; Mink, D.; Bagui, S.C.; Ghosh, T.; Plenkers, R.; McElroy, T.; Dulaney, S.; Shabanali, S. Introducing UWF-ZeekData22: A Comprehensive Network Traffic Dataset Based on the MITRE ATT&CK Framework. *Data* **2023**, *8*, 18. [CrossRef]
5. Neo4j, vs. Memgraph—How to Choose a Graph Database? Available online: https://memgraph.com/blog/neo4j-vs-memgraph (accessed on 20 October 2023).
6. Welcome to Neo4. Available online: https://neo4j.com/docs/getting-started/ (accessed on 20 October 2023).
7. Javorník, M.; Husák, M. Mission-centric decision support in cybersecurity via Bayesian Privilege Attack Graph. *Eng. Rep.* **2022**, *4*, e12538. [CrossRef]
8. Jacob, S.; Qiao, Y.; Ye, Y.; Lee, B. Anomalous distributed traffic: Detecting cyber security attacks amongst microservices using graph convolutional networks. *Comput. Secur.* **2022**, *118*, 102728. [CrossRef]

9. Wei, R.; Cai, L.; Zhao, L.; Yu, A.; Meng, D. DeepHunter: A graph neural network based approach for robust cyber threat hunting. Lecture Notes of the Institute for Computer Sciences. In *Security and Privacy in Communication Networks*; Part of the Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering; Springer: Berlin/Heidelberg, Germany, 2021; pp. 3–24. [CrossRef]
10. Haghshenas, S.H.; Hasnat, M.A.; Naeini, M. A temporal graph neural network for cyber attack detection and localization in smart grids. In Proceedings of the 2023 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT), Washington, DC, USA, 16–19 January 2023. [CrossRef]
11. Lee, M.-C.; Nguyen, H.T.; Berberidis, D.; Tseng, V.S.; Akoglu, L. GAWD: Graph anomaly detection in weighted directed graph databases. In Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, Virtual, 8–11 November 2021; ACM: New York, NY, USA, 2021.
12. Coupette, C.; Vreeken, J. Graph Similarity Description: How Are These Graphs Similar? In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, Virtual, 14–18 August 2021; ACM: New York, NY, USA, 2021.
13. Schindler, T. Anomaly Detection in Log Data Using Graph Databases and Machine Learning to Defend Advanced Persistent Threats. Available online: https://dl.gi.de/handle/20.500.12116/4016 (accessed on 23 January 2024).
14. Bai, Y.; Ding, H.; Bian, S.; Chen, T.; Sun, Y.; Wang, W. SimGNN: A Neural Network Approach to Fast Graph Similarity Computation. In Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, Melbourne, VIC, Australia, 11–15 February 2019; ACM: New York, NY, USA, 2019.
15. What Is the MITRE ATT&CK Framework? | Get the 101 Guide. Trellix. 2022. Available online: https://www.trellix.com/en-us/security-awareness/cybersecurity/what-is-mitre-attack-framework.html (accessed on 1 October 2023).
16. MITRE ATT&CK. Reconnaissance, Tactic TA0043—Enterprise | MITRE ATT&CK®. 2023. Available online: https://attack.mitre.org/tactics/TA0043/ (accessed on 2 August 2023).
17. MITRE ATT&CK. Discovery, Tactic TA0007—Enterprise | MITRE ATT&CK®. 2023. Available online: https://attack.mitre.org/tactics/TA0007/ (accessed on 2 August 2023).
18. MITRE ATT&CK. Credential Access, Tactic TA0006—Enterprise | MITRE ATT&CK®. 2023. Available online: https://attack.mitre.org/tactics/TA0006/ (accessed on 2 August 2023).
19. Gleich, D.F. PageRank Beyond the Web. *SIAM Rev.* **2015**, *57*, 321–363. [CrossRef]
20. Memgraph. Pagerank. Memgraph Docs. Available online: https://memgraph.com/docs/mage/query-modules/cpp/pagerank (accessed on 6 July 2023).
21. "Math Insight". The Degree Distribution of a Network—Math Insight. Available online: https://mathinsight.org/degree_distribution (accessed on 11 June 2023).
22. "Degree Distribution". Wikipedia. 17 February 2023. Available online: https://en.wikipedia.org/wiki/Degree_distribution (accessed on 12 August 2023).
23. Geeks for Geeks. Find Weakly Connected Components in a Directed Graph. Available online: https://www.geeksforgeeks.org/find-weakly-connected-components-in-a-directed-graph/# (accessed on 20 August 2023).
24. Memgraph. Node Classification. Memgraph Docs. Available online: https://memgraph.com/docs/mage/query-modules/python/node-classification-with-gnn (accessed on 20 August 2023).
25. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; Bengio, Y. Graph Attention Networks. *arXiv* **2018**, arXiv:1710.10903v3.
26. Florian. Understanding Graph Attention Networks [Video File]. Youtube. Available online: https://www.youtube.com/watch?v=A-yKQamf2Fc&ab_channel=DeepFindr (accessed on 20 August 2023).
27. Torch_geometric.nn.models.GAT. models.GAT. (n.d.). Available online: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.models.GAT.html (accessed on 20 August 2023).
28. Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.I.; Jegelka, S. Representation Learning on Graphs with Jumping Knowledge Networks. *arXiv* **2017**, arXiv:1806.03536v2.
29. Hamilton, W.L.; Ying, R.; Leskovec, J. Inductive Representation Learning on Large Graphs. NIPS. 31st Conference. 2018. Available online: https://snap.stanford.edu/graphsage/#:~:text=GraphSAGE%20is%20a%20framework%20for,Code (accessed on 2 August 2023).
30. Bagui, S.; Mink, D.; Bagui, S.; Ghosh, T.; McElroy, T.; Paredes, E.; Khasnavis, N.; Plenkers, R. Detecting Reconnaissance and Discovery Tactics from the MITRE ATT&CK Framework in Zeek Conn Logs Using Spark's Machine Learning in the Big Data Framework. *Sensors* **2022**, *22*, 7999. [CrossRef] [PubMed]