DSA4236 Project Write Up: Cybersecurity Attack Detection

April 17, 2025

Group Composition:

1. Ang Zhiyong, A0233445J, e0725445@u.nus.edu
2. Chan Jin Xiang Filbert, A0245018L, e0883965@u.nus.edu
3. Emma Lim Xiaoen, A0239300R, e0773898@u.nus.edu
4. Jax Lee Le Sheng, A0236637X, e0746341@u.nus.edu
5. See Wee Shen Rachel, A0238731A, e0773329@u.nus.edu

Contents

1 Abstract

Cyber security attacks remain one of the most disruptive threats to modern digital infrastructures, impacting sectors such as finance, e-commerce, cloud computing, and cybersecurity. These attacks not only cause service outages and financial losses but can also serve as a distraction for more serious threats like data breaches or ransomware. This project aims to develop an accurate and adaptive network intrusion detection system (NIDS) to detect various attacks using data science techniques.

We utilize the CIC-IDS2017 and CIC-IDS2018 datasets, focusing on time windows containing various network attacks and patters. Our approach begins with classical machine learning models—Logistic Regression and AdaBoost—augmented with graph-based metrics derived from packet communication patterns. These features notably increased recall and enabled the models to identify nearly all malicious traffic. However, this improvement came at the cost of higher false positive rates, which may hinder operational feasibility.

To address this, we implemented a Graph Attention Network (GAT), which explicitly learns from the relational structure of network traffic. Although GAT had a slightly lower F1 score and higher false negatives, it significantly reduced false positives, offering better practical trade-offs. Its interpretability and scalability make it a robust long-term solution.

Finally, we outline how our system can detect broader fraud patterns—such as account takeovers and lateral movement—highlighting the flexibility and industry relevance of graph-based detection methods in modern cybersecurity.

2 Introduction

We selected the dataset CIC-IDS2017/2018, that contains network traffic data with labeled instances of benign activity and Distributed Denial-of-Service (DDoS) attacks. For this study, we focus specifically on the days where DDoS attacks occurred, analyzing packet-level information to distinguish between normal and malicious traffic. This dataset is particularly relevant to industries that rely on secure and stable network infrastructures, such as financial services, cloud computing, e-commerce, and cybersecurity firms.

In this study, we aim to formulate attack detection methods using data science techniques, leveraging machine learning models to identify DDoS traffic patterns. We will compare the effectiveness of different approaches, evaluating their performance and implementation feasibility to enhance cybersecurity defenses against DDoS threats.

3 Data

The raw data used in this project originates from the CSE-CIC-IDS2017 and CSE-CIC-IDS2018 datasets—two benchmark datasets designed to simulate realistic benign and malicious network traffic. Each dataset is split by day, where each day features mostly benign traffic along with specific types of attacks (e.g., DDoS, Botnet, SQL injection). Although the attacks were simulated in a controlled environment, they are meant to closely mimic real-world cyber threats, making them useful for training and evaluating intrusion detection systems.

3.1 Preprocessing Steps

We extracted a 2-hour window from each day to balance attack and benign traffic and keep dataset size managable. Column names were then standardized to enable seamless merging.

3.2 Dataset Size

The final dataset consists of:

- **2017 Dataset:**
  - DDoS (attack + benign)
  - DoS (attack only)
  - Botnet (attack only)
- **2018 Dataset:**
  - DDoS (attack + benign)

After processing, each dataset contains 84 columns (or features). The label distribution is as follows:

Year/Count	Benign	Attack	Total
2017	382665	97718	480383
2018	1343290	575394	1918684

3.3 Key Features

Each row represents aggregated network packet statistics between two IP addresses during a given flow. Key features include:

- **Flow ID, Source IP, Destination IP:**
  - indicate communication endpoints and direction.

- **Packet Characteristics:**
  - such as total packets, packet size, inter-arrival time (IAT), etc.
- **Labels:**
  - traffic is labeled using a binary classification scheme (Benign = 0, Attack = 1).

3.4 Data Collection and Potential Biases

The data was collected in a lab environment using common applications and scripted attacks. While this offers clean labels and structured data, it introduces certain biases—for example, attack patterns may appear more distinct than in noisy, real-world environments. Additionally, label imbalance and attack-type separation influence model generalizability. Furthermore, all simulated attacks were carried out by the same source IP, which may increase bias due to little variability in IPs available for proper analysis.

3.5 CICFlowMeter Output and Graph Abstraction

We used CICFlowMeter to convert raw PCAPs from CIC-IDS2017 and DDoS2018 into bidirectional flows identified by the 5-tuple (src IP, dst IP, src port, dst port, protocol). Each flow is annotated with over 80 statistics—packet/byte counts, duration, inter-arrival times, and TCP flags.

Although CICFlowMeter does not directly compute graph-theoretical metrics, it provides a structured, high-dimensional representation of communication behaviors that can be readily transformed into attributed graphs. In the framework used in our chosen model (GAT), each unique IP address is modeled as a **graph node**, and each flow represented by a **directed edge** connecting a source and destination IP. The flow features form the edge attributes, while node features are derived from aggregated flow data and additional behavioral metrics such as entropy and communication frequency. This abstraction allows for effective relational learning via Graph Neural Networks (GNNs) while remaining computationally tractable and scalable.

4 Methods

4.1 Exploratory Data Analysis

4.1.1 Traffic Flow Analysis

A DDoS attack overwhelms a target server with excessive traffic, preventing legitimate users from accessing the system. Based on this, we hypothesize that network traffic experiences a significant surge during a DDoS attack, which may be reflected in a sharp increase in flow counts from IPs that are participating in the attack over a short period.

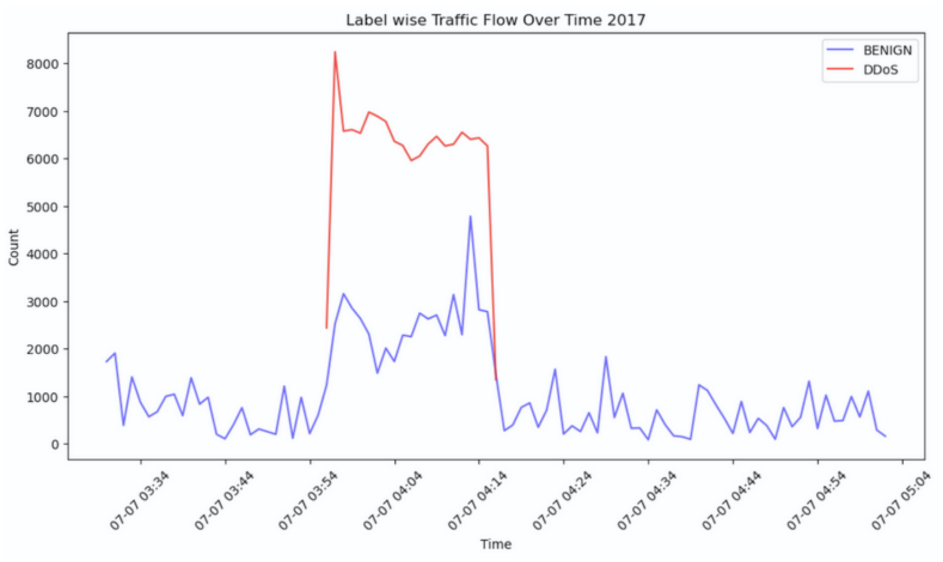


Figure 1: Traffic Flow over Time 2017

The dataset analysis largely supports our hypothesis. As evident in Figures 1 and 2, in both 2017 and 2018, we observed noticeable spikes in traffic flow during the time of the DDoS attacks. While the magnitude of these spikes varied between the two years, the idea is the same, which is to overload the target server with requests. The red plot lines signify the attack requests, while the purple lines are the normal traffic. With the introduction of the attack traffic, the cumulative traffic flow is significantly bumped up during the attack duration. This overwhelms the server with a sudden flood of requests, denying service to legitimate users.

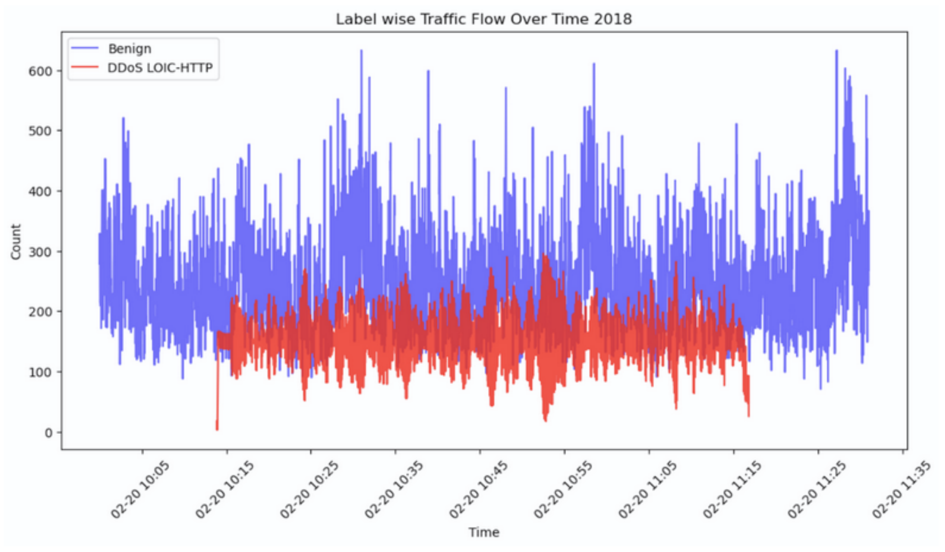


Figure 2: Traffic Flow over Time 2018

4.1.2 Packet Characteristic Analysis

We further hypothesize that DDoS traffic would exhibit irregularities in packet quantity and volume, that deviates from typical network behavior. Legitimate network behaviour should have larger and more variability due to the different types of requests from different IPs sending packets to the server. As such, we expect features related to packet length and packet count to play a significant role in decision-making.

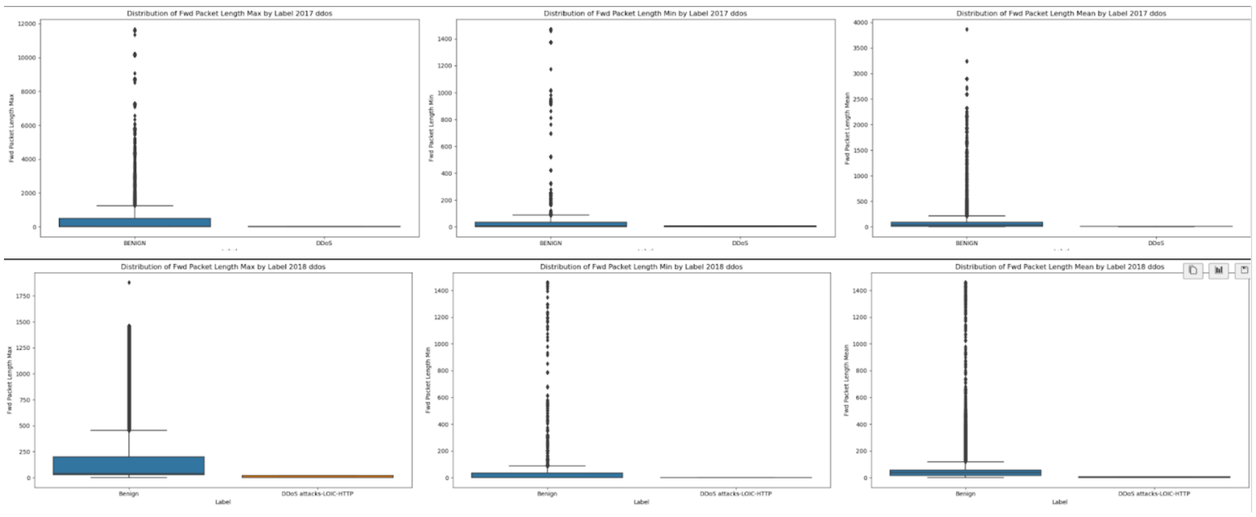
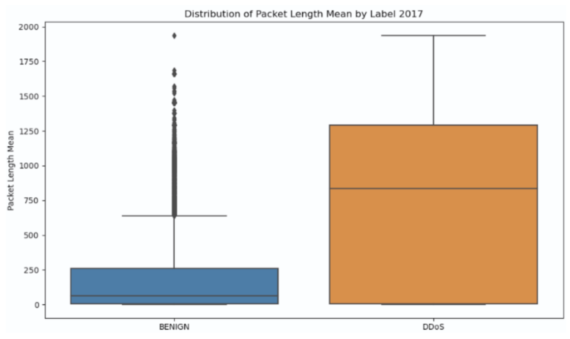


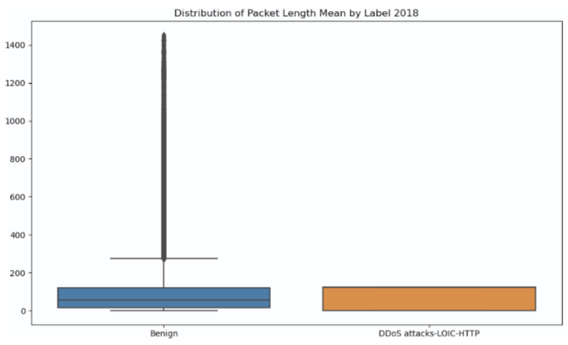
Figure 3: Forward Packets Analysis for 2017 and 2018

The data analysis supports our hypothesis, as seen in Figure 3, the distribution of forward packet in DDoS traffic for both years are starkly different from that of legitimate traffic. Forward packet lengths from benign network activity are spread over a very large range, compared to that of DDoS attacks which have relatively fixed lengths. The absence of variability confirms our hypothesis in differentiating between benign and attacks, based on packet characteristics.

However, other features presented some differences. For instance, as seen in Figure 4, the 2017 mean overall packet length for DDoS traffic was significantly higher than that of benign traffic. In contrast, this difference was much less pronounced in the 2018 dataset, where both classes displayed almost similar distributions for packet length.



(a) 2017



(b) 2018

Figure 4: Mean Packet Length 2017 & 2018

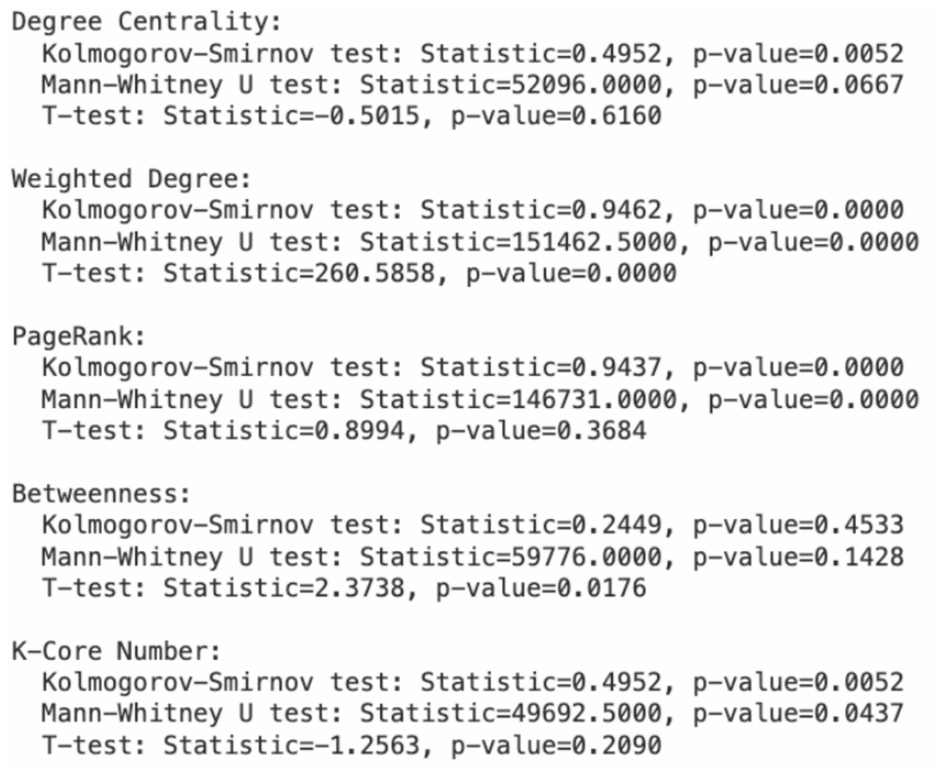
This discrepancy could be due to variations in the DDoS attack strategies employed over the two years. One possible interpretation is that the 2017 attacks relied on overwhelming the server with a high-volume packets, while the 2018 attacks may have focused more on initiating a large number of flows rather than maximizing individual packet volume. Interestingly, while the number of DDoS flows in 2018 did not exceed that of benign flows - as shown in the network traffic graph - it is possible that the attack targeted an already congested network. A small number of malicious flows with relatively large packet volumes might have triggered increased user activity inadvertently triggering the network overload.

4.1.3 Usefulness of Graph Structures

Lastly, We hypothesized that the graph structure of benign traffic should exhibit different statistical properties compared to malicious DDoS traffic. We expect benign traffic to be more evenly distributed communication across nodes. DDoS traffic should have a highly centralized structure, where many attack nodes (botnets) flood a small number of target nodes (victims). This could manifest in high edge density toward certain nodes, leading to an anomalous spike in connections.

We constructed graph structures using the 2018 dataset. In these graphs, source and destination IP addresses were treated as nodes, with directed edges created from source to destination IPs. Edge weights were defined as the number of flows initiated from the source to the corresponding destination. We then computed the following graph-based metrics: Degree Centrality, Weighted Degree Centrality, PageRank, Betweenness Centrality, K-Core Number.

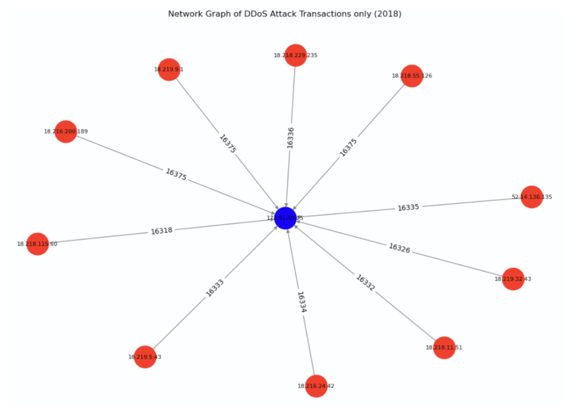
To evaluate whether these metrics exhibit significant differences between attack and benign labels, we applied a variety of statistical tests, namely the Kolmogorov-Smirnov test, Mann-Whitney U test, and T-test. A low p-value (less than 0.05) indicates significant distributional or central tendency differences between the two groups.



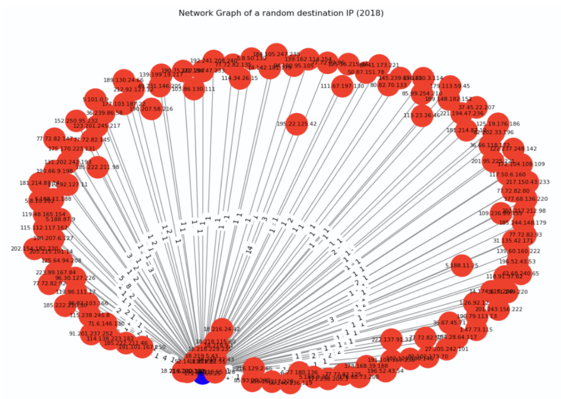
```
Degree Centrality:
  Kolmogorov-Smirnov test: Statistic=0.4952, p-value=0.0052
  Mann-Whitney U test: Statistic=52096.0000, p-value=0.0667
  T-test: Statistic=-0.5015, p-value=0.6160

Weighted Degree:
  Kolmogorov-Smirnov test: Statistic=0.9462, p-value=0.0000
  Mann-Whitney U test: Statistic=151462.5000, p-value=0.0000
  T-test: Statistic=260.5858, p-value=0.0000

PageRank:
  Kolmogorov-Smirnov test: Statistic=0.9437, p-value=0.0000
  Mann-Whitney U test: Statistic=146731.0000, p-value=0.0000
  T-test: Statistic=0.8994, p-value=0.3684

Betweenness:
  Kolmogorov-Smirnov test: Statistic=0.2449, p-value=0.4533
  Mann-Whitney U test: Statistic=59776.0000, p-value=0.1428
  T-test: Statistic=2.3738, p-value=0.0176

K-Core Number:
  Kolmogorov-Smirnov test: Statistic=0.4952, p-value=0.0052
  Mann-Whitney U test: Statistic=49692.5000, p-value=0.0437
  T-test: Statistic=-1.2563, p-value=0.2090
```

Figure 5: Statistical tests of graph structures

The results show that Degree Centrality, PageRank, and K-core Number are effective in distinguishing between attack and benign nodes, while Betweenness provides little discrimination. These findings suggest that node and edge-level metrics can be valuable for decision-making.



(a)



(b)

Figure 6: Network Graph Visualizations

From Figure 6a, we observe a cluster of 10 malicious IPs aggressively targeting a single IP with a high volume of flows. In contrast in 6b, the benign network activity is more evenly distributed across a larger number of nodes, with significantly lower flow counts per connection. This stark difference highlights the structural anomaly introduced by the DDoS attack. Thus, we shifted our focus toward leveraging graph network structure of the datasets to improve model performance.

4.2 Designing Robust Train-Test Strategies

Our data analysis reveals significant differences between the 2017 and 2018 datasets. For instance, not only do the feature distributions vary in their trends, but the label distributions also differ considerably. Moreover, in 2017 only 2 distinct IP addresses participated in the attack. While in 2018, multiple IP addresses targeted a single IP address. Although both attacks aimed to flood servers, the strategies and patterns observed differed. These discrepancies raise an important challenge: how can we effectively use these datasets for real-time attack detection?

We found that DoS and Botnet traffic patterns shared similarities with DDoS behaviors, particularly in terms of flow dynamics. Due to this, we supplemented our 2017 training dataset with DoS and Botnet data to enhance the model's ability to generalize to more attacks. We excluded 2018 DoS and Botnet data from 2018's dataset because they lacked critical features—such as source and destination IP addresses—that are essential for constructing meaningful network graph structures.

To prevent data leakage, we avoided a random 80/20 split since many attack-labeled rows stem from the same event. This could result in flows from the same attack appearing in both train and test sets, inflating performance. Instead, we used 2 temporal methods of training: first is training on the enriched 2017 dataset and testing on the 2018 DDoS data. Secondly is by using a rolling window to determine if there is an attack within a short timeframe. This is particularly useful as we aim to mimic real world settings and tracking, where it is crucial to identify attacks that happen in short periods of time. These ensure clean separation and provide a more realistic evaluation of real-world effectiveness.

4.3 Model Selection

As a baseline model, we selected Logistic Regression (LR) due to its reputation as a strong linear baseline. Its high interpretability makes it particularly valuable in cybersecurity contexts, where understanding the rationale behind decisions is crucial.

As a comparator model, we implemented AdaBoost, which adaptively increases focus on misclassified points. This quality makes it well-suited for detecting rare attack instances. Additionally, by combining multiple weak learners, AdaBoost becomes robust to noise — a property we anticipate to be important in this dataset. During our exploratory data analysis (EDA), PCA results suggested the dataset is not linearly separable, which supports the use of tree-based models like AdaBoost over linear models such as Logistic Regression.

To further enhance model performance, we incorporated graph-based metrics into our features. Statistical analysis indicated that these metrics could significantly improve predictive power, so we included them in our challenger models. In addition, we experimented with Graph Convolutional Networks (GCNs) and Graph Attention Networks (GATs), which are designed to leverage the temporal network structure inherent in cybersecurity datasets.

4.4 Baseline and Graph-Metric Models

We began by removing features with no variance, as they provided no discriminatory power for classification. We then examined multicollinearity and filtered out highly correlated features to reduce noise and prevent overfitting.

Beyond data cleaning, we engineered additional features using directional graph-based metrics to capture structural network behavior. Following the simple graph structure described in Section 4.1.3, graph metric values were computed by averaging the metrics of the corresponding source and destination IPs for each row in the dataset. Weights are defined by the number of unique directional flows initiated between each source and destination IP.

All features were standardized to ensure uniformity across the dataset and optimize model performance. Models were optimized using GridSearch with 5-fold cross-validation to fine-tune hyperparameters prior to final testing and evaluation.

Feature	Description	Calculation Method	Weighted
avr_weighted_degree	Measures how many connections a node has, weighted by flow count.	Average of the weighted degree centrality of the source and destination IPs.	Yes
avr_page_rank	Indicates the relative importance of a node based on its network connections.	Average of the PageRank values of the source and destination IPs.	Yes
avr_degree_centrality	Represents the fraction of nodes a node is connected to in the network.	Average of the degree centrality scores of the source and destination IPs.	No
avr_k_core	Reflects how deep a node is embedded within the core of the graph.	Average of the k-core values of the source and destination IPs.	No

Table 1: Feature descriptions, calculation methods, and weighting for graph-metric features

4.5 Graph Attention Network (GAT) for Flow-based Intrusion Detection

4.5.1 Motivation and Design Rationale

Initial experiments with ensemble models such as AdaBoost using decision trees yielded moderate classification performance. However, these models lacked the capacity to model relational structures between network entities and temporal dependencies in flow behavior. Specifically, they treat data points independently, failing to capture the interactions between IP addresses and the evolving nature of coordinated attacks.

To address these limitations, we implemented a Graph Attention Network (GAT) [3] tailored for flow-based intrusion detection. The GAT architecture enables message passing between nodes (IP addresses) with learnable attention weights, while integrating both node-level and edge-level features. This approach transforms intrusion detection into a graph-based, temporally segmented, edge classification task.

4.5.2 Data Preprocessing and Temporal Structuring

The CIC-IDS2017 dataset was cleaned to remove invalid timestamps and infinite values. Missing values were imputed using mean imputation, and all numerical features were standardized using z-score normalization.

To emulate real-time detection, flow records were partitioned into 1-minute bins:

$$\tau_i = \lfloor t_i \rfloor_{\text{minute}}$$

Each bin τ formed a graph $\mathcal{G}_\tau = (\mathcal{V}_\tau, \mathcal{E}_\tau)$ where:

- $\mathcal{V}_\tau$ is the set of all unique IP addresses appearing as source or destination within the minute.
- $\mathcal{E}_\tau$ is the set of directed edges representing flows (s_i, d_i) where $s_i, d_i \in \mathcal{V}_\tau$.

This fixed-size temporal binning balances computational efficiency with temporal causality, simulating how production intrusion detection systems operate incrementally in real-time.

4.5.3 Node Feature Engineering

Each node (IP address) $v \in \mathcal{V}_\tau$ was assigned a feature vector $h_v^{(0)} \in \mathbb{R}^{d+3}$, consisting of:

1. **Aggregated flow features:**
$$\mathbf{f}_v = \sum_{i \in \mathcal{F}_v} \mathbf{x}_i$$
where $\mathbf{x}_i$ is the standardized feature vector for flow i, and $\mathcal{F}_v$ is the set of flows with v as the source.

2. **Source/Destination counts:** $c_{\text{src}}(v)$ and $c_{\text{dst}}(v)$, the number of times v appears as source or destination respectively.

3. **Destination entropy:**
$$H(v) = -\sum_{j \in \mathcal{D}_v} p_j \log p_j, \quad \text{where } p_j = \frac{f_j}{\sum_k f_k}$$
and f_j is the count of flows from v to destination j.

These features encode both statistical and structural behavior, crucial for identifying anomalous traffic patterns such as DDoS or scanning.

4.5.4 Model Architecture: EdgeGAT

The model architecture consists of two components:

(a) Node Embedding via GAT: Node embeddings are learned using stacked GAT layers [3]:‘

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(l)} W^{(l)} h_u^{(l)}\right)$$

where $\alpha_{vu}^{(l)}$ are attention coefficients computed as:

$$\alpha_{vu}^{(l)} = \frac{\exp\left(\text{LeakyReLU}(\mathbf{a}^\top [W h_v \,\|\, W h_u])\right)}{\sum_{k \in \mathcal{N}(v)} \exp\left(\text{LeakyReLU}(\mathbf{a}^\top [W h_v \,\|\, W h_k])\right)}$$

(b) Edge Classification MLP: For each edge (flow) (u, v) with flow features f_{uv}, the final edge representation is:

$$z_{uv} = \text{MLP}([h_u^{(2)} \parallel h_v^{(2)} \parallel f_{uv}])$$

which is passed through a 2-layer multilayer perceptron with ReLU activations and softmax output for binary classification.

4.5.5 Training Strategy and Optimization

The model was trained using the cross-entropy loss:

$$\mathcal{L} = -\sum_{(u,v)} y_{uv} \log \hat{p}_{uv}^{(y_{uv})}$$

where $y_{uv} \in \{0, 1\}$ is the true class, and $\hat{p}_{uv}$ are the softmax-normalized predicted probabilities.

Training configuration:

- Optimizer: Adam with learning rate 5×10^{-5}
- Epochs: 200
- Skipping class-imbalanced graphs (e.g., all-benign)
- Float64 precision enforced for numerical stability
- ‘tqdm‘ used for epoch and batch-level progress tracking

4.5.6 Evaluation on Out-of-Sample Test Set

The model was evaluated on a separate dataset (DDOS2018), preprocessed using the same scaler and imputer. For each minute graph:

- Node and edge features were recomputed.
- Predictions were obtained from the trained model.
- Class labels were derived as: $\hat{c}_{uv} = \arg\max(\hat{p}_{uv})$

4.5.7 Temporal Processing Comparison

Table 2 compares temporal data strategies considered:

Strategy	Description	Trade-offs
Full Dataset	Processes all records at once	Unrealistic for live inference
Sliding Window	Shifts fixed window over time	Accurate but computationally expensive
Fixed Chunk (1-min)	Uses discrete temporal bins	Realistic and scalable

Table 2: Comparison of temporal strategies for intrusion detection

4.6 Extension to Spatio-Temporal Graphs with A3TGCN2

While the Graph Attention Network (GAT) architecture achieved promising results by modeling the spatial dependencies between IP addresses within each one-minute snapshot, it inherently processes graphs as static entities. However, in realistic threat scenarios, attacks often unfold over time. For instance, port scans may escalate into brute-force attacks, and beaconing behavior may precede large-scale data exfiltration or DDoS activity. These temporal patterns are not captured by models that treat each graph independently.

To address this, we implemented a spatio-temporal graph neural network—**A3TGCN2**—which incorporates both spatial attention and temporal convolution across a sequence of graph snapshots. In this model, each minute-level graph G_τ is treated as a timestep in a dynamic sequence. Node features are stacked across time and passed through A3TGCN2 layers to learn time-evolving embeddings. These embeddings are then fused for each edge (u, v) by concatenating the source and destination node representations along with flow-level features, which are passed into a multilayer perceptron (MLP) for classification.

```
class A3TGCN2EdgeClassifier(nn.Module):
    def init(self, inchannels, outchannels=2, batchsize=1):
        super().init()
        self.recurrent = A3TGCN2(inchannels=inchannels, outchannels=64,
                                 periods=1, batchsize=batchsize)
        self.edgemlp = nn.Sequential(
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, outchannels)
        )

    def forward(self, x, edgeindex, edgeweight, edgesrc, edgedst):
        x = x.to(device)
        edgeindex = edgeindex.to(device)
        edgesrc = edgesrc.to(device)
        edgedst = edgedst.to(device)
```

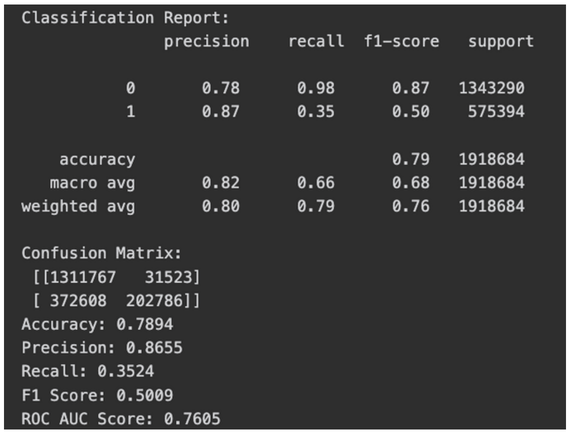
```
if edgeweight is not None:
    edgeweight = edgeweight.to(device)
h = self.recurrent(x, edgeindex, edgeweight)[0]
hsrc = h[edgesrc]
hdst = h[edgedst]
edgeinput = torch.cat([hsrc, hdst], dim=1)
return self.edgemlp(edgeinput)
```

4.7 Performance Metrics

For this problem, we decided to maximize the F1 score because it balances both precision and recall, which are crucial in our context. Specifically, we believe it is most important to identify attack instances accurately, so we prioritize having a high true positive (TP) rate and a low false positive (FP) rate. Additionally, recall was closely considered during our analysis, as we wanted to ensure that as many attack instances as possible were correctly identified, even at the cost of precision in some cases.

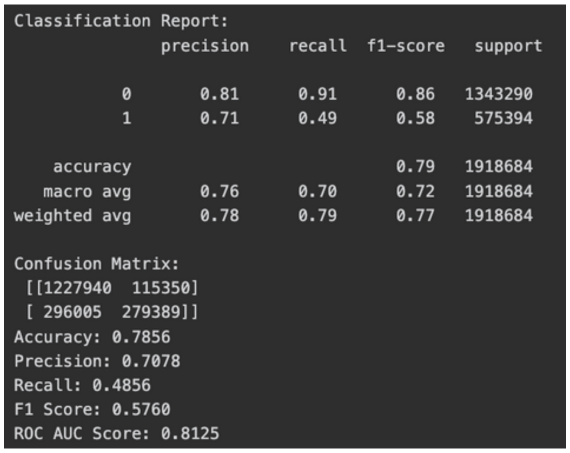
5 Results

5.1 Model Results



```
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.98      0.87   1343290
           1       0.87      0.35      0.50    575394

    accuracy                           0.79   1918684
   macro avg       0.82      0.66      0.68   1918684
weighted avg       0.80      0.79      0.76   1918684

Confusion Matrix:
 [[1311767   31523]
 [ 372608  202786]]
Accuracy: 0.7894
Precision: 0.8655
Recall: 0.3524
F1 Score: 0.5009
ROC AUC Score: 0.7605
```

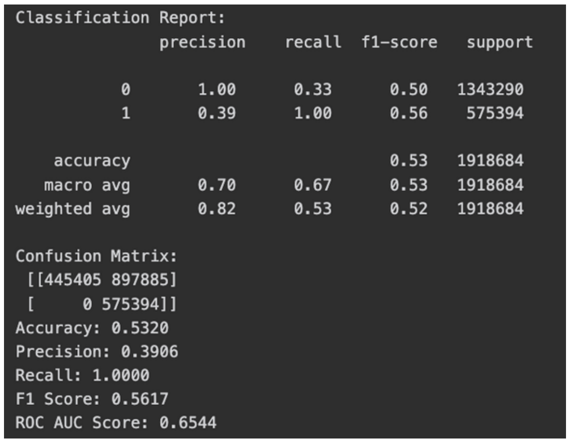
(a) Baseline LR



```
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.91      0.86   1343290
           1       0.71      0.49      0.58    575394

    accuracy                           0.79   1918684
   macro avg       0.76      0.70      0.72   1918684
weighted avg       0.78      0.79      0.77   1918684

Confusion Matrix:
 [[1227940  115350]
 [ 296005  279389]]
Accuracy: 0.7856
Precision: 0.7078
Recall: 0.4856
F1 Score: 0.5760
ROC AUC Score: 0.8125
```

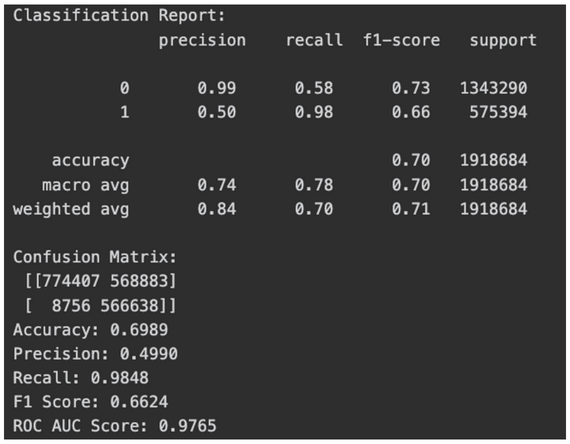
(b) ADABoost

Figure 7



```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.33      0.50   1343290
           1       0.39      1.00      0.56    575394

    accuracy                           0.53   1918684
   macro avg       0.70      0.67      0.53   1918684
weighted avg       0.82      0.53      0.52   1918684

Confusion Matrix:
 [[445405 897885]
 [     0 575394]]
Accuracy: 0.5320
Precision: 0.3906
Recall: 1.0000
F1 Score: 0.5617
ROC AUC Score: 0.6544
```

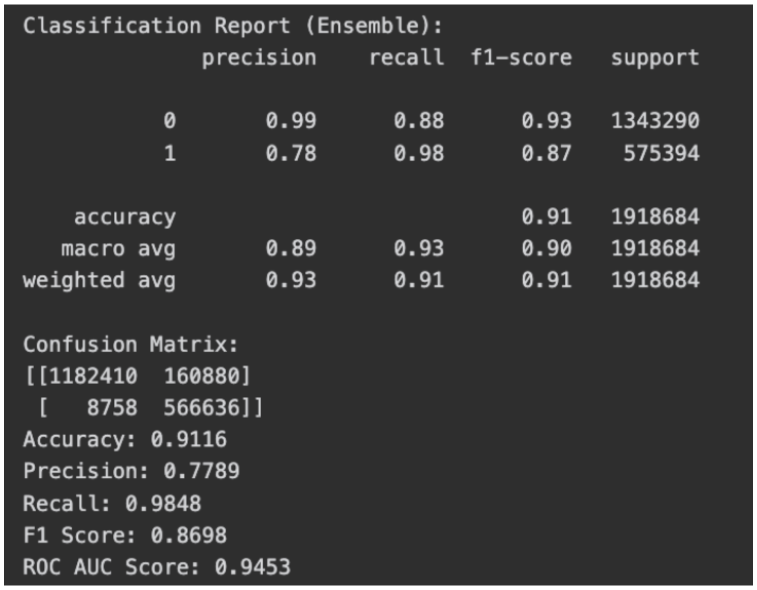
(a) Baseline LR with graph metrics



```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.58      0.73   1343290
           1       0.50      0.98      0.66    575394

    accuracy                           0.70   1918684
   macro avg       0.74      0.78      0.70   1918684
weighted avg       0.84      0.70      0.71   1918684

Confusion Matrix:
 [[774407 568883]
 [  8756 566638]]
Accuracy: 0.6989
Precision: 0.4990
Recall: 0.9848
F1 Score: 0.6624
ROC AUC Score: 0.9765
```

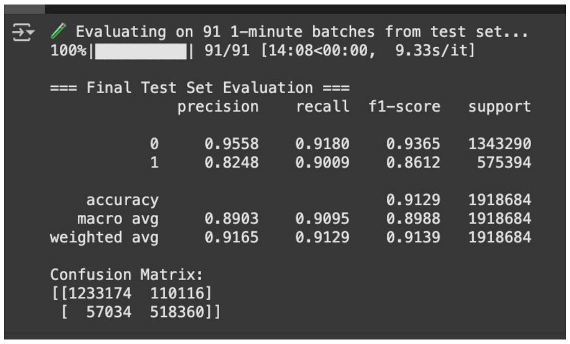
(b) ADABoost with graph metrics

Figure 8



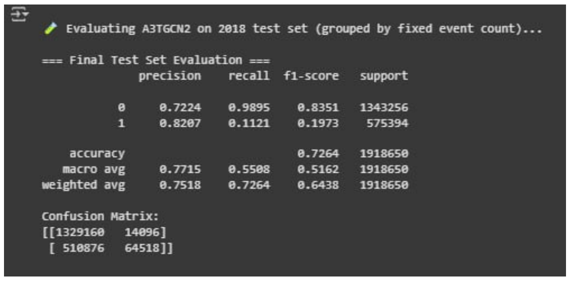
```
Classification Report (Ensemble):
              precision    recall  f1-score   support

           0       0.99      0.88      0.93   1343290
           1       0.78      0.98      0.87    575394

    accuracy                           0.91   1918684
   macro avg       0.89      0.93      0.90   1918684
weighted avg       0.93      0.91      0.91   1918684

Confusion Matrix:
[[1182410  160880]
 [   8758  566636]]
Accuracy: 0.9116
Precision: 0.7789
Recall: 0.9848
F1 Score: 0.8698
ROC AUC Score: 0.9453
```

Figure 9: Ensemble Model



```
🧪 Evaluating on 91 1-minute batches from test set...
100%|██████████| 91/91 [14:08<00:00,  9.33s/it]

=== Final Test Set Evaluation ===
              precision    recall  f1-score   support

           0     0.9558    0.9180    0.9365   1343290
           1     0.8248    0.9009    0.8612    575394

    accuracy                         0.9129   1918684
   macro avg     0.8903    0.9095    0.8988   1918684
weighted avg     0.9165    0.9129    0.9139   1918684

Confusion Matrix:
[[1233174  110116]
 [  57034  518360]]
```

(a) GAT



```
🧪 Evaluating A3TGCN2 on 2018 test set (grouped by fixed event count)...

=== Final Test Set Evaluation ===
              precision    recall  f1-score   support

           0     0.7224    0.9895    0.8351   1343256
           1     0.8207    0.1121    0.1973    575394

    accuracy                         0.7264   1918650
   macro avg     0.7715    0.5508    0.5162   1918650
weighted avg     0.7518    0.7264    0.6438   1918650

Confusion Matrix:
[[1329160   14096]
 [ 510876   64518]]
```

(b) A3TGCN2

Figure 10

6 Discussion on Results

6.1 Analysis of Impact of Graph-Based Features on Model Performance

The introduction of graph-based features led to notable improvements in model performance for both Logistic Regression and AdaBoost classifiers. Specifically, the F1 scores increased across the board, with the most significant boost observed in the recall metric. In fact, recall scores approached 100%, indicating that the models were able to correctly identify nearly all attack instances in the test set. However, this improvement came at the cost of a substantial increase in false positives, which lowered overall precision.

This behavior initially raised concerns about overfitting. However, since the training and testing datasets are temporally separated, data leakage is unlikely. To better understand this phenomenon, we examined feature importances. As seen in Figure 11, average_weighted_degree consistently ranked among the top features.

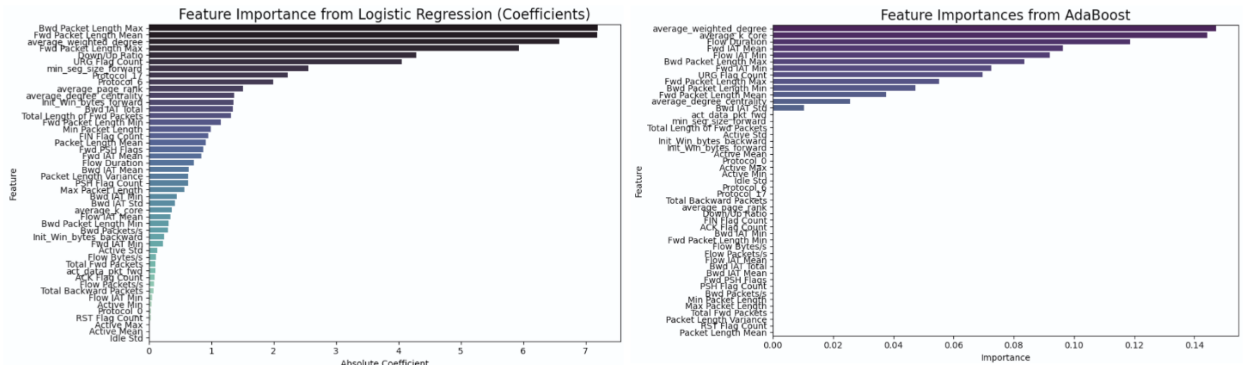


Figure 11: Feature Importance of Models with data including Graph Metrics

From EDA, we observed that malicious nodes tend to have greater number of flows initiated, which reflects in higher edge weights. This makes graph-based features, particularly weighted ones, highly predictive of attacks. However, they could have introduced noise when interpreting benign activity, which leads to increased false alarms. In short: graph features drive high recall (catching nearly all attacks), but at the cost of lower precision (more benign traffic misclassified as malicious).

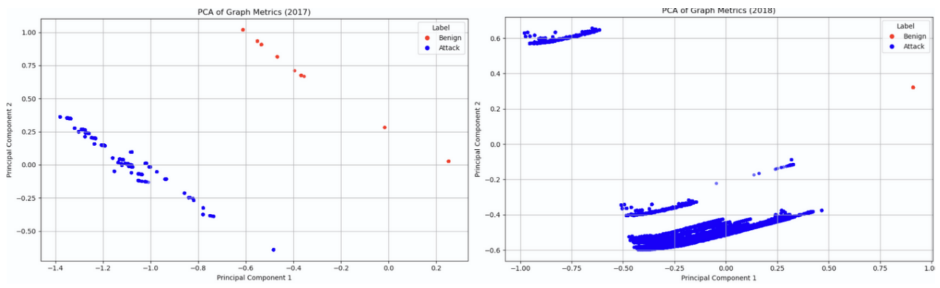


Figure 12: Sampled PCA on Graph Metrics

Further insights came from PCA visualizations of the graph features. In the 2017 training data, the classes were linearly separable, meaning models like Logistic Regression could easily distinguish attacks from benign flows. However, the 2018 test data showed much less separability, suggesting a shift in the underlying data distribution. This explains why the Logistic Regression model, trained on clear 2017 patterns, achieved perfect recall but also triggered a high number of false positives—it was overgeneralizing based on outdated decision boundaries.

This highlights a key challenge in real-world detection tasks: patterns learned in the past may not generalize perfectly over time, especially for benign behaviors. Despite this, the strong recall indicates that the graph features are still valuable for identifying malicious behavior, even if they do not fully distinguish benign flows as cleanly. Despite its trade-offs, the graph features prove valuable for detecting attacks. To balance precision and recall, we ensembled all four models. This ensemble approach yielded a more balanced performance and resulted in the highest F1 score overall.

6.2 Analysis on Observed Performance Degradation of A3TGCN2

Despite the architectural sophistication of A3TGCN2, our experiments showed a noticeable drop in classification accuracy compared to the GAT-based model. We hypothesize several contributing factors to this degradation:

- **Noisy or inconsistent temporal alignment:** CICFlowMeter generates flow records based on connection-level summaries rather than precise packet timestamps. As a result, the temporal continuity between successive graph snapshots may be weak or partially misaligned, limiting the model's ability to learn meaningful temporal transitions.
- **Overfitting due to limited temporal context:** The model was trained with a short window of past graph snapshots (e.g., 5 minutes). This limited history might not have captured sufficient temporal diversity, causing the recurrent layers to overfit to short-term patterns while failing to generalize across varying attack behaviors.
- **High variance in graph structure over time:** The number of active IP addresses and flow connections varies significantly between time bins. This inconsistency may destabilize temporal learning, especially when embeddings are propagated across drastically different graph topologies.
- **Increased model complexity:** A3TGCN2 introduces additional learnable parameters and architectural depth. Given the modest size of the dataset and the sparsity of malicious labels in certain intervals, the model may suffer from undertraining or convergence instability, especially without a large-scale pretraining or regularization strategy.

Although A3TGCN2 did not yield better performance in this instance, it introduced a temporal learning capability that static models like GAT cannot provide. This experiment suggests that with more precise temporal annotation (e.g., packet-level streaming) or sequence-aware preprocessing, temporal GNNs could become a powerful tool for real-time intrusion detection.

6.3 Final Model Selection

However, as mentioned in earlier sections, the traditional supervised models treat the data points independently and do not capture relationships between nodes of a network, and are likely to be unable to be effective on dynamic attack patterns. We posit that the Graph Attention Network (GAT) represents a more principled and robust approach to network intrusion detection in the long term due to the following reasons:

6.3.1 Graph Structure Awareness

GAT is inherently designed to capture the relational structure of network traffic, allowing it to model interactions between packets, flows, or hosts more effectively than traditional models that assume independent samples. This ability to exploit graph topology is particularly valuable for identifying coordinated or stealthy attacks that manifest in complex traffic patterns.

6.3.2 Advantages of Sliding Windows and Temporal Features in GAT

Many real-world attack patterns evolve over time. Therefore, incorporating a sliding window approach and temporal features into the Graph Attention Network (GAT) significantly improves detection accuracy and adaptability. DDoS attacks, brute-force attempts, and lateral movements do not occur instantaneously—they build up over time. A sliding window approach segments traffic into time-based snapshots, enabling the model to detect temporal shifts in behavior. The temporal features also help resolve the data leakage issue that previous models faced, by considering only the data within the timeframe.

6.3.3 Interpretability and Scalability

The attention mechanism in GAT provides transparent insight into which nodes (e.g., packets or connections) are most influential during message passing. This interpretability can be leveraged for forensic analysis and security auditing, offering advantages over black-box ensemble methods. This also allows the model to identify cliques which may have a high influence over the entire network, therefore may require monitoring or regulative measures.

GAT also provides a modular framework that can be extended with temporal dynamics, hierarchical graph structures, and richer node/edge features. This makes it future-proof for evolving threat landscapes, whereas ensemble models may require extensive feature engineering to adapt.

6.3.4 Operational Practicality: False Negatives vs False Positives

While GAT exhibits slightly more false negatives (FN), this is acceptable in practice given that the overall F1 score remains competitive. Network infrastructure is typically designed to tolerate some level of malicious traffic, especially under controlled volumes.

On the other hand, false positives (FP) can disrupt legitimate users and overwhelm security response teams. GAT's lower false positive rate makes it more practical for real-world deployment. In scenarios where false positives do occur, mitigation techniques such as CAPTCHA challenges can be applied to validate user intent with minimal disruption.

Despite achieving slightly lower F1 scores, GAT provides operational advantages—such as fewer false positives and better adaptability to real-world network topologies. This makes it especially appealing for practical deployment, where maintaining user experience and minimizing alert fatigue is crucial.

6.4 Summary and Implications

The proposed GAT model outperformed traditional classifiers in terms of generalization and robustness, owing to its ability to:

- Prioritize structurally relevant neighbors via attention
- Integrate edge-specific flow features with node behavior
- Simulate real-time detection via temporal graph segmentation

This work demonstrates the potential of graph-based learning for cybersecurity applications where relationships—not just attributes—carry predictive signal.

7 Hypothetical Deployment Pipeline

Our proposed deployment pipeline integrates both traditional ensemble models and deep learning on graphs to balance efficiency with detection depth. The pipeline operates as follows:

7.1 Raw Traffic Parsing & Aggregation

Network traffic is continuously ingested and parsed using tools like [1] or [2], and aggregated into fixed-size time windows (e.g., per minute).

7.2 Graph Construction & Feature Engineering

Within each window, network flows are grouped into graph structures where nodes represent IP addresses and edges represent flow interactions. Graph metrics (e.g., weighted degree, betweenness centrality) and flow-level features are computed and stored.

7.3 Fast Filtering via Ensemble Model

The ensemble model (Logistic Regression + AdaBoost with graph features) is applied first to perform lightweight, high-recall filtering. This allows for rapid detection of potentially suspicious flows while minimizing computational cost.

7.4 GNN-Enhanced Deep Inspection

- **Graph Attention Network (GAT)** is applied to a filtered high-risk subset, offering deeper relational analysis of suspicious patterns.

- **A3TGCN2** can be scheduled to run periodically (e.g., every 5 minutes) or on-demand during peak hours or known threat windows. It processes temporally-segmented graphs, capturing long-term and coordinated attack behaviors in real-time.

7.5 Trigger-Based Scheduling

Model inference frequency can be dynamically adjusted based on:

- Traffic volume thresholds (e.g., spikes in flow count)
- Time-of-day schedules (e.g., higher scrutiny during off-hours)
- Security incident triggers (e.g., alerts from external threat intelligence feeds)

7.5.1 Feedback Loop for Continuous Learning

Predictions flagged by the system are reviewed by analysts via a SIEM platform (e.g., `Splunk` or ELK `Stack`). Confirmed true/false positives are fed back into the model training pipeline, enabling continual learning and performance improvement over time.

7.6 Result Integration & Visualization

All detections are pushed to a centralized dashboard for:

- Real-time alerts
- Historical pattern tracking
- Analyst drill-down and exportable reports

7.7 Detecting Fraud Patterns Outside DDoS

With its graph-aware architecture, GAT can be adapted to detect various types of fraud patterns beyond DDoS, including:

- **Coordinated Account Takeovers**
  - Shared infrastructure (e.g., botnets or subnets) often leads to "fan-out" patterns—one IP connecting to many accounts. GAT can identify such anomalies through attention on shared neighbors.
- **Credential Stuffing or Brute-force Attacks**
  - Repeated login attempts from the same or few IPs create redundant edge patterns. GAT can detect these temporal and structural irregularities across time windows.

8 Conclusion

This project highlights the value of incorporating graph-based features and deep learning on graphs into the Network Intrusion Detection (NID) pipeline. Traditional models like Logistic Regression and AdaBoost saw significant recall gains from graph metrics such as weighted degree and PageRank, leveraging patterns where attack nodes initiate more flows. However, these models struggle with early detection and evolving threats due to their assumption of independent data points. Hence, we explored Graph Neural Networks (GNNs). GAT improved detection by learning attention-based relationships between IPs, making it effective for identifying anomalies in relational traffic. We further experimented with A3TGCN2, a spatio-temporal GNN capable of modeling long-term coordinated behaviors. While conceptually promising, A3TGCN2 underperformed on our dataset. As such, we propose GAT as our best model due to its practicality and potential adaptability to other attack patterns.

9 Appendix

9.1 Preprocessed Data

Variable Name	Data Type	Description
Flow ID	object	Unique identifier for each network flow
Source IP	object	IP address of the sender
Source Port	int64	Port number of the sender
Destination IP	object	IP address of the receiver
Destination Port	int64	Port number of the receiver
Protocol	int64	Protocol used (e.g., TCP, UDP)
Timestamp	object	Time when the flow started
Flow Duration	int64	Total duration of the flow (in microseconds)
Total Fwd Packets	int64	Number of packets sent in the forward direction
Total Backward Packets	int64	Number of packets sent in the backward direction
Total Length of Fwd Packets	int64	Total size of forward packets
Total Length of Bwd Packets	float64	Total size of backward packets
Fwd Packet Length Max/Min/Mean/Std	mixed	Statistics of forward packet sizes
Bwd Packet Length Max/Min/Mean/Std	mixed	Statistics of backward packet sizes
Flow Bytes/s	float64	Flow byte rate per second
Flow Packets/s	float64	Flow packet rate per second
Flow IAT Mean/Std/Max/Min	float64	Inter-arrival time stats between packets
Fwd IAT Total/Mean/Std/Max/Min	float64	Inter-arrival time stats in the forward direction
Bwd IAT Total/Mean/Std/Max/Min	float64	Inter-arrival time stats in the backward direction
Fwd/Bwd PSH Flags	int64	PSH flag occurrences in forward/backward direction

Table 3: Preprocessed Data Dictionary

Variable Name	**Data Type**	**Description**
Fwd/Bwd URG Flags	int64	URG flag occurrences in forward/backward direction
Fwd Header Length	int64	Header size of forward packets
Bwd Header Length	int64	Header size of backward packets
Fwd Packets/s	float64	Forward packet rate
Bwd Packets/s	float64	Backward packet rate
Min/Max Packet Length	int64	Packet size extremes in the flow
Packet Length Mean/Std/Variance	float64	Statistics of all packet lengths
FIN/SYN/RST/PSH Flag Count	int64	TCP flag counters
ACK/URG/CWE/ECE Flag Count	int64	TCP flag counters
Down/Up Ratio	int64	Ratio of download to upload size
Average Packet Size	float64	Average size of packets
Avg Fwd/Bwd Segment Size	float64	Average segment sizes
Fwd/Bwd Avg Bytes/Bulk	int64	Average bytes per bulk transfer in each direction
Fwd/Bwd Avg Packets/Bulk	int64	Average packets per bulk transfer
Fwd/Bwd Avg Bulk Rate	int64	Bulk data rate
Subflow Fwd/Bwd Packets	int64	Packet count in subflow
Subflow Fwd/Bwd Bytes	int64	Byte count in subflow
Init_Win_bytes_fwd/bwd	int64	Initial TCP window size
act_data_pkt_fwd	int64	Count of actual data packets in forward direction
min_seg_size_forward	int64	Minimum segment size in forward direction
Active Mean/Std/Max/Min	float64	Active period stats
Idle Mean/Std/Max/Min	float64	Idle period stats
Label	int64	Class label (Benign = 0, Attack = 1)

Table 4: Preprocessed Data Dictionary

9.2 Processed and Feature Engineered Data

The 2 tables below describes the features used to train and test the baseline model Logistic Regression and ADAboost.

Column Name	**Data Type**	**Description**
Flow Duration	int64	Total duration of the flow in microseconds
Total Fwd Packets	int64	Total number of packets in the forward direction
Total Backward Packets	int64	Total number of packets in the backward direction
Total Length of Fwd Packets	float64	Total length of packets in the forward direction
Fwd Packet Length Max	float64	Maximum length of a forward packet
Fwd Packet Length Min	float64	Minimum length of a forward packet
Fwd Packet Length Mean	float64	Mean length of forward packets
Bwd Packet Length Max	float64	Maximum length of a backward packet
Bwd Packet Length Min	float64	Minimum length of a backward packet
Flow Bytes/s	float64	Number of bytes per second in the flow
Flow Packets/s	float64	Number of packets per second in the flow
Flow IAT Mean	float64	Mean inter-arrival time for packets in the flow
Flow IAT Min	float64	Minimum inter-arrival time for packets in the flow
Fwd IAT Mean	float64	Mean inter-arrival time in forward direction
Fwd IAT Min	float64	Minimum inter-arrival time in forward direction
Bwd IAT Total	float64	Total inter-arrival time in backward direction
Bwd IAT Mean	float64	Mean inter-arrival time in backward direction
Bwd IAT Std	float64	Standard deviation of inter-arrival time (backward)
Bwd IAT Min	float64	Minimum inter-arrival time in backward direction
Fwd PSH Flags	int64	Count of PSH flags in forward direction
Bwd Packets/s	float64	Packets per second in backward direction
Min Packet Length	float64	Minimum packet length in the flow
Max Packet Length	float64	Maximum packet length in the flow

Table 5: Processed and Feature Engineered Data Dictionary

Column Name	**Data Type**	**Description**
Packet Length Mean	float64	Mean length of packets in the flow
Packet Length Variance	float64	Variance of packet lengths in the flow
FIN Flag Count	int64	Number of FIN flags seen in packets
RST Flag Count	int64	Number of RST flags seen in packets
PSH Flag Count	int64	Number of PSH flags seen in packets
ACK Flag Count	int64	Number of ACK flags seen in packets
URG Flag Count	int64	Number of URG flags seen in packets
Down/Up Ratio	float64	Ratio of bytes from destination to source
Init_Win_bytes_forward	int64	Initial window size in forward direction
Init_Win_bytes_backward	int64	Initial window size in backward direction
act_data_pkt_fwd	int64	Number of active data packets in forward direction
min_seg_size_forward	int64	Minimum segment size in forward direction
Active Mean	float64	Mean time a flow is active before going idle
Active Std	float64	Standard deviation of active times
Active Max	float64	Maximum active time
Active Min	float64	Minimum active time
Idle Std	float64	Standard deviation of idle time between flows
Protocol_0	bool	One-hot encoded protocol = 0
Protocol_6	bool	One-hot encoded protocol = 6
Protocol_17	bool	One-hot encoded protocol = 17
Label	category	Class label (Benign = 0, Attack = 1)

Table 6: Processed and Feature Engineered Data Dictionary

9.3 Graph Metric Features

The table below describes graph metric features engineered to supplement the existing dataset.

Variable Name	**Data Type**	**Description**
average_weighted_degree	float64	Average weighted degree centrality of the source and destination IPs.
average_page_rank	float64	Average PageRank values of the source and destination IPs.
average_degree_centrality	float64	Average degree centrality of the source and destination IPs.
average_k_core	float64	Average of the k-core values of the source and destination IPs.

Table 7: Data Dictionary for Graph Metric Features

References

[1] insanecyber. Introduction to zeek: Open-source threat hunting and network traffic analysis, April 2025.

[2] MarkViglione. Suricata: What is it and how can we use it, 2022.

[3] Arantxa Casanova Adriana Romero Yoshua Bengio Pietro Lio Petar Velickovic, Guillem Cucurull. In *Graph Attention Networks*, 2018.