Detecting Network Intrusion

Ty Misiorek (zem4by), Sanket Doddabendigere (kfr9mc), Shirley Li (fky9xf), Shawn Thomas (yzy2bt), Ansh Pathapadu (tqc7wn)

1 Problem Statement

In the modern world, nearly everything can be done online. Whether it be handling finances, work meetings, shopping, or accessing educational material, a large portion of human life has shifted towards the internet. In the digital world, network attacks are growing concern which can cause severe financial damages and shut down entire operations. Random Forest, Support Vector Machine, and Logistic Regression have been popular machine learning models for intrusion detection as they are simple to train and explain. However, with increasing complexity and multidimensionality of network traffic, these models have struggled to detect the complex relationships between data points, thereby being less effective in real-world applications [1]. Additionally, most traditional models rely on datasets that may not cover the full spectrum of current attack patterns, limiting their generalizability in unfamiliar environments [2].

In this project, we compare classical and graph-based approaches for intrusion detection. We begin with logistic regression as a simple, interpretable baseline. Then random forest, for a relatively simple, but more robust means of prediction. Building on these, we construct two types of graphs for graph neural networks. First, a flow-based graph connecting the network traffic, and a K-Nearest Neighbors graph connecting each row with its k most similar neighbors. We implement a 4-layer graph convolutional network and a 4 attention head graph attention network. By evaluating our models through precision, recall, and F1-score, we identify which method is most suitable for detecting network attacks.

2 Literature Review

Increasing complexity of network traffic has made it more difficult to accurately predict network intrusions. In order to overcome such problems, researchers have turned towards graph-based models, which are more effective in representing relationships between data points. One of the breakthroughs here was Graph Convolutional Networks (GCNs) by Kipf and Welling [3]. GCNs allow the model to learn from the local nodes so that patterns that conventional models miss can be identified. Building on this, Velickovic et al. proposed Graph Attention Networks (GATs) [4], which assign different levels of importance to neighboring nodes to refine the model's ability to identify subtle anomalies. All these methods have already proven to be effective in detecting fraud and IoT security and have potential for network intrusion detection as well [5].

More recently, the k-Nearest Neighbor Learning with Graph Neural Networks (kNNGNN) introduced by Kang et al. [6] combines the classic kNN algorithm with graph-based learning. Instead of treating each data point in isolation, kNNGNN constructs a graph where each node (representing a network traffic instance) is linked to its k-nearest neighbors based on feature similarity. It enables the model to learn local relationships while using the power of GNNs to refine feature representations. This is particularly useful in network intrusion detection because it dynamically adapts to evolving attack patterns, reducing the reliance on predefined rules or extensive hyperparameter tuning.

Despite these advances, there are still gaps in research. While Bilot et al. presented the first overview of GNN-based intrusion detection in 2023, many advanced GNN-based methods are still unexplored in this area[7]. Traditional models are still struggling against the complexity of modern network traffic, and many graph-based methods are still computationally costly and require more tuning. This paper aims to address these gaps by combining classical machine learning methods with advanced graph neural networks and comparing them in terms of precision.

3 Data Collection

The CIC-IDS2017 dataset was used for this study, as it provided a well-structured and labeled dataset for network intrusion detection. This dataset was chosen because it includes a diverse range of real-world attack scenarios and normal network traffic, making it suitable for both supervised and unsupervised anomaly detection tasks. The data is publicly available through the Canadian Institute for Cybersecurity and Kaggle, eliminating the need for manual data collection. The dataset consists of two main components: network traffic features that capture details of connections, such as packet sizes, timestamps, inter arrival time, and protocol types, and labeled attack categories that indicate whether the traffic is normal or malicious.

An analysis of our CIC-IDS2017 flows reveals a pronounced skew in class frequencies (Table 1). Of the 2520751 total records, benign traffic dominates at 2095057 flows (83.11%), while the combined attack classes comprise only 16.89% of the data. Among attacks, DoS is the most common type at 193745 flows (7.69%), followed by DDoS (128014, 5.08%), Port Scanning (90694, 3.60%), Brute Force (9150, 0.36%), Web Attacks (SQL Injection + XSS, 2143, 0.09%) and Bots (1948, 0.08%). Below is a simplified class distribution table.

Table 1: Simplified Class Distribution

Category	Count	% of Total	Norm:Attack	Includes
Normal Traffic	2,095,057	83.11%	-	BENIGN
DoS	193,745	7.69%	10.81 : 1	GoldenEye, Hulk, Slowloris, SlotHTTPTest
DDoS	128,014	5.08%	16.37 : 1	DDoS
Port Scanning	90,694	3.60%	23.10 : 1	PortScan
Brute Force	9,150	0.36%	228.97 : 1	Brute Force
Web Attacks	2,143	0.09%	977.63 : 1	SQL Injection, XSS
Bots	1,948	0.08%	1075.49 : 1	Bot
Total	2,520,751	100.00%	-	-

Such extreme imbalance can bias supervised learners toward the majority class, leading to deceptively high overall accuracy but poor detection of rare events [10]. In particular, infrequent attacks like Bots and SQL-injection risk being under-recognized, inflating false-negative rates. Without corrective measures, classifiers may default to predicting "benign" rather than capture minority attacks.

To mitigate this, we employ class-weight balancing during model training. This strategy down-weights benign samples and up-weights under-represented attacks at each split, encouraging the learner to allocate capacity toward rare but critical patterns. When combined with group-aware splitting and controlled tree complexity, it forms the core of our approach to counter dataset bias.

4 Algorithms

To develop an effective model to detect network attacks, we employed several methods to quantify the risk of network intrusion. We started by employing less complex methods such as logistic regression and random forests to develop a baseline predictions. Then we tested more complex models, applying graph neural networks, specifically Graph Convolutional Networks (GCNs) and Graph Attention Networks (GATs).

The dataset came largely cleaned, so the amount of data preprocessing was negligible and we were able to immediately begin working on our algorithms. We chose to start with logistic regression as it is simple to implement, and would give confidence on our ability to more accurately predict network attacks in our more complex models.

This dataset is large, and some of the used models, namely the GCN and GAT are computationally expensive. Running into issues with training on the entire dataset, we opted to run our models on a subset of the data. Initially, we had doubts about the viability of taking a subset of the data while preserving meaningful connections between each data point. However, as will be discussed later, we believe working with subsets of the data does not significantly impede our model's predictive power.

Due to the class imbalance of our data, to ensure when we sample the data each class is still represented properly we used stratified sampling, which maintains the class distribution of the original dataset, which is defined as:

$n_h = n\,\frac{N_h}{N}$, where N is total examples. N_h is examples in each class, and n is the number of taken samples.

4.1 Logistic Regression

We started our analysis with logistic regression as a baseline for our results. As logistic regression is relatively efficient, we were able to run it on the entire dataset. After imputing missing values, we conducted permutation feature importance to identify which covariates are strongly associated with predicting network intrusions to simplify our list of features for the more complex models and to prevent and possible overfitting. The results of the regression is discussed in the evaluation section.

4.2 Random Forest

For the next step, we applied a Random Forest classifier to predict network anomalies. Given the size of the dataset, we opted to use a stratified subset to reduce computational demands during the testing phase.

The Random Forest classifier is well-suited for capturing complex, non-linear relationships between features, unlike traditional linear models such as logistic regression. One key benefit of using Random Forest is its ability to handle multicollinearity; by selecting a random subset of features at each split, it reduces the chance of correlated features dominating the model. Additionally, by averaging over multiple trees, the model mitigates overfitting, ensuring that no single feature disproportionately influences the predictions.

However, the random forest model is limited in interpretability, so it may be difficult to gain insight on the specific relationship between network attacks and the covariates. Also, the distribution of classes within the dataset is moderately imbalanced. Because tree models are sensitive to class imbalance [8], it is possible that the model performs poorly.

To enhance model performance, we conducted a permutation feature importance analysis to identify the most influential features. This allowed us to focus on the most relevant variables, reducing the complexity of subsequent models and minimizing the risk of overfitting. The results from the Random Forest classifier, including its evaluation and feature importance, are discussed in the following section.

4.3 Graph Convolutional Network & Graph Attention Networks (GCNs and GATs)

Following the Random Forest, we implemented Graph Neural Network based models, specifically using a Graph Convolutional Network (GCN) and Graph Attention Network (GAT).

Originally, we had planned on defining our graph construction in two ways. Our main priority was to represent the graph in terms of network flows. Each row in the dataset represents a network interaction between a source and target address. To construct a graph used for our GNN models, we planned to split each row into two tuples:

(Source IP, Source Port) $\longrightarrow$ (Destination IP, Destination Port)

However, representing the graph in this manner led to poor results. One potential reason is the class imbalance. Graph Neural Networks tend to perform poorly on imbalanced data, so we believe this is a possible culprit for the poor performance. This idea is supported due to the minimal minority class recall that results from our models when using this graph construction. Attempting to account for this class imbalance, we experimented with using Synthetic Minority Oversampling Technique (SMOTE), however, this had marginal improvements on performance. We believe this is due to SMOTE's inability to create synthetic data points whose edges between the real points were actually meaningful network flows. Our other belief is a byproduct of the class imbalance, and that is, the graph had hubs that were dominated by the majority class (benign), and outside of these hubs, the graph was relatively sparse. With such few connections between the minority class nodes, the GNN did not have sufficient opportunity to learn these patterns among these classes that distinguish them from the benign class. These two issues combined were difficult to address. If the edges were already not meaningful, and oversampling the minority classes didn't lead to new meaningful connections, then it is very difficult to have a high-accuracy GNN. Thus, we shifted our focus to our alternative idea for graph construction, KNN-GNN [6].

Constructing graphs for GNNs using K-Nearest Neighbors is an established technique, however, using a KNN-GNN for network intrusion detection is a novel approach. The graph construction is straightforward. After dropping identifying columns and standardizing the numeric data features, we compute the k most similar neighbors for each node (k=8), dropping self-matches and pruning edges that connect the training and test set to prevent data leakage. While this method of construction strays from our original plan, this solves the issue of minority classes having low importance in the graph, by ensuring each node is equally connected.

To prevent leakage, we take multiple steps. We drop all edges between test and training nodes, both ways. We preprocess only on the training data, and standardize only on training data. Additionally, we ensure basic things such as dropping identifiers and the class labels from the feature list before creating the graph. However, in the GAT and GCN we observe a lower

4.3.1 GCN and GAT Architecture: Our Graph Convolutional Network closely follows the formulation of Kipf & Welling [3]. Let the k-NN graph produced from our outlined K-Nearest Neighbors algorithm be $G = (V, E, \mathbf{X})$, where each node $v_i \in V$ represents one flow record with an 80-dimensional, standardized feature vector $\mathbf{x}_i \in \mathbb{R}^{80}$. After adding self-loops the normalized adjacency is

$$\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})\mathbf{D}^{-\frac{1}{2}},$$

with degree matrix $\mathbf{D} = \mathrm{diag}(\mathbf{A1})$. A single GCN layer performs

$$\mathbf{H}^{(l+1)} = \sigma\big(\tilde{\mathbf{A}}\,\mathbf{H}^{(l)}\mathbf{W}^{(l)}\big),$$

where $\mathbf{H}^{(0)} = \mathbf{X}$, $\mathbf{W}^{(l)}$ is a learnable weight matrix, and σ is the ReLU activation function.

Network depth and width: The final model contains 2 convolutional layers - an input layer, one hidden layer, and an output layer that produces log-probabilities for the 15 attack categories:

$$80 \longrightarrow 64 \longrightarrow 15,$$

After each hidden layer we apply a ReLU activation, and have an optimized Dropout (0.5) for potential generalization benefits.

Handling class imbalance: Because BENIGN flows greatly outnumber some attack types by around three orders of magnitude (Table 1), the loss is weighted with the square-root inverse frequency:

$$w_c = \frac{\sqrt{1/n_c}}{\sum_j \sqrt{1/n_j}} \times |C|,$$

where n_c is the number of training samples of class c and $|C| = 15$. This keeps minority classes influential without letting their weights explode.

Optimization: We used the standard Adam optimizer for training the model. The model hyper-parameters were chosen to be:

- Learning Rate = 0.01,
- Weight Decay = 5×10^{-4},
- Hidden Dimensions = 64
- Dropout = 0.5
- Number of Epochs = 100 epochs.

During training the loss is computed only on the nodes in $\mathcal{T}_{\text{train}}$ to avoid any data leakage during the training process.

Message-passing: In each GCN layer, every node aggregates information from its six nearest neighbors, via the kNN graph, using a shared $\mathbf{W}^{(l)}$ to detect patterns in that local neighborhood. Using two layers allows each node to learn from neighbors up to two connections away, allowing the model to propagate signals across the graph while maintaining computational efficiency.

The architecture used for the GAT is similar to the GCN, aside from the addition of attention. Originally, we had planned on utilizing GATs because we assumed when representing our graph as network flows, it was likely certain connections would be of higher importance than others, so weighing edges would help capture these more complex interactions. However, switching the graph to a KNN-based graph, the notion of certain edges carrying more inherent importance is less plausible as all nodes and their neighbors are similar to each other, so the learned attention weights no longer reflect meaningful differences in connectivity. In spite of this, by comparing the results of the GCN and GAT, we can reveal whether the attention offers higher performance than the GCN.

We kept the GAT configuration aligned with the GCN: 64 hidden units, identical dropout (0.5), learning rate (0.01), weight decay (5×10^{-4}), and 100 training epochs. Additionally, we use 4 attention heads. Though the most commonly used number is 8, computational efficiency concerns led us to decreasing the amount. Like the GCN class weights, w_c are computed through square-root inverse frequency rather than inverse-frequency due to the large class imbalance

5 Evaluation

5.0.1 Code: The implementation of all algorithms used in this project is available on GitHub:

`github.com/tymisiorek/ML_Project`

Logistic regression and random forest are evaluated on the entire dataset, however, the GNNs are evaluated on a 20% stratified subset of the original data, due to computational issues. Allocating more data to the GNNs caused the machine to crash, even when running on Rivanna with GPUs and multi-core CPUs, limiting us to using only a portion of the data. We utilize precision, recall, and F1-score as our primary metrics. Precision highlights the number of correct positive predictions, recall measures the fraction of positive data points that are actually predicted, while F1 measures a combination of the two. Primarily, we prioritize F1 as the combination of the two leads to a more robust metric, however, we value recall, as marking benign behavior as malicious is preferable to marking malicious behavior as benign.

5.1 Logistic Regression Evaluation

The logistic regression was fitted on the entire dataset, as it is computationally lightweight enough to run on the full data. Below is the results of the logistic regression:

Table 2: Logistic Regression Results

Class	Precision	Recall	F1-score	Support
BENIGN	1.00	0.64	0.78	569,706
Bot	0.02	0.99	0.04	587
DDoS	0.76	1.00	0.86	38,408
DoS Hulk	0.51	0.99	0.67	23,577
DoS Slowhttptest	0.09	0.89	0.16	1,650
DoS slowloris	0.15	0.91	0.25	1,739
FTP-Patator	0.17	1.00	0.29	2,380
Infiltration	0.00	0.73	0.00	11
PortScan	0.80	1.00	0.89	47,641
SSH-Patator	0.07	0.97	0.13	1,769
Web Attack – Brute Force	0.01	0.23	0.01	452
Web Attack – Sql Injection	0.00	0.33	0.00	6
Web Attack – XSS	0.01	0.82	0.02	196
Accuracy		0.70		688,122
Macro avg	0.28	0.81	0.32	688,122
Weighted avg	0.94	0.70	0.78	688,122

These results show a relatively low effectiveness in identifying malicious network traffic. This was expected, as logistic regression can only pick up linear relationships, so we predicted it to have a low accuracy. Even after performing principal component analysis (PCA), which reduced the feature set from 80 to 25, the results were relatively unaffected. However, the model was able to achieve reasonable predictive power on the higher represented groups in the data, giving encouraging results for our other methods.

To evaluate the model and measure feature importance, we employed various measures. We used permutation feature importance to determine the most significant covariates. Among the most important were FwD IAT Total, Flow Duration, Bws IAT Total, and Flow IAT Max. However, the low recall on BENIGN indicates that the model commonly misclassifies non-malicious flows as attacks. This suggests that the regression is overly aggressive in flagging underrepresented classes, which could be attributed to the overall class imbalance. To address this, we could attempt to change the classification thresholds, however, as logistic regression was meant to serve as a baseline for comparison, we opted to instead explore the other models more in-depth.

5.2 Random Forest Evaluation

We trained a Random Forest on a 70/30 stratified split of 252K flows, preserving class proportions to mitigate the 83:17 benign–attack imbalance. The ensemble comprised 100 trees, each limited to a maximum depth of 10 with at least five samples per leaf, and used `class_weight="balanced"`. These hyperparameters (i) prevent over-specialization on the majority class, (ii) bound model complexity to curb overfitting, and (iii) guarantee that minority classes appear in every terminal node. Below is a predicted class distribution of our random forest model.

Table 3: Random Forest Results

	Prec.	Rec.	F1	Support
BENIGN	1.00	0.97	0.99	102492
Bot	0.09	0.98	0.16	85
DDoS	1.00	1.00	1.00	5691
DoS GoldenEye	0.97	0.99	0.98	473
DoS Hulk	0.89	1.00	0.94	10435
DoS Slowhttptest	0.94	0.99	0.96	245
DoS slowloris	1.00	0.98	0.99	289
FTP-Patator	1.00	1.00	1.00	371
Heartbleed	0.00	0.00	0.00	5
Infiltration	0.00	0.00	0.00	1
PortScan	0.99	1.00	1.00	7126
SSH-Patator	0.96	1.00	0.98	266
Web Attack – Brute Force	0.14	0.83	0.24	65
Web Attack – Sql Injection	0.00	0.00	0.00	1
Web Attack – XSS	0.14	0.67	0.23	30
Accuracy			0.98	127575
Macro avg	0.61	0.76	0.63	127575
Weighted avg	0.99	0.98	0.98	127575

On the held-out set, the forest attained 98% accuracy and a weighted F of 0.98, markedly outperforming logistic regression (70% accuracy, weighted F=0.78). RF reduced both false positives and false negatives across most attack types: macro-average precision rose from 0.28 (LR) to 0.61, while recall remained high. In contrast, LR's single linear decision boundary cannot capture interactions such as `if packet_length_max > 1200 AND fwd_iat_mean < 0.05`, whereas RF's hierarchical, piecewise splits naturally model such conjunctions.

Limiting tree depth to 10 strikes the balance between expressivity and generalization: shallow trees cannot memorize individual flows yet can still learn non-linear patterns across 80 features. Requiring at least five samples per leaf further ensures that each split is supported by sufficient data, stabilizing predictions on rare classes.

Gini-importance analysis highlights the top predictors: `Flow Duration`, `Total Fwd Packets`, `Fwd Packet Length Max`, `Bwd IAT Std`, and `Flow IAT Mean`. These results confirm that both temporal (inter-arrival times) and volumetric (packet counts, lengths) statistics drive the model's ability to distinguish flood-style attacks from benign traffic.

5.3 GCN Evaluation

The GCN was trained on an 80/20 train-test split on 20% of the data using stratified sampling. Below are the classification results produced by our implementation:

Table 4: GCN Stats

Class	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.9911	0.9526	0.9714	75,925
Bot	0.0345	0.0370	0.0357	81
DDoS	0.9676	0.9670	0.9673	5,119
DoS Hulk	0.8180	0.9186	0.8654	3,145
DoS SlowHTTPTest	0.7244	0.9318	0.8151	220
DoS Slowloris	0.6400	0.8889	0.7442	234
FTP-Patator	0.3264	0.9600	0.4871	325
Web Attack-SQL Injection	0.0000	0.0000	0.0000	1
PortScan	0.7954	0.9913	0.8826	6,356
SSH-Patator	0.9699	0.5265	0.6825	245
Web Attack - Brute Force	0.1301	0.8421	0.2254	57
Infiltration	0.0000	0.0000	0.0000	1
Web Attack-XSS	0.0000	0.0000	0.0000	24
Accuracy		0.9524		—
Macro avg	0.4921	0.6166	0.5136	91,733
Weighted avg	0.9647	0.9524	0.9564	91,733

The model achieves a weighted F_1 of 0.956, significantly higher than logistic regression and marginally better than the GAT on the same split—by nearly perfect identification of the four most frequent attack families (DDoS, DoS Hulk, DoS SlowHTTPTest, PortScan). The message-passing mechanism is particularly effective for burst-type attacks whose statistical signatures (very short inter-arrival times, extreme packet-length maxima) are amplified when aggregated over the eight-hop KNN neighbourhood. Performance collapses on the scarcest classes (Bot, Web-XSS, Infiltration). Inspection of the learned adjacency shows that these rare flows are often embedded inside majority-BENIGN regions of the k-NN graph, so their messages are drowned out layer after layer ("oversmoothing"). The square-root weighting helps recall for FTP-Patator and Web Brute Force but cannot compensate when support drops below ~ 100 samples.

With an accuracy of 95.2 % and a weighted F_1 of 0.956, the GCN captures the dominant traffic patterns well. Precision–recall scores stay high as long as a class has at least a few hundred examples, but once support drops below that threshold (e.g. Bot or Web XSS) the signal from minority nodes is overwhelmed by their majority-BENIGN neighbours—even after square-root class weighting.

Figure 1 (Shown on the next page) shows rapid convergence with no sign of overfitting—the test curve tracks the training curve within 0.01 NLL from epoch 30 onward—confirming that 100 epochs are sufficient. GCN's strong results on the abundant classes demonstrate that graph convolutions can leverage spatial correlations in network flows. However, without explicit imbalance remedies, graph-based models remain brittle on the long tail of attack types.

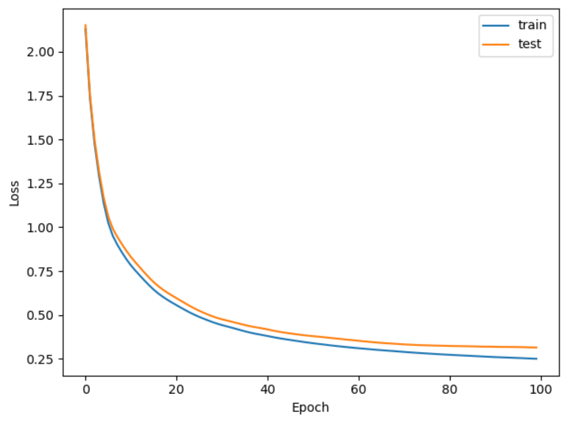

Figure 1: GCN Training vs Testing Loss

5.4 GAT Evaluation

The GAT model was also trained on 80% of the subsetted data (20% of the full dataset) and was evaluated on a 20% test set. The results are displayed below:

Table 5: GAT Results

Class	Precision	Recall	F1-score	Support
BENIGN	0.9901	0.9509	0.9701	75,925
Bot	0.0000	0.0000	0.0000	81
DDoS	0.9638	0.9617	0.9627	5,119
DoS Hulk	0.7828	0.9100	0.8416	3,145
DoS SlowHTTPTest	0.7816	0.9273	0.8482	220
DoS Slowloris	0.6749	0.8162	0.7389	234
FTP-Patator	0.4100	0.9877	0.5794	325
Web Attack – SQL Injection	0.0000	0.0000	0.0000	1
PortScan	0.7900	0.9899	0.8787	6,356
SSH-Patator	0.5683	0.5265	0.5466	245
Web Attack – Brute Force	0.1174	0.8421	0.2060	57
Infiltration	0.0000	0.0000	0.0000	1
Web Attack – XSS	0.0000	0.0000	0.0000	24
Accuracy		0.9502		–
Macro avg	0.4676	0.6086	0.5056	91,733
Weighted avg	0.9615	0.9502	0.9540	91,733

While taking a stratified maintains class proportions, accuracy in predicting minority classes remained low. Training on a larger portion of the dataset would be preferable, increasing the number of points for minority classes, however, this was not feasible given our computational resources.

Despite the model's shortcomings on underrepresented classes, the GAT performed relatively well. Calculating the weights with square-root inverse frequency instead of inverse frequency, we bias towards the underrepresented classes, and perform slightly better on such classes than originally anticipated. This comes at the expense of recall for the BENIGN class. However, allowing recall for non malicious traffic to suffer while achieving higher accuracy for malicious attacks is intended behavior, as misclassifying traffic as malicious is less damaging than completely missing actual malicious traffic. Thus, using square-root inverse weighing carried merit. However, prediction accuracy for the least common classes, such as Web Attack-XSS and Bot remained low, highlighting the limitations of graph-based models on skewed data.

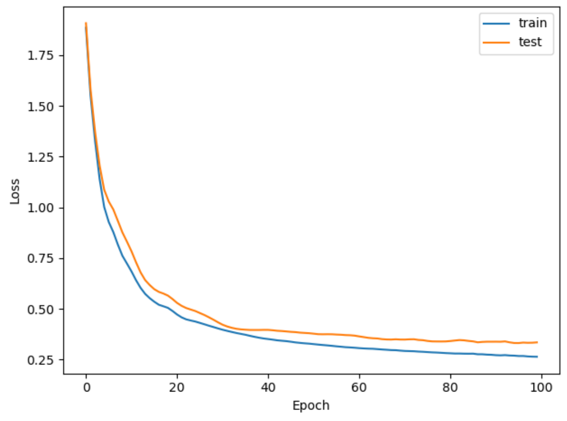

Figure 2: GAT Training vs Testing Loss

Looking at the Training and Test Loss Graph 2, 100 epochs seems to be an appropriate amount, as the loss for both curves begins to converge. One concern is the test loss being very similar to the training loss. Initially we thought this could be due to data leakage, however, as discussed, we took multiple precautions to prevent this, so we believe this could be attributed to the models general inability to accurately predict any classes outside of the top 4 most common, due to the heavy class imbalance.

Compared to the GCN, the GAT achieved slightly better performance on some minority classes, such as FTP-Patator and DoS SlowHTTPTest. However, the overall performance was not substantially better, which suggests that the attention mechanism had a limited impact in this context. As mentioned previously, we believe that the KNN-based graph already enforces similarity-based connections, making additional attention weights less informative. However, the slight variations in performance show that attention perhaps is slightly better at identifying certain classes than the GCN and vice versa.

5.5 Conclusion

Our results suggest that the random forest is most effective in correctly classifying network traffic, giving high precision and recall across most classes, while also remaining relatively computationally efficient compared to the graph-based models. Its strength lies in its ability to capture complex, nonlinear interactions between features, which is especially important given the high dimensionality of the dataset (80 features). Additionally, we visualized the feature importance derived from the trained forest. Among the top contributors were features such as Flow Duration, Total Fwd Packets, Fwd Packet Length Max, and Flow IAT Mean, indicating that time-based and flow-based statistics are crucial for distinguishing attack patterns. Finally, we conducted cross-validation with 5 folds to ensure generalizability. The variance across folds was minimal (standard deviation of F1-Score = ±0.008), confirming that the model's performance is stable across different subsets of the data.

The GCN improved heavily upon the baseline logistic regression, however, failed to accurately predict minority classes as well as the random forest. While utilizing a GAT was initially appealing, it offered only slight differences from the GCN due to our chosen method of graph construction. Although the random forest outperformed the GNNs, through more thorough hyperparameter tuning as well as increased diversity in our data, we believe that they could possibly perform to a similar level as the random forest based on our given results. By leveraging techniques such as Bayesian optimization, we believe more suitable hyperparameters could be found. Additionally, training over more epochs, while increasing the weights for the minority classes could help the model learn the smaller classes while having enough epochs to still learn the majority. Finally, having access to the full dataset for training would be advantageous for the graph models. Since GNNs are more sensitive to class imbalance than random forest, having more data points for the minority classes could help.

The random forest, GCN, and GAT all perform well on a weighted average level, however, suffer from issues in predicting minority classes. Ultimately, although the random forest model remains the strongest model in terms of accuracy and efficiency, we believe that each model used could be further improved. Implementing the aforementioned strategies, we expect our models to improve performance even on the lowest represented classes, closing the remaining gap in accuracy and yielding a robust intrusion detection system.

5.5.1 *Team Roles:*

- These are main roles, however, everyone contributed to each others parts as well
- Ty Misiorek (zemby): GAT
- Sanket Doddabendigere (kfr9mc): GNNs
- Shirley Li (fky9xf): Random Forest
- Shawn Thomas (yzy2bt): GCN
- Ansh Pathapadu (tqc7wb): Logistic Regression

References

[1] Sommer, R., & Paxson, V. (2010). Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *IEEEXplore.* https://ieeexplore.ieee.org/document/5504793

[2] Liu, H., & Lang, B. (2019). Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey. *Applied Sciences*, 9(20), 4396. https://www.mdpi.com/2076-3417/9/20/4396

[3] Kipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. In *arXiv.* https://arxiv.org/abs/1609.02907

[4] Velickovic, P., et al. (2018). Graph Attention Networks. In *arXiv.* https://arxiv.org/abs/1710.10903

[5] Zhou, J., et al. (2020). Graph Neural Networks: A Review of Methods and Applications. *ScienceDirect.* https://www.sciencedirect.com/science/article/pii/S2666651021000012

[6] Kang, J., et al. (2020). k-Nearest Neighbor Learning with Graph Neural Networks. In *MDPI.* https://www.mdpi.com/2227-7390/9/8/830

[7] Bilot, T., El Madhoun, N., Al Agha, K., & Zouaoui, A. (2023). Graph Neural Networks for Intrusion Detection: A Survey. *IEEE Access*, 11, 49114-49139. https://ieeexplore.ieee.org/document/10123384

[8] Is Random Forest a Good Option for Unbalanced Data Classification?, *Stack Exchange*, (2016), https://stats.stackexchange.com/questions/242833/is-random-forest-a-good-option-for-unbalanced-data-classification, Accessed: Feb. 19, 2025.

[9] E-GraphSAGE: A Graph Neural Network based Intrusion Detection System for IoT, *Arxiv*, (2022), https://arxiv.org/pdf/2103.16329, Accessed: March. 26, 2025.

[10] S. S. Gopalan, D. Ravikumar, D. Linekar, A. Raza, and M. Hasib, "Balancing Approaches towards ML for IDS: A Survey for the CSE–CIC IDS Dataset," in *Proc. Int. Conf. Commun., Signal Process. Their Appl. (ICCSPA)*, Dubai, UAE, Mar. 2021, pp. 1–8, doi:10.1109/ICCSPA49915.2021.9385742.