Neural Networks CS 752

Network Traffic w/ Feature Ablation and Additive Attention

Professor Laura Dietz

Written by Jack Schneider

11/27/24

I. Task:

Definition: Given inputs of network traffic ***w*** parameters such as packet sizes,bwd metrics, fwd metrics, etc. predict traffic label ***y*** as 'Benign' or 'DDos'. Analyze how ***additive attention*** and ***feature ablation*** affect model performance.
Inputs: Input of network parameters ***w***, taken from network traffic DDos data set
Outputs: Output a prediction ***y*** of 'Benign' or 'DDos'
Reimplementation: No Reimplementation

II. Motivation:

Domain: Network Traffic and Cyber Security
Importance: Cyber attacks are an ever growing issue for everyday people and company owners. Resulting in the release of personal information like passwords and costing companies millions of dollars in damages.
Helpful: Solving this task is important because this could help prevent different cyber attacks from occurring. An efficient model could definitely improve the detection of bad network traffic and would be an additional defensive measure for a company saving companies money.

III. Data set:

Download: https://www.kaggle.com/datasets/chethuhn/network-intrusion-dataset/data labeled as "Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv"
Confirmation: Data has been downloaded and accessed in jupyter notebook
Train/Test split: First 80% of network traffic examples will be used as training data and the proceeding 20% will be used for testing and validation.
Suitability: The data that I found will be suitable for training and testing to make accurate predictions with 43.3% of the data being DDos. The train test split contains 78,130 DDos examples in the training set and 19,588 DDos examples in the testing set.

Preparations: To prepare the data for training and testing, separating the data frame into X being everything not ' Label' and Y being ' Label'. Then removing any numbers that are nan and clipping the values to be in a smaller range of numbers to prepare for normalization. After preparing the data to be suitable for training the data is then split 80% train and 20% test and loaded into train and test data loaders. Each iteration of the data loader has the shape (64,78) where 64 is the batch size and 78 is the number of features.

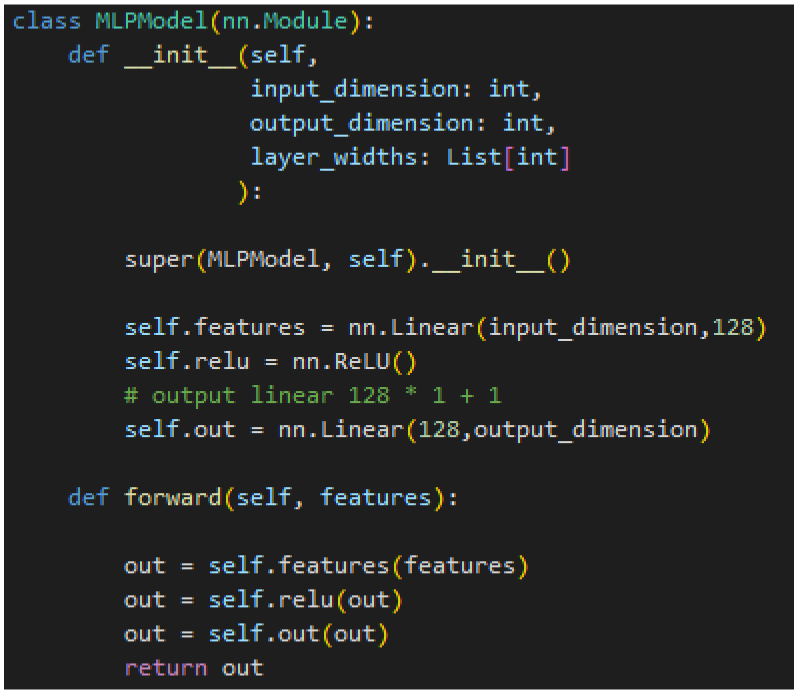
IV. Standard Neural Network Approach

NN Architecture: Multi-Layer Perceptron
Units/Parameters/Degrees of Freedom: 8,449 Parameters
Appropriate: A multi-layer perceptron is a suitable model for this use case since multilayer perceptrons are able to perform classification through the use of linear layers and activation functions.

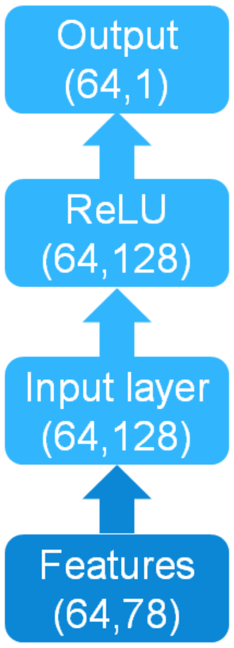
Self.features layer: A linear layer to transform into the dimensions of d_model attaching learned weights and biases.
Relu: An activation function to pronounce positives else zero. Improving recognition
Self.out layer: A linear layer to transform d_model into the dimensions of out attaching learned weights and biases.

Rationale:
Multi-layer perceptrons are great at basic classification problems. But arent the best when it comes to complex relationships which might be involved when discussing network traffic. This model was picked in hopes of performing moderately well on the data set with room for improvement through feature attention and feature ablation.

Pseudocode:



```
class MLPModel(nn.Module):
    def __init__(self,
                 input_dimension: int,
                 output_dimension: int,
                 layer_widths: List[int]
                ):

        super(MLPModel, self).__init__()

        self.features = nn.Linear(input_dimension,128)
        self.relu = nn.ReLU()
        # output linear 128 * 1 + 1
        self.out = nn.Linear(128,output_dimension)

    def forward(self, features):

        out = self.features(features)
        out = self.relu(out)
        out = self.out(out)
        return out
```



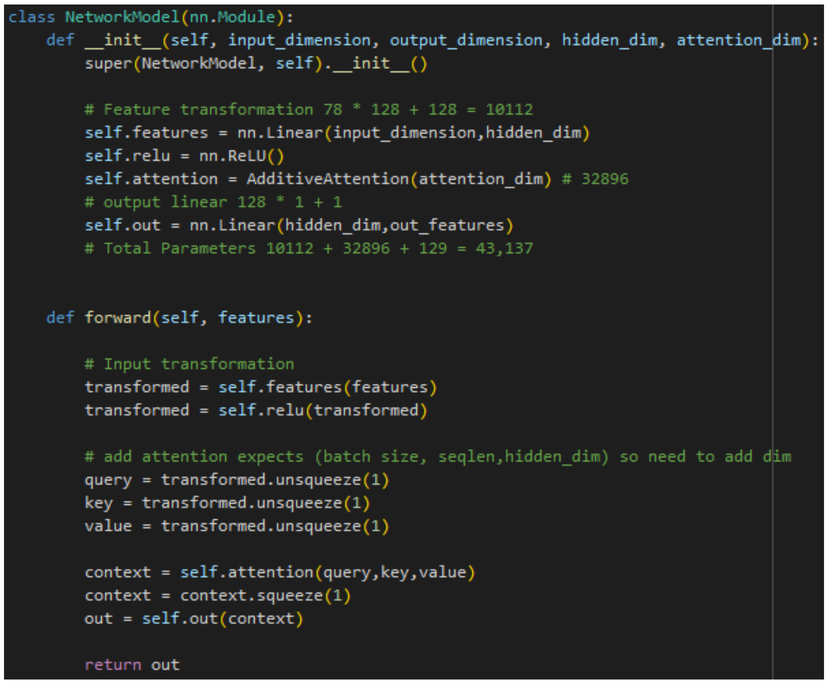
V. Custom Neural Network Approach

NN Architecture: MLP + Additive Attention
Units/Parameters/Degrees of Freedom: 43,137
Appropriate: Adding Additive Attention into the Multi-layer perceptron model will further build upon the identification of complex relationships with the data set.

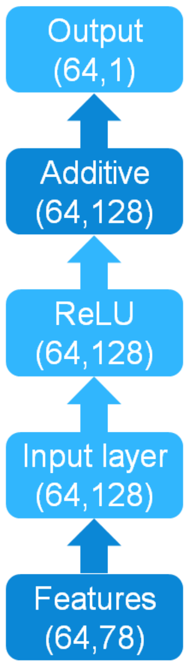
Self.features layer: A linear layer to transform into the dimensions of d_model attaching learned weights and biases.
Relu: An activation function to pronounce positives else zero. Improving recognition
Self.attention layer: Additive attention to help further improve model relationships identification.
Self.out layer: A linear layer to transform d_model into the dimensions of out attaching learned weights and biases.

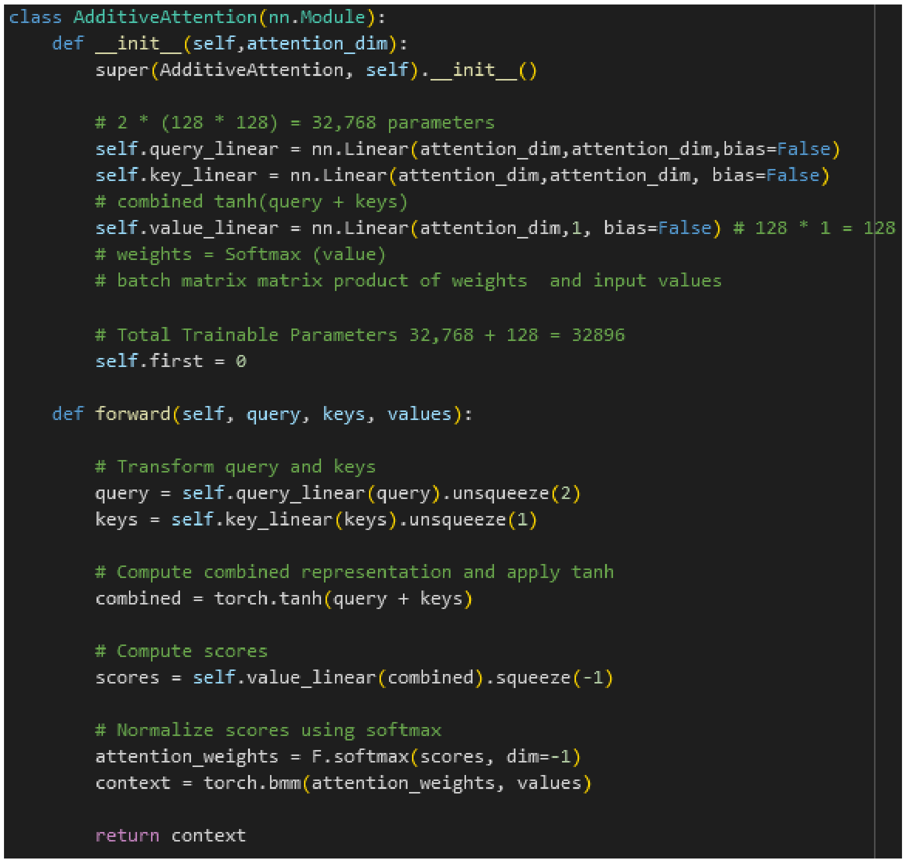
Rationale:
The NetworkModel follows a similar architecture to the Multi-layer Perceptron with an additional layer of additive attention. Additive attention will be helpful for the model to perform well with data involved in complex relationships like Network Traffic Data. This addition will hopefully improve performance metrics compared to the Multi-layer Perceptron Model.
Additionally, the NetworkModel will be tested with feature ablation. Feature ablation will be helpful to further improve model performance in a couple ways. Decreasing the overall training time by removing unneeded features and reducing initial dimensionality as well as removing features that might hurt the models performance.

Pseudocode:

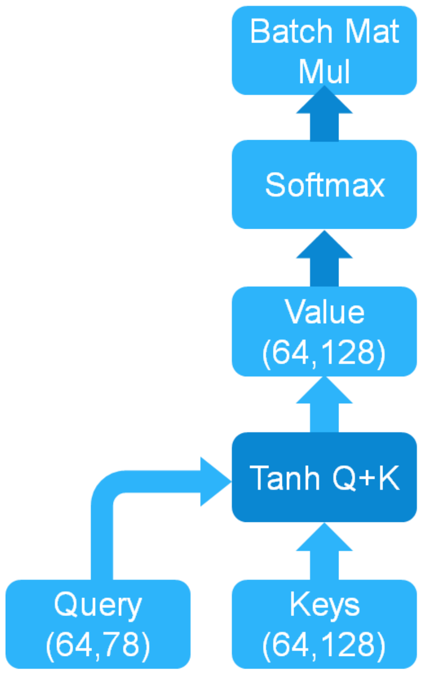


```
class NetworkModel(nn.Module):
    def __init__(self, input_dimension, output_dimension, hidden_dim, attention_dim):
        super(NetworkModel, self).__init__()

        # Feature transformation 78 * 128 + 128 = 10112
        self.features = nn.Linear(input_dimension,hidden_dim)
        self.relu = nn.ReLU()
        self.attention = AdditiveAttention(attention_dim) # 32896
        # output linear 128 * 1 + 1
        self.out = nn.Linear(hidden_dim,out_features)
        # Total Parameters 10112 + 32896 + 129 = 43,137


    def forward(self, features):

        # Input transformation
        transformed = self.features(features)
        transformed = self.relu(transformed)

        # add attention expects (batch size, seqlen,hidden_dim) so need to add dim
        query = transformed.unsqueeze(1)
        key = transformed.unsqueeze(1)
        value = transformed.unsqueeze(1)

        context = self.attention(query,key,value)
        context = context.squeeze(1)
        out = self.out(context)

        return out
```





```python
class AdditiveAttention(nn.Module):
    def __init__(self,attention_dim):
        super(AdditiveAttention, self).__init__()

        # 2 * (128 * 128) = 32,768 parameters
        self.query_linear = nn.Linear(attention_dim,attention_dim,bias=False)
        self.key_linear = nn.Linear(attention_dim,attention_dim, bias=False)
        # combined tanh(query + keys)
        self.value_linear = nn.Linear(attention_dim,1, bias=False) # 128 * 1 = 128
        # weights = Softmax (value)
        # batch matrix matrix product of weights  and input values

        # Total Trainable Parameters 32,768 + 128 = 32896
        self.first = 0

    def forward(self, query, keys, values):

        # Transform query and keys
        query = self.query_linear(query).unsqueeze(2)
        keys = self.key_linear(keys).unsqueeze(1)

        # Compute combined representation and apply tanh
        combined = torch.tanh(query + keys)

        # Compute scores
        scores = self.value_linear(combined).squeeze(-1)

        # Normalize scores using softmax
        attention_weights = F.softmax(scores, dim=-1)
        context = torch.bmm(attention_weights, values)

        return context
```



VI. Existing solutions

No Existing Solutions Known

VII. Evaluation

Hyper-parameters:

D_model: 128
Learning_rate: 0.005
Optimizer: Optim.SGD
Criterion: MSEloss
Epochs: 20
D_attention: 128

Measures: The measures that will be used to determine the models performance are confusion matrices of each model, F1 score, last reported MSE,

Comparison: All methods will be performed on the same data set and keeping track of metrics on each model. As Well as comparing accuracies the confusion matrices parameters over time in addition to f1 score.
Significance: T-score significance test will be used to determine the whether improvement between a model and baseline is significant or not
Plots/ Tables:

VIII. Expectation

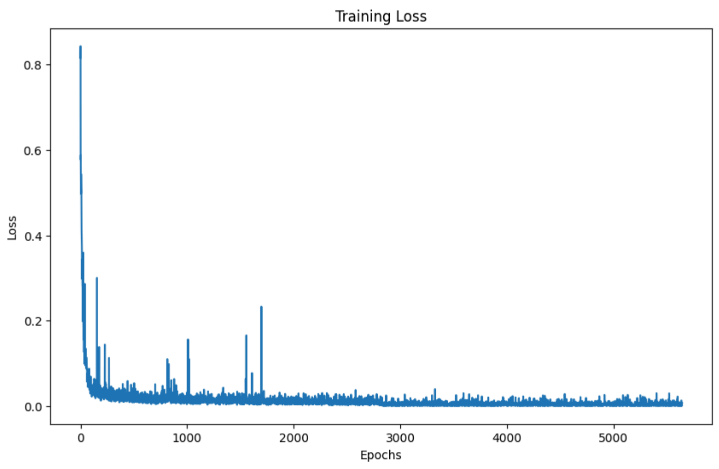
Models

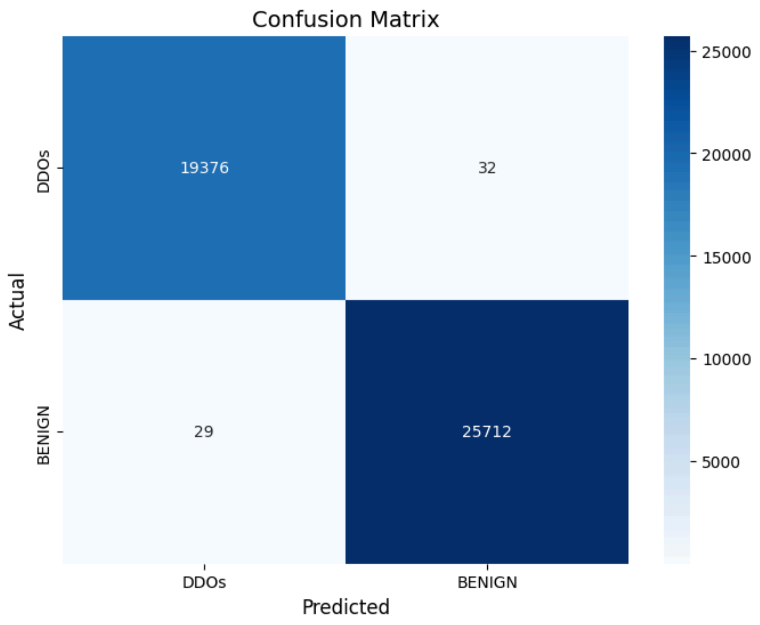
1. MLP
2. MLP + Additive Attention
3. MLP + Additive Attention + Feature Ablation

Between the three different strategies, MLP + Additive Attention + Feature Ablation will perform the best. The Additive Attention will further improve the MLP's performance on complex relationships and Feature Ablation might potentially reduce training time and improve performance.

IX. Results

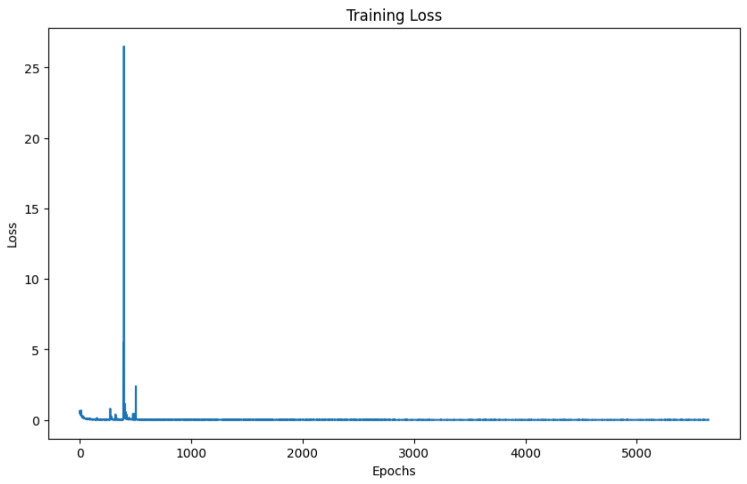
Network Model

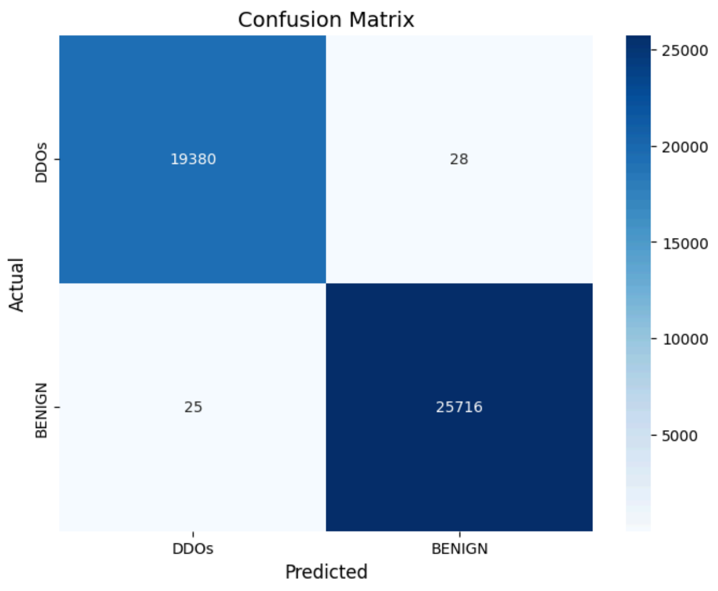




Accuracy:	0.9986489180269773
Precision:	0.9988733926420885
Recall:	0.9987569919204475
F-1 score:	0.9988151888899679

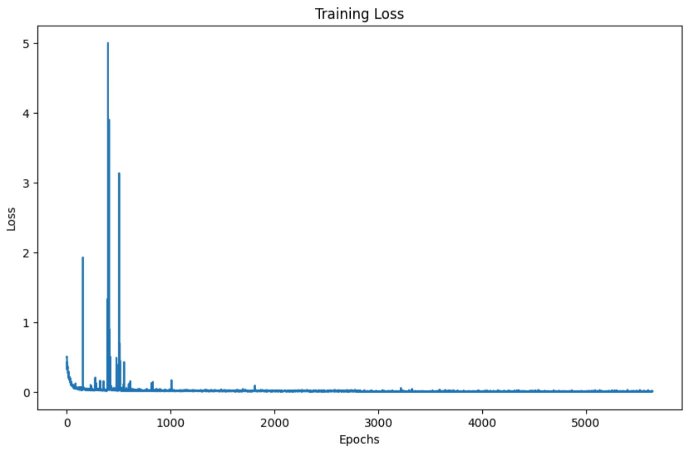
Multi-Layer Perceptron

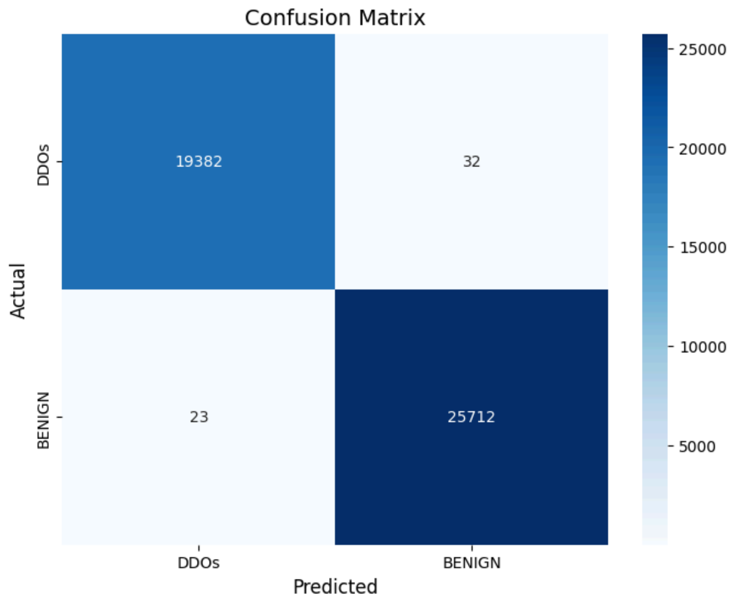




Accuracy:	0.9988261091054066
Precision:	0.9990287867604212
Recall:	0.9989123679303915
F-1 score:	0.9989705739535787

Network Model w/ Feature Ablation



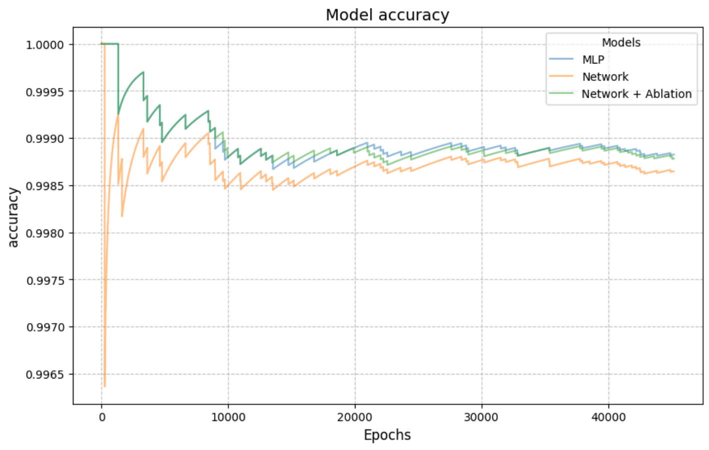


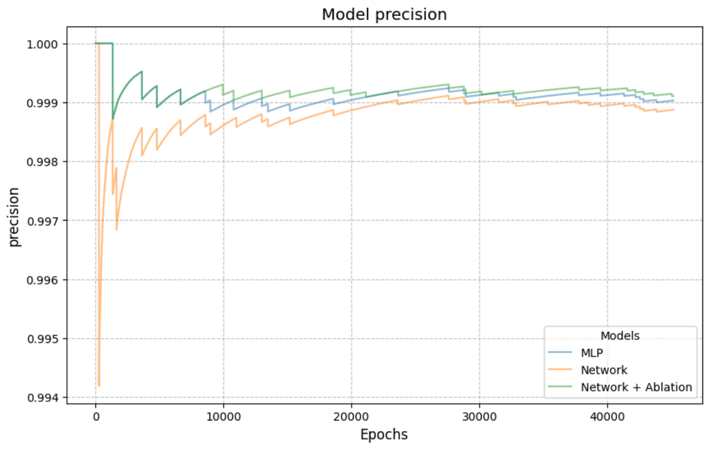
Accuracy:	0.9987818113357992
Precision:	0.9991062755002914
Recall:	0.9987569919204475
F-1 score:	0.9989316031779949

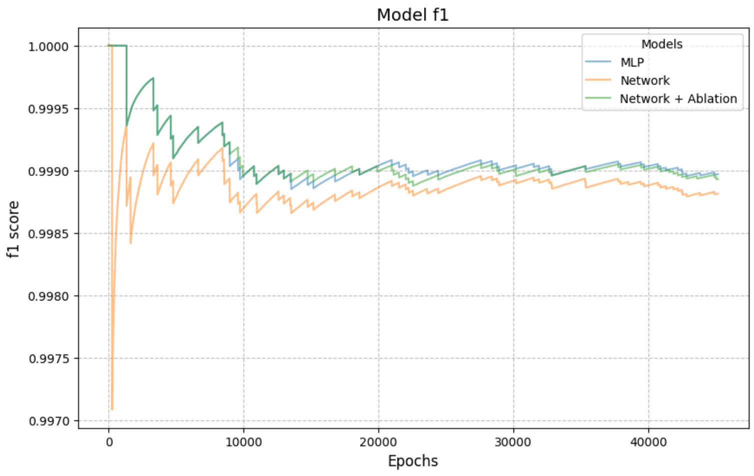
Analysis

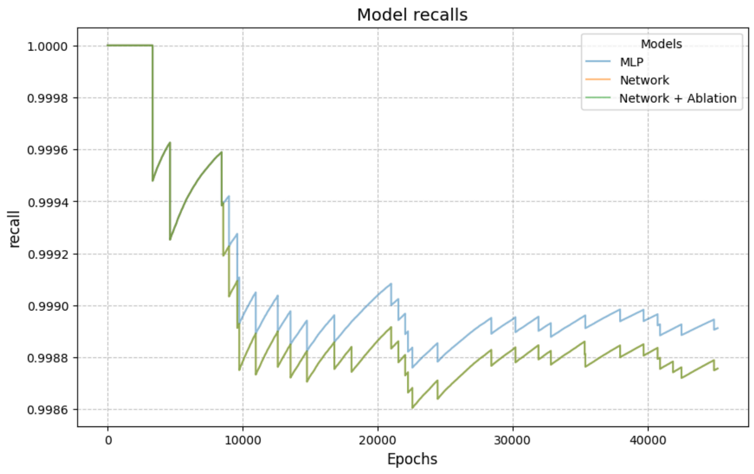
- ***Base Model (MLP):*** Performed fairly well achieving high accuracy (0.9988), precision (0.9990), recall (0.9989), and F1 score (0.9989). This model effectively classified benign and DDoS network traffic, showcasing the robustness of the multi-layer perceptron on this dataset.
- ***Network Model + Additive Attention:*** Demonstrated a similar performance with the base model. The additive attention mechanism enhanced the model's ability to capture complex relationships in the data, leading to better generalization and predictive power.
- ***Network Model + Additive Attention + Feature Ablation:*** The feature ablation model also achieved an accuracy (0.9988) but slightly improved precision (0.9991) while maintaining the F1 score (0.9989). This suggests that removing less significant features reduced noise and improved performance, albeit marginally.

Comparisons









T-Test between Accuracies

Network Model + Feature Ablation
T-Statistic: -9.488025194530975
Critical t-value: 1.6448873781830808
P-Value: 1.0
Decision using T-value:
There is a significant difference between the base model and the sample model.
The models are likely to perform differently.
Decision using P-value:
There is no significant difference between the base model and the sample model.
The models perform similarly.

Network Model
T-Statistic: -293.25372255391176
Critical t-value: 1.6448873781830808
P-Value: 1.0
Decision using T-value:
There is a significant difference between the base model and the sample model.
The models are likely to perform differently.
Decision using P-value:
There is no significant difference between the base model and the sample model.
The models perform similarly.

T_test between Precision

Network Model + Feature Ablation
T-Statistic: 121.34089185842632
Critical t-value: 1.6448873781830808
P-Value: 0.0
Decision using T-value:
There is a significant difference between the base model and the sample model.
The models are likely to perform differently.
Decision using P-value:
There is a significant difference between the base model and the sample model.
The models are likely to perform differently.

Network Model
T-Statistic: -174.24244502823055
Critical t-value: 1.6448873781830808
P-Value: 1.0
Decision using T-value:
There is a significant difference between the base model and the sample model.
The models are likely to perform differently.
Decision using P-value:
There is no significant difference between the base model and the sample model.
The models perform similarly.

T-Test between Recall

Network Model + Feature Ablation
T-Statistic: -64.08463314671832
Critical t-value: 1.6448873781830808
P-Value: 1.0
Decision using T-value:
There is a significant difference between the base model and the sample model.
The models are likely to perform differently.
Decision using P-value:
There is no significant difference between the base model and the sample model.
The models perform similarly.

Network Model
T-Statistic: -64.08463314671832
Critical t-value: 1.6448873781830808
P-Value: 1.0
Decision using T-value:
There is a significant difference between the base model and the sample model.
The models are likely to perform differently.
Decision using P-value:
There is no significant difference between the base model and the sample model.
The models perform similarly.

T Test between F1 score

Network Model + Feature Ablation
T-Statistic: -9.605167097276428
Critical t-value: 1.6448873781830808
P-Value: 1.0
Decision using T-value:
There is a significant difference between the base model and the sample model.
The models are likely to perform differently.
Decision using P-value:
There is no significant difference between the base model and the sample model.
The models perform similarly.

Network Model
T-Statistic: -301.73524993633936
Critical t-value: 1.6448873781830808
P-Value: 1.0
Decision using T-value:
There is a significant difference between the base model and the sample model.
The models are likely to perform differently.
Decision using P-value:
There is no significant difference between the base model and the sample model.
The models perform similarly.

X. Results

Results Table

	Accuracy	**MSE**	**F-1**	**T-score (accuracy)**
Multi-Layer Perceptron	0.999	0.0052	0.999	n/a
Network Model	0.999	0.0066	0.999	No significance
Network Model (Ablation)	0.999	0.0064	0.999	No significance

Conclusion

After performing three different strategies of models on network traffic. The overall performance increase was not significant for most metrics. One metric that was significantly improved was involving precision between the MLP model and the network model with feature ablation. When also talking about training time the MLP model was quicker than the network model without the feature ablation. I believe the performances of the two initial models were similar due to the initial large amount of trainable parameters. The large amount of trainable parameters allowed for the Multi-layer Perceptron model to perform well initially leaving not a lot of room for significant improvement. Feature ablation proved beneficial to an extent since metrics such as accuracy were improved from the network model. Four separate different ablations were tested. The following groups where removing 'bwd' feature information, '*fwd' feature information, '*flow' related information, and '*Flag Count' feature information. The Feature ablation that proved to improve overall performance of the network model was removing '*flow' related features. From this the conclusion is that Feature ablation proved beneficial for network model performance.