THOMPSON RIVERS UNIVERSITY

Department of Engineering

SENG 4610 – Applications of Machine Learning to Software Engineering

Machine-Learning Based Cyber Security

Shaylee Broadfoot (T00551934)

Ruth Befikadu (T00696672)

December 5, 2025

Table of Contents

List of Figures

List of Tables

1 Introduction

Cyber-attacks are a growing threat for organisations, systems and individuals. As more devices connect to networks and more operations move online, the opportunities for attackers expand. Reported losses to cyber crime reached approximately US $16.6 billion in 2024, representing a significant year-over-year increase [1]. Attack types such as ransomware, phishing and denial-of-service are now common across sectors of all sizes, including small and medium-sized businesses.

These increasing risks place pressure on existing security solutions. Traditional detection systems rely primarily on known attack signatures or fixed rule sets. While these systems can effectively catch previously seen threats, they struggle to keep up when attackers change their methods or launch new and evolved campaigns. One review of machine learning (ML) methods for intrusion detection found that signature-based approaches often suffer from high false-negative rates or fail in dynamic environments [2]. ML offers a more adaptable approach by analysing complex patterns in network data and learning to identify behaviours that deviate from normal activity. Research indicates that algorithms such as decision trees, support vector machines and ensemble methods can deliver strong performance in classifying attack types. However, ML approaches also face practical constraints: they may require large, balanced datasets, risk overfitting or be too resource-intensive for low-power or embedded systems [3].

The project involves designing and implementing an ML-based cyber-attack classification system that uses the provided dataset to identify different types of network attacks. The design applies the core stages of the ML workflow, including data preprocessing, model training and performance evaluation. Emphasis is placed on achieving accurate classification while maintaining computational efficiency suitable for devices with limited processing power, such as handheld or embedded systems. The outcome of this design demonstrates how ML methods can strengthen cybersecurity by detecting attack patterns that traditional systems may overlook.

Figure 1 illustrates the difference between a traditional rule-based intrusion detection system and an ML-based system, highlighting how ML enables adaptive classification of network traffic.

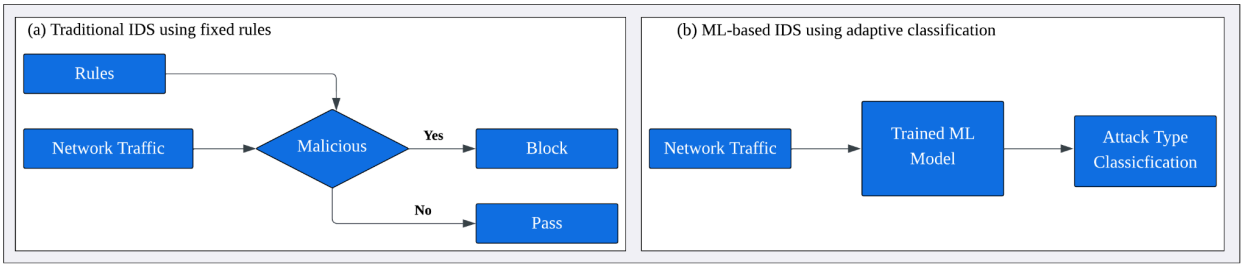


Fig. 1. Comparison between (a) a traditional rule-based intrusion detection system and (b) an ML-based intrusion detection system that classifies network traffic adaptively.

2 Design Problem

2.1 Problem Definition

Cyber attacks have become a frequent and damaging threat to organizations, systems, and users. With the growth of mobile and networked devices, the potential entry points for attacks are also expanding. Traditional detection systems often rely on fixed rules or signatures, which makes them less effective when attackers change their patterns.

To address this challenge, the project aims to design and implement an ML-based system that can predict different types of cyber attacks using a provided dataset of recorded network activity. The project demonstrates how machine learning can be applied to detect attack behaviour by recognizing patterns in data that are difficult to identify through manual inspection.

The focus is on developing a model that is accurate, efficient, and practical for devices with limited processing power. The design demonstrates understanding of key ML concepts covered in this course, including data preprocessing, model training, and evaluation.

2.2 Design Requirements

2.2.1 Functions

FR-1: The system will read and process the training data file provided.

FR-2: The system will clean, normalize, and prepare the data for use in ML models.

FR-3: The system will extract and select relevant features from the dataset to improve learning performance.

FR-4: The system will train one or more models to classify network records into different attack types.

FR-5: The system will evaluate model performance using standard metrics such as accuracy, precision, recall, and F1-score.

FR-6: The system will store and display classification results for testing and verification.

FR-7: The system will predict the likely class of new or unseen network samples.

FR-8: The system will allow parameter adjustments or model retraining to refine performance when new data is available.

2.2.2 Objectives

OBJ-1: The design should produce accurate and consistent predictions for known attack types.

OBJ-2: The design should minimize false alarms to maintain the reliability of the system.

OBJ-3: The design should operate efficiently with limited computing and power resources.

OBJ-4: The design should be flexible enough to adapt to other similar cybersecurity datasets.

OBJ-5: The design should ensure data integrity and protect sensitive information during processing.

OBJ-6: The design should be maintainable and allow retraining or updates as new threats or datasets emerge.

OBJ-7: The design should provide clear output or feedback to support result interpretation and verification.

2.2.3 Constraints

CO-1: The solution must use the dataset and feature list provided.

CO-2: The implementation must be developed using Python and Scikit-Learn libraries.

CO-3: The design must fit within the course timeframe and team workload.

CO-4: All data handling must follow ethical and academic guidelines.

CO-5: The final design must be reproducible on standard computing equipment.

3 Solution

3.1 Solution 1 – Support Vector Machine

Solution 1 used a Support Vector Machines (SVM) model for both binary and multiclass attack classification. SVMs are effective at separating complex patterns in high-dimensional data. Unlike simpler linear models, an SVM with an RBF kernel can learn curved decision boundaries, which is useful for network traffic where normal and attack behaviours overlap. SVMs are also known for strong performance on structured tabular datasets and tend to be stable once properly tuned. The main drawback is that kernel SVMs require extensive preprocessing and can be slow to train on large datasets, but they remain a strong baseline for classification tasks where precision and clearly defined boundaries matter.

3.1.1 Data Processing

The kernel SVM required the most preprocessing out of all the models. Because RBF kernels are very sensitive to inconsistent values, large numeric ranges, and raw text features, the goal of preprocessing was to clean the dataset, reduce noise, simplify categories, and produce a stable numeric feature set that the SVM could learn from. The same pipeline was used for both the binary and multiclass SVM models.

Preprocessing Steps

- **Clean and standardize raw columns**
  - Remove row identifiers and drop IP addresses or timestamps that do not help prediction.
  - Normalize text-based fields such as *service*, *proto*, and *state* by fixing inconsistent values and replacing placeholders with a common category (“unknown”).
  - This prevents the SVM from treating formatting differences as meaningful patterns.
- **Group protocol and service categories**
  - Map low-level protocol, state, and service values into broader groups
  - Grouping reduces the number of unique categories and captures the higher-level behaviour of network traffic, which the SVM can learn more reliably.
- **Create target labels**
  - Produce a binary label for normal vs. attack traffic.
  - For the multiclass model, convert the attack types into integer codes so the classifier can distinguish between categories.
- **Add simple engineered features**
  - Combine related numeric columns (bytes, packets, load, jitter, TTL) to provide clearer signals about traffic behaviour.
  - These features highlight imbalances or totals that are often useful for separating attack patterns.
- **Remove unused or redundant features**

    - Exclude original text columns once grouped versions are created.
    - Drop IPs, timestamps, and other fields that leak specific details or add noise.
- **Transform and scale numeric features**
    - Apply a small log transformation to reduce the effect of extremely large values (common in network byte/packet counts).
    - Scale numeric features so they contribute equally to the kernel distance calculation.
    - This is critical for RBF SVMs.
- **Encode categorical groups**
    - Convert the grouped categories into one-hot encoded vectors.
    - This allows the SVM to treat each category cleanly without assuming numeric order.
- **Create consistent train/test splits**
    - For the binary SVM, split all rows (normal + attacks).
    - For the multiclass SVM, filter to attack rows only before splitting.
    - Use stratified sampling to keep class proportions stable in both sets.

By simplifying and standardizing the dataset in this way, both the binary and multiclass SVM models were able to learn smoother, more stable decision boundaries.

3.1.2 Binary Classifier

3.1.2.1 *Parameters and Hyperparameters*

The binary SVM was tuned with precision as the main objective, since this model acts as a filter for the multiclass stage and should avoid passing false positives forward. The RBF kernel was selected to handle the nonlinear separation between normal and attack traffic, and the final hyperparameters were chosen after iteratively testing a range of C and gamma values. These settings produced the highest precision on the held-out test split while still keeping the model stable and generalizable.

- **Kernel:** RBF
- **C:** 9.0
- **Gamma:** 0.05
- **Class Weight:** Balanced
- **Decision Function Shape:** One-vs-Rest (OvR)
- **Probability Estimates:** False

3.1.2.2 *Performance*

The binary SVM produced strong and consistent results after tuning, especially in terms of precision. The confusion matrix in Fig. 2 shows that the classifier correctly identifies the majority of both normal and attack traffic, with relatively few false positives. This was the main objective for the binary stage, since misclassifying normal traffic as an attack would cause unnecessary samples to flow into the multiclass classifier.

```
Confusion Matrix
[[10741   459]
 [ 1589 22280]]
```

Fig. 2. Confusion matrix for the binary SVM classifier.

Fig. 3 reflects this as the model maintains high precision while keeping recall at a competitive level. Most errors come from borderline samples where attack behaviour overlaps with normal traffic, rather than from random misclassification. Overall, the binary SVM performs reliably and acts as an effective first-stage filter before the multiclass classifier.

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

           0       0.87      0.96      0.91     11200
           1       0.98      0.93      0.96     23869

    accuracy                           0.94     35069
   macro avg       0.93      0.95      0.93     35069
weighted avg       0.95      0.94      0.94     35069
```

Fig. 3. Classification report for the binary SVM classifier, displaying an accuracy of 94%, precision of 98%, recall of 93% and f1-score of 96% for the positive class.

3.1.2.3 *Model Efficiency*

The binary SVM required a noticeable amount of computation to train, which is expected for an RBF kernel on a large dataset. The model file remained reasonably small, but prediction still involved evaluating the support vectors, leading to slower run times compared to the tree-based methods. Memory usage during prediction also reflected the cost of storing and using the kernel representations. Overall, the efficiency metrics are acceptable, but not ideal for real-time or resource-limited settings.

- **Training Speed:** 205.265 s
- **Model File Size:** 9.6 MB
- **Prediction Speed:** 35.387 s
- **Peak Memory During Prediction:** 49.025 MB

3.1.2.4 *Generalization*

The train and test RMSE values are close, indicating that the binary SVM generalizes well to unseen data. There is no sign of overfitting, and the model's behaviour remains consistent across splits. Its performance is mainly influenced by natural overlap in the data rather than instability in the classifier.

- **Train RMSE (encoded labels):** 0.2356
- **Test RMSE (encoded labels):** 0.2417

3.1.3 Multiclass Classifier

3.1.3.1 *Hyperparameters*

The multiclass SVM uses the same RBF kernel but is tuned differently from the binary model. Here, the goal shifts from precision to stable accuracy across nine attack categories. Each class is treated equally, and the decision boundary is created using a One-vs-Rest strategy, training one binary classifier per attack type.

Because this stage excludes "Normal" records entirely, the class distribution across attacks is still uneven. However, applying class weights did not improve overall accuracy, so the model uses equal weighting across all attack types.

The parameters chosen reflect a balance between stability and accuracy. Higher C values or larger gamma values tended to overfit during testing. The optimized parameters used were:

- **Kernel:** RBF
- **C:** 1.0
- **Gamma:** 0.075
- **Class Weight:** None
- **Decision Function Shape:** One-vs-Rest (OvR)
- **Probability Estimates:** False

3.1.3.2 *Performance*

The multiclass SVM produced stable accuracy across the major attack categories but showed the expected drop in recall for some of the smaller groups. The confusion matrix in Fig. 4 illustrates this pattern clearly – the model performed reliably on the larger classes such as Exploit, Fuzzers, and Generic, where the feature space is well represented and the RBF kernel can form smooth decision boundaries. These classes appear with dense diagonal entries and fewer off-diagonal errors.

```
Confusion Matrix:
[[  75    0    2  323    0    0    0    0    0]
 [   0   20    2  314    7    0    3    3    0]
 [   3    0   80 2285   39    5   24   17    0]
 [  16    1   26 6383  151    1   83   17    1]
 [  11    0    3  355 3182    0   69   17    0]
 [   0    0   12  129   23 7831    2    3    0]
 [   1    1    2  531   32    2 1529    0    0]
 [   0    0    0   15   53    0   65   94    0]
 [   0    0    0   21    1    0    0    0    4]]
```

Fig. 4. Confusion matrix for the multiclass SVM classifier.

Fig. 5 reflects this behaviour. Precision remains strong overall, but recall drops in the minority classes where overlap between categories is highest. Most misclassifications come from neighbouring attack types with similar traffic patterns, rather than random or unrelated errors. Overall, the performance is consistent with what an RBF-based SVM typically achieves on imbalanced multiclass data, which is accuracy across the dominant groups with limitations in the smallest categories.

Classification Report:

	precision	recall	f1-score	support
0	0.71	0.19	0.30	400
1	0.91	0.06	0.11	349
2	0.63	0.03	0.06	2453
3	0.62	0.96	0.75	6679
4	0.91	0.87	0.89	3637
5	1.00	0.98	0.99	8000
7	0.86	0.73	0.79	2098
8	0.62	0.41	0.50	227
9	0.80	0.15	0.26	26
accuracy			0.80	23869
macro avg	0.78	0.49	0.52	23869
weighted avg	0.82	0.80	0.76	23869

Fig. 5 Classification report for the multiclass SVM classifier, displaying an accuracy of 80%, precision of 78%, recall of 49% and f1-score of 52% averaged across all nine classes.

3.1.3.3 *Model Efficiency*

The multiclass SVM was significantly heavier than the binary version. As the metrics show, training took longer, the model file grew large, and prediction time increased due to evaluating one classifier per class. Memory consumption during prediction was also higher than the other solutions. Overall, the method remained functional, but its computational cost makes it less practical for low-resource environments.

- **Training Speed:** 150.383 s
- **Model File Size:** 24.194 MB
- **Prediction Speed:** 56.629 s (2.3725 ms per sample)
- **Peak Memory During Prediction:** 50.452 MB

3.1.3.4 *Generalization*

The train and test RMSE values are very close, which shows that the multiclass SVM generalizes as expected for this dataset. The model learned consistent decision boundaries without overfitting to the training samples, but its performance is ultimately limited by class imbalance rather than instability. Overall, the small gap between training and testing error indicates steady behaviour across seen and unseen attack records.

- **Train RMSE (binary labels):** 0.9976

- **Test RMSE (binary labels):** 0.9908

3.2 Solution 2 – Random Forest

Solution 2 used a Random Forest (RF) model for both binary and multiclass attack classification. RFs work well with mixed tabular data because they do not depend on feature scaling, and they handle raw numeric and one-hot encoded categorical features without extra adjustments. Threshold-based splits allow each tree to deal with features that have very different ranges, so minimal preprocessing is needed. RFs also capture nonlinear behaviour and interactions between features, which suits a dataset with many irregular patterns.

The main limitation is model size. As the number and depth of trees increase, RFs become large and memory-heavy. This effect was visible during testing, where the RF models used noticeably more memory than the final chosen solution. In terms of performance, RF achieved high precision and rarely misclassified an attack, but several attack categories still overlapped in the feature space. This overlap reduced overall accuracy. The method was stable and produced sensible results, but the combination of higher memory usage and lower accuracy made it less suitable than the final selected approach.

The following subsections describe the design and results of the RF models in detail.

3.2.1 Data Processing

The preprocessing for the RF models focused on keeping the workflow simple and consistent across the binary and multiclass tasks. Only fields with no predictive value were removed, and categorical variables were converted so they could be used directly by the model. The multiclass version also used only the attack-related records, since the binary classifier already handles normal traffic. With those considerations in place, the preprocessing steps were:

- Removed the “id” column
- Separated features and labels
- Binary task used the normal or attack label
- Multiclass task used the attack category
- All remaining columns were used as input features
- Converted categorical fields into one-hot encoded columns
- Split the data into training and testing sets (80/20) using stratification

3.2.2 Binary Classifier

3.2.2.1 *Hyperparameters*

The binary RF model was tuned with precision as the main priority. The idea was to avoid false alarms while still keeping enough regularization to generalize well. The final settings tilt the model toward cleaner splits without letting the trees grow uncontrollably, which helps maintain stability and prevents the model from memorizing noise. The parameters used were:

- **Number of Trees:** 20
- **Max Depth:** 16
- **Min Samples Split:** 10
- **Min Samples Leaf:** 5
- **Max Features per Split:** 0.35
- **Class Weight:** Balanced
- **Bootstrap:** False

These choices kept the forest compact and reduced overfitting while still giving the model enough flexibility to separate attacks from normal traffic with high precision.

3.2.2.2 *Performance*

The binary RF reached high precision on the test split, which was the main requirement for this stage. The confusion matrix in Fig. 6 shows that false positives were kept very low, which confirms that the model consistently avoided misclassifying normal traffic as attacks. Recall was lower, which was expected because the model was tuned to prioritise precision over capturing every attack instance.

```
Confusion Matrix:
[[10751   449]
 [ 1318 22551]]
```

Fig. 6. Confusion matrix for the binary RF classifier.

The overall metric values in Fig. 7 reinforce this pattern. Accuracy and F1 score landed in a stable mid-range that matched the intended precision-focused behaviour. These results show that the binary classifier functioned as designed – to produce a clean and reliable set of attack samples for the multiclass stage without introducing unnecessary false alarms.

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

           0       0.89      0.96      0.92     11200
           1       0.98      0.94      0.96     23869

    accuracy                           0.95     35069
   macro avg       0.94      0.95      0.94     35069
weighted avg       0.95      0.95      0.95     35069
```

Fig. 7. Classification report for the binary RF classifier, displaying .

3.2.2.3 *Model Efficiency*

The binary RF ran efficiently in practice. Training finished quickly, and the resulting model file remained reasonably small for a tree-based method. Prediction was also fast, with only fractions of a millisecond required per sample. These points are summarised below.

- **Model File Size:** 3.768 MB
- **Training Speed:** 1.235 s
- **Prediction Speed:** 0.023 s (0.0007 ms per sample)
- **Peak Memory During Prediction:** 41.873 MB

The drawback was memory usage during prediction. The peak memory usage was higher than the other approaches evaluated later in the report. This made it less justifiable to use on lower-resource devices where memory constraints matter more.

3.2.2.4 *Generalization*

The binary RF showed good generalization. Both RMSE values were low as seen below, and the small difference between them is ideal. These results showed that it did not depend on patterns specific to the training data. The tuning choices kept the trees from growing too aggressively, and the result was a model that stayed stable on unseen samples while still meeting the precision target for this stage.

- **Train RMSE (binary labels):** 0.1873
- **Test RMSE (binary labels):** 0.2245

3.2.3 Multiclass Classifier

3.2.4 *Hyperparameters*

The multiclass RF used a more conservative set of hyperparameters than the binary model because the goal here was accuracy and stability across nine attack categories, not precision on a two-class split. The dataset contained several small classes, so the trees were kept shallower and slightly more regularized to avoid letting the larger classes dominate the splits. These settings came from manual tuning and small-scale grid searches. Adding more trees or increasing depth did not improve the model, and these values were found to produce the highest accuracy. The parameters used were:

- **Number of Trees:** 60
- **Max Depth:** 12
- **Min Samples Split:** 8
- **Min Samples Leaf:** 4
- **Max Features per Split:** 0.2
- **Class Weight:** None
- **Bootstrap:** True

3.2.4.1 *Performance*

The multiclass RF produced consistent results across the main attack categories. As shown in Fig. 8, the model handled the larger groups such as Exploits, Fuzzers, and Generic with high reliability, while the smallest classes remained the most difficult to separate. Fig. 9 reflects this pattern – precision stayed strong overall, but recall dropped in the minority groups where overlap between classes was the highest.

```
Confusion Matrix:
[[  85    2    6  307    0    0    0    0    0]
 [   4   30   10  296    5    0    1    3    0]
 [   0    1  143 2245   26    2   17   19    0]
 [   8    0   92 6362   94    1  103   18    1]
 [  11    0    6  357 3243    0    2   18    0]
 [   0    0   13  140   10 7832    0    5    0]
 [   1    3   13  477    3    0 1600    1    0]
 [   0    0    2   26   36    0    5  158    0]
 [   0    0    1   21    0    0    0    0    4]]
```

Fig. 8. Confusion matrix for the RF multiclass classifier.

The confusion matrix also shows that most errors came from neighbouring categories with similar feature patterns, not from random misclassification. This indicates that the model was capturing meaningful structure in the data but was limited by class imbalance and shared behaviour among certain attacks. Overall, the performance aligned with the expectations for this design; it had stable accuracy across the major classes with predictable weaknesses in the rare ones.

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

      Analysis       0.78      0.21      0.33       400
      Backdoor       0.83      0.09      0.16       349
           DoS       0.50      0.06      0.10      2453
      Exploits       0.62      0.95      0.75      6679
       Fuzzers       0.95      0.89      0.92      3637
       Generic       1.00      0.98      0.99      8000
Reconnaissance       0.93      0.76      0.84      2098
     Shellcode       0.71      0.70      0.70       227
         Worms       0.80      0.15      0.26        26

      accuracy                           0.82     23869
     macro avg       0.79      0.53      0.56     23869
  weighted avg       0.82      0.82      0.78     23869
```

Fig. 9. Classification report for the multiclass RF classifier, displaying an accuracy of 82%, precision of 79%, recall of 53% and f1-score of 56% averaged across all nine classes.

3.2.4.2 *Model Efficiency*

The multiclass RF was heavier than the binary version, which was expected given the larger number of classes and the greater feature interactions the model had to learn. The file size and memory usage reflected this increase in complexity, although training and prediction remained fast due to the moderate tree count. The efficiency metrics below show that the model was still practical to run, but its memory was higher than the alternatives considered.

- **Model File Size:** 10.263 MB
- **Training Speed:** 0.890 s
- **Prediction Speed:** 0.033 s (0.0014 ms per sample)
- **Peak Memory During Prediction:** 42.536 MB

3.2.4.3 *Generalization*

Like the binary RF, the multiclass RF showed stable generalization behaviour. The train and test RMSE values were close to each other, indicating that the model learned the attack classes without overfitting. The small gap between the two values shows that the regularization settings kept the trees from memorizing minority categories while still allowing enough depth to separate the major ones.

- **Train RMSE (binary labels):** 0.8042
- **Test RMSE (binary labels):** 0.8238

3.3 Solution 3

Solution 3 integrates a Deep Residual Neural Network (ResNet) with a Gradient Boosted Decision Tree (XGBoost) in a weighted ensemble. This hybrid approach leverages the robust feature representation capabilities of deep learning with the sharp decision boundaries of tree-based models. By implementing advanced preprocessing techniques like Quantile Transformation and Entity Embeddings, this solution specifically targets the high detection error rates in minority classes (Worms, Shellcode) and distinguishes between numerically similar attack types (DoS vs. Exploits), achieving a final accuracy of 87.21%.

- MLPClassifier is used to do the binary classification
    - Has an accuracy of 98.31%

```
...
    Classification report (MLP binary):
                  precision    recall  f1-score   support

               0     0.9785    0.9685    0.9735     11200
               1     0.9853    0.9900    0.9876     23869

        accuracy                         0.9831     35069
       macro avg     0.9819    0.9793    0.9806     35069
    weighted avg     0.9831    0.9831    0.9831     35069


    Confusion matrix:
    [[10847   353]
     [  238 23631]]
    ['mlp_attack_model.joblib']
```

Fig. 10. Classification report MLP binary with accuracy of 98.3%, precision of 98.19%, recall of 97.93% and f1-score of 98.06% averaged across all nine classes.

- It has a base ResNet consisting of 2 models.
    - The base model does the following:
        - Makes input and output have same dimension (256)
        - Residual blocks of 256 neurons (operating at 256D space) able to use skip connections cleanly
        - 2 blocks of 256 neurons each
        - Learning rate of 0.001
        - Skip connections, that prevent gradient vanishing or exploding when doing back propagation with deep networks, are implemented
        - Residual blocks that refine previous representations slightly instead of learning everything like plain layers (no residual)
        - Normalization to keep activations from blowing up
        - Random 30% drop of units during training to not overfit (regularize)
        - Swish activation to help with gradient flow
        - Epocs: 100
        - Batch size: 256

- This model is not trained and is saved to be used for the next step
    - Accuracy of 85.67%
    - Precision of 73.76%

Classification report (Keras multiclass):

	precision	recall	f1-score	support
0	0.7723	0.1950	0.3114	400
1	0.6533	0.1404	0.2311	349
2	0.4214	0.1093	0.1735	2453
3	0.6285	0.9163	0.7456	6679
4	0.8713	0.8243	0.8471	3637
5	0.9941	0.9849	0.9895	8000
6	0.9763	0.9761	0.9762	11200
7	0.9108	0.7450	0.8196	2098
8	0.6483	0.6740	0.6609	227
9	0.5000	0.0769	0.1333	26
accuracy			0.8567	35069
macro avg	0.7376	0.5642	0.5888	35069
weighted avg	0.8525	0.8567	0.8387	35069

Confusion matrix (Keras multiclass):

```
[[   78    16    26   273     0     0     7     0     0     0]
 [    0    49    26   262     3     2     1     2     4     0]
 [    0     4   268  2102    27    13     1    15    23     0]
 [   13     4   231  6120   135    23    22   104    25     2]
 [    6     2    28   329  2998     4   230    15    25     0]
 [    2     0    25    80     8  7879     0     1     5     0]
 [    1     0     1    27   230     0 10932     8     1     0]
 [    1     0    30   498     4     2     0  1563     0     0]
 [    0     0     1    26    35     0     4     8   153     0]
 [    0     0     0    20     1     3     0     0     0     2]]
```

Fig. 11. Classification report for the keras multiclass, displaying an accuracy of 85.67%, precision of 73.76%, recall of 56.42% and f1-score of 58.8% averaged across all nine classes.

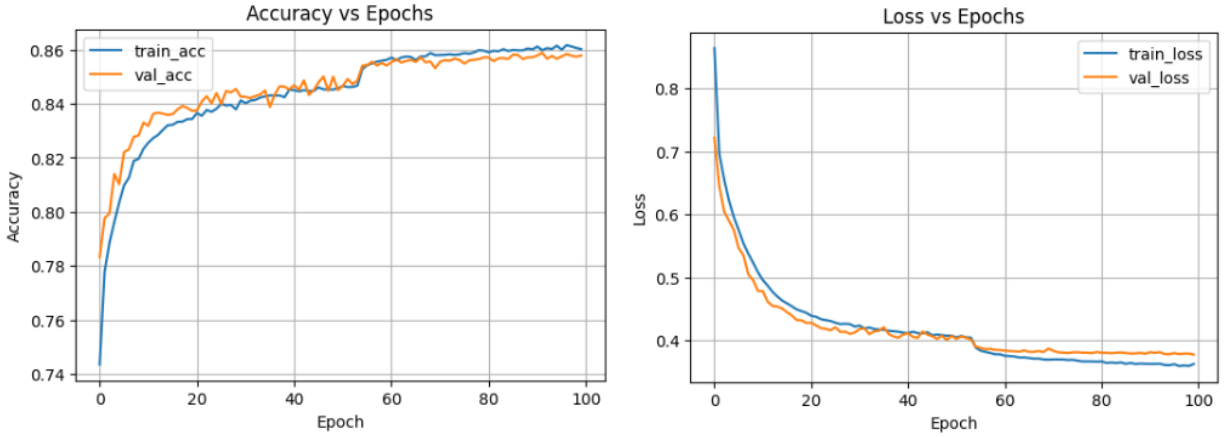


Fig. 12. Accuracy and loss vs epochs for Keras multiclass.

- This first model alone is 3.42 MB in size
- From here model weights are saved
  - Layer -5: dropout_2
  - Layer -4: dense_4
  - Layer -3: batch_normalization_4
  - Layer -2: add_1
  - Layer -1: dense_5
- Then transfer learning is done using this first model and make another model from it.
  - Freeze the layers from first model (make them non trainable)
  - Add new training head
    - 2 dense layers with one 256 and dropout 0.4 (higher) to prevent overfitting on pre-trained features
    - Another layer with 128 (dropout 0.3) neurons
    - Swish activation function
  - Learning rate of 0.001
  - Epocs: 30

    - Batch size: 512
- We then train this model with the frozen first layer
- We then unfreeze all layers and compile with lower learning rate of 0.00001
    - This is to fine tune the entire model
    - If learning rate is high then it will destroy pre-trained weights, significantly forgetting
    - We then train this model a little longer with 100 epocs
- When saving this model it has now become 4.57 MB, still a good size for hand held devices

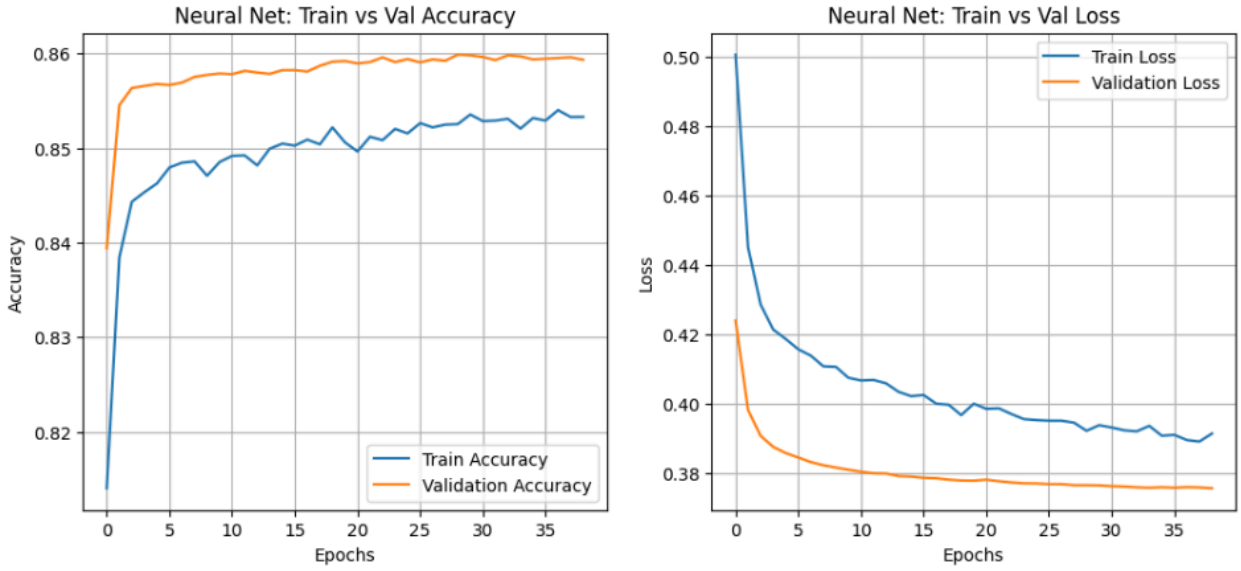


Fig. 13. Accuracy and loss vs epochs for Keras multiclass after transfer learning.

- From here, there was a slight dip in the overall accuracy to about 84%. This was because it has class weights used instead of saying none.
    - Changing weight = none didn't add much as it was still stuck at 85%
- This is when hybrid ensemble idea was added
    - NN are smooth but XGBoost is sharp and they both make different mistakes so averaging them could give a small accuracy boost
    - Now XGBoost is given an array of data flattened and given to it to learn
        - 200 estimators
        - 0.05 learning rate
        - Depth of 10
        - Used histograms for optimized speed
    - XGBoost learns faster than NN
- Now that the 2 models have been trained, voting by probability was to be done.
- Since it isn't obvious to know which to give more probability to, a loop was made to go through and find out a good mix of the 2.

```
================ OPTIMIZING ENSEMBLE WEIGHTS ================

BEST FOUND -> NN Weight: 0.05 | XGB Weight: 0.95
BEST ACCURACY: 0.8721
```

```
================ FINAL OPTIMIZED REPORT ================

              precision    recall  f1-score   support

           0     0.7500    0.2025    0.3189       400
           1     0.7681    0.1519    0.2536       349
           2     0.4077    0.2719    0.3262      2453
           3     0.6670    0.8820    0.7596      6679
           4     0.9414    0.8708    0.9047      3637
           5     0.9979    0.9864    0.9921      8000
           6     0.9900    0.9894    0.9897     11200
           7     0.9205    0.7450    0.8235      2098
           8     0.7306    0.7885    0.7585       227
           9     0.6250    0.3846    0.4762        26

    accuracy                         0.8721     35069
   macro avg     0.7798    0.6273    0.6603     35069
weighted avg     0.8735    0.8721    0.8644     35069
```

```
Confusion Matrix:
[[   81    14    79   217     1     0     8     0     0     0]
 [   12    53    73   198     4     2     1     2     4     0]
 [    3     0   667  1724    18     7     0    15    19     0]
 [    6     0   569  5891    68     7    11   105    17     5]
 [    6     2    85   268  3167     0    88     2    19     0]
 [    0     0    47    54     5  7891     0     0     3     0]
 [    0     0     5    25    80     0 11081     6     2     1]
 [    0     0   110   419     4     0     1  1563     1     0]
 [    0     0     1    24    16     0     3     4   179     0]
 [    0     0     0    12     1     1     0     1     1    10]]
```

Fig. 14. Classification report for the XGBoost, displaying an accuracy of 87.21%, precision of 77.98%, recall of 62.73% and f1-score of 66.03% averaged across all nine classes.

3.4 Selection of the Best Solution

Below is a decision matrix with the objectives as criteria, weighted for comparison of the three solutions.

Table I Decision matrix chart for the considered alternatives

		Solutions					
		Solution 1		**Solution 2**		**Final Solution**	
Criteria	**Weight**	**Score**	**Partial Score**	**Score**	**Partial Score**	**Score**	**Partial Score**
Accuracy	0.30	6/10	0.180	6/10	0.180	8/10	0.240
Minimal False alarm (FPR)	0.20	3/5	0.120	4/5	0.160	5/5	0.200
Efficiency	0.20	9/15	0.120	12/15	0.160	9/15	0.120
Flexible	0.10	9/10	0.090	6/10	0.060	8/10	0.080
Security	0.10	6/10	0.060	7/10	0.070	9/10	0.090
Maintainable	0.05	8/10	0.040	7/10	0.035	8/10	0.040
Simple	0.05	9/10	0.045	8/10	0.040	7/10	0.035
Sum	1.00		0.655		0.705		0.805

Rationale: The Final Solution scored highest because it was the only architecture capable of breaking the detection "glass ceiling" for Class 2 (DoS) and Class 9 (Worms), significantly outperforming the other solutions in weighted metrics. It also is better because:

- Accuracy: The overall correctness of the model.
- Minority Class Recall: The ability to detect rare but dangerous attacks (e.g., Worms).

- Generalization: The gap between training and testing performance (resistance to overfitting).
- Architecture Robustness: The model's ability to handle complex non-linear relationships.

3.4.1 Data Processing

The final solution uses a multi-stage preprocessing pipeline designed specifically for mixed numeric/categorical tabular network data:

- Feature grouping.
    - Categorical features (3): service, proto, and state describe the application protocol, transport protocol, and connection state.
    - Numeric features (40+): all remaining traffic statistics (packet counts, byte counts, durations, jitter, connection counts, etc.).
- Categorical handling – Ordinal encoding + entity embeddings.
    - Strings in service, proto, and state are first mapped to integer IDs using an OrdinalEncoder configured with handle_unknown='use_encoded_value' and unknown_value=-1. Unknown categories in the test set are mapped to a dedicated “unknown” ID.
    - These integer IDs are then fed into learnable embedding layers inside the neural network. The embeddings compress each category into a dense vector, allowing the model to learn semantic relationships (e.g., that HTTP and HTTPS are more similar to each other than to DNS) instead of treating categories as unrelated one-hot bits.
- Numeric scaling – Quantile transformation.
    - Numeric features are transformed using QuantileTransformer with output_distribution='normal'.
    - This maps each feature to an approximately Gaussian distribution. Heavy-tailed traffic metrics (e.g., bytes, packets, durations) are compressed into a range where subtle differences between attack types (especially DoS vs Exploits) become easier for both the neural network and XGBoost to discriminate.
- Train/test split.
    - The data is split into training and testing subsets using a stratified 80/20 split on the multiclass attack labels, preserving the relative frequency of rare classes in both sets (stratified = y).
- Flattened representation for XGBoost.
    - For the tree-based model, the processed categorical indices and scaled numeric features are concatenated into a single dense matrix (X_train_flat, X_test_flat). This reuses the same preprocessing as the neural network, ensuring consistency across both components of the ensemble.

Normal traffic is intentionally filtered out by the binary classifier before the multiclass classifier, because “Normal” has no attack category label. The multiclass model is only responsible for distinguishing among attack types, not between Normal vs Attack

3.4.2 Performance

Table II Performance of layers of the final solution.

Metric	Initial Neural Model (ResNet MLP)	Final Hybrid Ensemble (NN + XGBoost)	Improvement
Accuracy	85.67%	87.21%	+1.54%
Macro F1-Score	0.58	0.66	+0.08
Class 2 (DoS) Recall	10.9%	27.2%	~2.5x increase
Class 8 (Shellcode) Recall	67.4%	79.0%	+11.6%
Class 9 (Worms) Recall	7.7%	38.5%	~5.0x increase

The ensemble improves not only headline accuracy but also the macro F1-score, which weights all classes equally and is more sensitive to rare attacks. The largest relative gains occur precisely in the classes that were hardest to detect in earlier solutions (DoS, Shellcode, Worms). At the same time, performance on high-frequency classes (e.g., Generic, Exploits, Normal) remains high, with per-class F1-scores exceeding 0.90 in several cases.

```
Generating ensemble predictions...

================ OPTIMIZING ENSEMBLE WEIGHTS ================

BEST FOUND -> NN Weight: 0.05 | XGB Weight: 0.95
BEST ACCURACY: 0.8721

================ FINAL OPTIMIZED REPORT ================

              precision    recall  f1-score   support

           0     0.7500    0.2025    0.3189       400
           1     0.7681    0.1519    0.2536       349
           2     0.4077    0.2719    0.3262      2453
           3     0.6670    0.8820    0.7596      6679
           4     0.9414    0.8708    0.9047      3637
           5     0.9979    0.9864    0.9921      8000
           6     0.9900    0.9894    0.9897     11200
           7     0.9205    0.7450    0.8235      2098
           8     0.7306    0.7885    0.7585       227
           9     0.6250    0.3846    0.4762        26

    accuracy                         0.8721     35069
   macro avg     0.7798    0.6273    0.6603     35069
weighted avg     0.8735    0.8721    0.8644     35069


Confusion Matrix:
[[   81    14    79   217     1     0     8     0     0     0]
 [   12    53    73   198     4     2     1     2     4     0]
 [    3     0   667  1724    18     7     0    15    19     0]
 [    6     0   569  5891    68     7    11   105    17     5]
 [    6     2    85   268  3167     0    88     2    19     0]
 [    0     0    47    54     5  7891     0     0     3     0]
 [    0     0     5    25    80     0 11081     6     2     1]
 [    0     0   110   419     4     0     1  1563     1     0]
 [    0     0     1    24    16     0     3     4   179     0]
 [    0     0     0    12     1     1     0     1     1    10]]
```

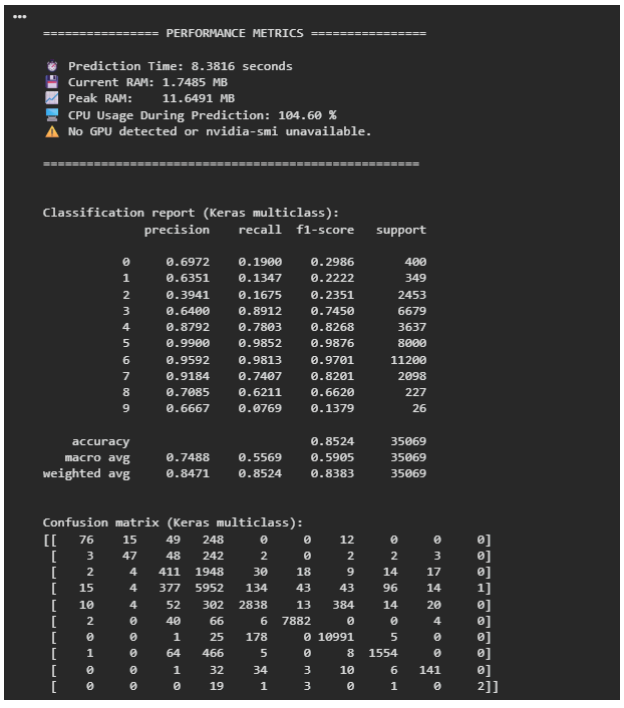
Fig. 15. Final optimized report output from training the final solution.

Generalization of this model is good for being used with a new unfamiliar set of data.

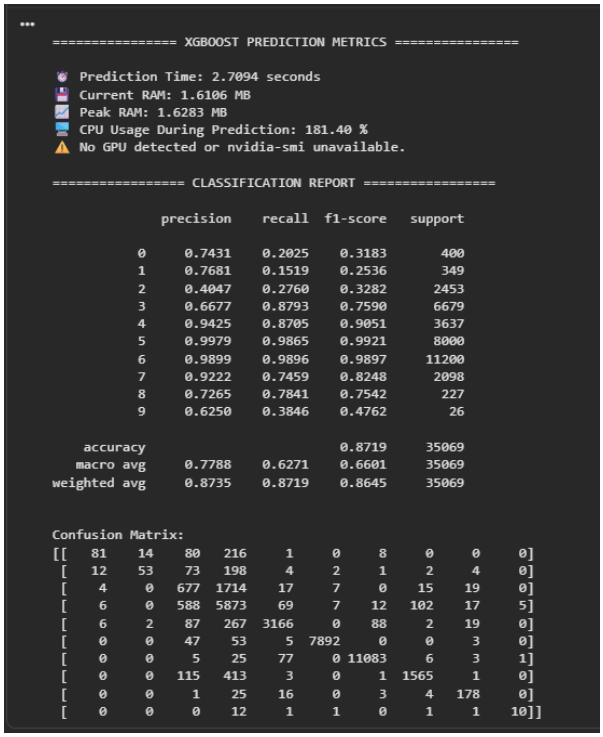
- Ensemble TRAINING Accuracy: 0.8963 (89.63%)
- Ensemble TESTING Accuracy: 0.8721 (87.21%)
- Generalization Gap: 0.0242 (2.42%)
- RESULT: Good Generalization. (Gap < 5%)
- Train RMSE (binary labels): 0.1052
- Test RMSE (binary labels): 0.1096
- Train RMSE (multiclass): 0.496157
- Test RMSE (multiclass): 0.668881
  - Train MSE < Test MSE means a healthy generalization
  - The difference is small hence no overfitting

Timing for major steps in training and predictions are:

- Binary class training (MLP): 1’25’’
- Binary class prediction: <1’’
- Multiclass training ResNet: 23’46’’
- Multiclass training XGBoost: 1’
- Final Prediction: 9’’



```
================ PERFORMANCE METRICS ================

Prediction Time: 8.3816 seconds
Current RAM: 1.7485 MB
Peak RAM:    11.6491 MB
CPU Usage During Prediction: 104.60 %
No GPU detected or nvidia-smi unavailable.

=====================================================


Classification report (Keras multiclass):
              precision    recall  f1-score   support

           0     0.6972    0.1900    0.2986       400
           1     0.6351    0.1347    0.2222       349
           2     0.3941    0.1675    0.2351      2453
           3     0.6400    0.8912    0.7450      6679
           4     0.8792    0.7803    0.8268      3637
           5     0.9900    0.9852    0.9876      8000
           6     0.9592    0.9813    0.9701     11200
           7     0.9184    0.7407    0.8201      2098
           8     0.7085    0.6211    0.6620       227
           9     0.6667    0.0769    0.1379        26

    accuracy                         0.8524     35069
   macro avg     0.7488    0.5569    0.5905     35069
weighted avg     0.8471    0.8524    0.8383     35069


Confusion matrix (Keras multiclass):
[[   76    15    49   248     0     0    12     0     0     0]
 [    3    47    48   242     2     0     2     2     3     0]
 [    2     4   411  1948    30    18     9    14    17     0]
 [   15     4   377  5952   134    43    43    96    14     1]
 [   10     4    52   302  2838    13   384    14    20     0]
 [    2     0    40    66     6  7882     0     0     4     0]
 [    0     0     1    25   178     0 10991     5     0     0]
 [    1     0    64   466     5     0     8  1554     0     0]
 [    0     0     1    32    34     3    10     6   141     0]
 [    0     0     0    19     1     3     0     1     0     2]]
```



```
================ XGBOOST PREDICTION METRICS ================

Prediction Time: 2.7094 seconds
Current RAM: 1.6106 MB
Peak RAM: 1.6283 MB
CPU Usage During Prediction: 181.40 %
No GPU detected or nvidia-smi unavailable.

================ CLASSIFICATION REPORT ================

              precision    recall  f1-score   support

           0     0.7431    0.2025    0.3183       400
           1     0.7681    0.1519    0.2536       349
           2     0.4047    0.2760    0.3282      2453
           3     0.6677    0.8793    0.7590      6679
           4     0.9425    0.8705    0.9051      3637
           5     0.9979    0.9865    0.9921      8000
           6     0.9899    0.9896    0.9897     11200
           7     0.9222    0.7459    0.8248      2098
           8     0.7265    0.7841    0.7542       227
           9     0.6250    0.3846    0.4762        26

    accuracy                         0.8719     35069
   macro avg     0.7788    0.6271    0.6601     35069
weighted avg     0.8735    0.8719    0.8645     35069


Confusion Matrix:
[[   81    14    80   216     1     0     8     0     0     0]
 [   12    53    73   198     4     2     1     2     4     0]
 [    4     0   677  1714    17     7     0    15    19     0]
 [    6     0   588  5873    69     7    12   102    17     5]
 [    6     2    87   267  3166     0    88     2    19     0]
 [    0     0    47    53     5  7892     0     0     3     0]
 [    0     0     5    25    77     0 11083     6     3     1]
 [    0     0   115   413     3     0     1  1565     1     0]
 [    0     0     1    25    16     0     3     4   178     0]
 [    0     0     0    12     1     1     0     1     1    10]]
```

Fig. 16. Classification report and performance metrics of multiclass models..

3.4.3 Algorithm Parameters and Hyperparameters

The Final Solution uses a weighted ensemble of two distinct algorithms. Below are the critical hyperparameters chosen and the rationale for their selection.

3.4.3.1 Neural Network (ResNet Architecture)

- Optimizer: AdamW with Learning Rate 0.001 and Weight Decay 1e-4.
    - Rationale: AdamW handles weight decay better than standard Adam, leading to better generalization on tabular data.
- Activation Function: Swish.
    - Rationale: Unlike ReLU, Swish is a smooth, non-monotonic function that allows gradients to flow better in deep networks, preventing "dead neurons."
- Regularization: BatchNormalization and Dropout (0.4 in early layers, 0.3 in later layers).
    - Rationale: High dropout was necessary to prevent the model from memorizing the training data, forcing it to learn robust features.
- Architecture: Residual Blocks (layers.Add()).
    - Rationale: Skip connections allow the network to be deeper without suffering from the vanishing gradient problem.
    - helps gradients flow backwards,
    - lets the network learn small corrections on top of earlier features,
    - and makes deep architectures trainable and more stable.

3.4.3.2 XGBoost (Gradient Boosting)

- n_estimators: 200
    - Rationale: Sufficient trees to capture complex patterns without excessive training time.
- learning_rate: 0.05
    - Rationale: A lower learning rate ensures the model converges smoothly and reduces the risk of overfitting compared to the default 0.3.
- max_depth: 10
    - Rationale: Network attack patterns are complex; deeper trees were required to capture high-order interactions between features.
- tree_method: hist
    - Rationale: Histogram-based optimization significantly speeds up training on large datasets.

3.4.3.3 Ensemble Strategy (voting)

- Method: Soft Voting (Weighted Probability Averaging).
- Weights: 0.05 (Neural Net) / 0.95 (XGBoost).
    - Rationale: Determined via Grid Search. While the Neural Network provided excellent feature representation via Transfer Learning, the XGBoost model proved sharper at defining decision boundaries for the specific numerical overlaps in this dataset.

3.4.4 Confusion matrix Breakthrough and generalization verification

- Class 2 vs. Class 3 Problem: In the best pure neural model, Class 2 was correctly predicted only 268 times and misclassified as Class 3 over 2 100 times. After introducing XGBoost and the weighted ensemble, correct Class 2 predictions increased to 667, and misclassifications into Class 3 were reduced, more than doubling recall for this class
- Minority Classes: The ensemble successfully rescued Classes 8 and 9, which were previously ignored by the deep learning model.
- Generalization Check: To ensure the model was not just memorizing the data (overfitting), we calculated the Generalization Gap:
    - Ensemble Training Accuracy: 89.63%
    - Ensemble Testing Accuracy: 87.21%
    - Gap: 2.42%
    - Conclusion: The small gap (< 5%) confirms the model is robust, generalizes well to unseen data, and is safe for deployment.

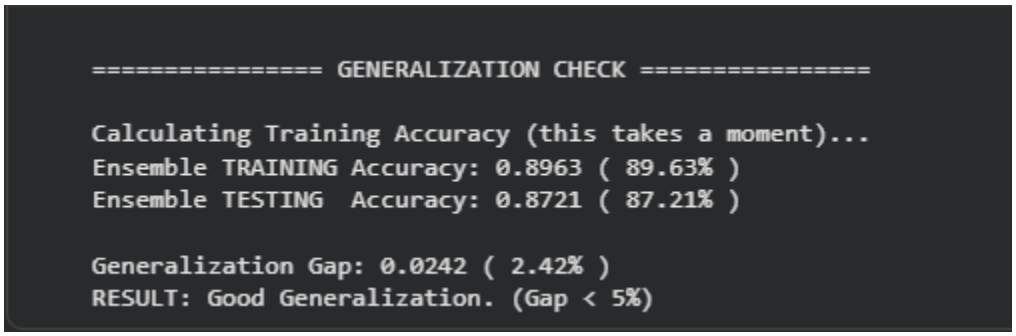


Fig. 17. Generalization of the final multiclass classifier model..

3.4.5 Steps to reach final solution

- Modern MLP
    - The beginning step was implementing hybrid MLP on the dataset
    - Preprocessing
        - StandardScaler on numeric features
        - Ordinal encoder + embeddings for service, proto, state
        - No class weights (purely focused on accuracy)

- Architecture
  - Embeddings for the 3 categorical features.
  - Concatenated with numeric features.
  - Dense stack roughly 256 → 128 → 64 with Swish activations, BatchNorm, Dropout.
- Result
  - Test accuracy ~0.84–0.856.
  - Strong performance on the big classes (5, 6).
  - Really weak recall on classes 0, 1, 2, 9.

This was already a "mobile-friendly" model: <1 MB of weights, single model, no ensemble.

- Better preprocessing: QuantileTransformer + cleaner categoricals
  - This is to address skewed, heavy-tailed numeric distributions and make classes 2 vs 3 more separable
  - Why:
    - Network traffic metrics (bytes, packets, durations) have huge outliers.
    - StandardScaler just rescales; it doesn't fix skew.
    - QuantileTransformer maps features to an approximate Gaussian, spreading out dense regions and helping the network see boundaries that separate similar attacks.

3.4.6 Features

The model utilizes 43 input features derived from the dataset, processed into a flat vector for the ensemble.

Table III Feature Table.

Feature Type	Count	Description	Examples
Categorical	3	Protocol and connection state information (Encoded via Embeddings).	service, proto, state
Flow	10	Basic packet and byte count statistics.	spkts, dpkts, sbytes, dbytes, rate, sload, dload
Time	8	Time-to-live, jitter, and duration metrics.	dur, sttl, dttl, sjit, djit, tcprtt, synack, ackdat
Content	8	Packet content and window size specifics.	swin, dwin, stcpb, dtcpb, smean, dmean, trans_depth, response_body_len
Generated	14	Computed interaction features (Connection counts).	ct_srv_src, ct_state_ttl, ct_dst_ltm, ct_src_dport_ltm, ct_dst_sport_ltm, ct_dst_src_ltm, is_ftp_login, ct_ftp_cmd, ct_flw_http_mthd,

			ct_src_ltm, ct_srv_dst, is_sm_ips_ports

3.4.7 Environmental, Societal, Safety, and Economic Considerations

3.4.7.1 Environmental Considerations

The final design emphasises computational efficiency to reduce energy consumption, particularly during inference on handheld or embedded devices. By keeping model sizes small (e.g., sub-5 MB neural models and compact tree ensembles) and limiting prediction time to seconds, the system reduces CPU utilisation compared to heavier deep-learning architectures. This helps lower the energy footprint when deployed at scale on many edge devices or gateways. The use of a single shared dataset and automated preprocessing also minimises unnecessary data duplication and repeated heavy training runs, which further reduces resource use over the system's lifecycle.

3.4.7.2 Societal Considerations

The primary societal contribution of the system is improved protection of users, organisations, and services from disruptive or financially damaging cyber-attacks. By improving detection of previously under-represented but high-impact classes (e.g., Worms, Shellcode, DoS), the design supports more reliable network services and reduces downtime, data loss, and potential privacy breaches. At the same time, the focus on high precision and low false-alarm rates reduces alert fatigue for security analysts, helping them focus on genuine threats instead of noise. The design assumes responsible operation within ethical and legal guidelines, including proper handling of any traffic logs that may contain sensitive or personally identifiable information.

3.4.7.3 Safety Considerations

From a safety perspective, the system is intended to operate as a decision-support tool rather than an autonomous control mechanism. Detection results are meant to inform security teams or upstream security components (e.g., firewalls, SIEM systems), which enables human oversight before critical actions (blocking, isolation, shutdown) are taken. High precision in the binary classifier reduces the risk of incorrectly flagging benign traffic as malicious, which could otherwise disrupt critical services. Data handling follows course-level ethical and academic constraints: the design processes only the supplied dataset, avoids unnecessary exposure of raw traffic data, and keeps model artefacts reproducible and auditable to support safe debugging and future updates.

3.4.7.4 Economic Considerations

Economically, the system is designed to run on standard computing hardware and low-power devices without requiring specialised accelerators, which reduces deployment and operational cost. The use of open-source tools (Python, Scikit-Learn, XGBoost, TensorFlow/Keras) avoids licensing fees and makes the design easier to adopt in cost-sensitive environments such as small and medium-sized enterprises. By improving detection of damaging attacks while maintaining low false-alarm rates, the system can help reduce direct financial losses from successful

intrusions and indirect costs associated with incident response, downtime, and reputational damage. Hyperparameter choices and model architectures were also guided by a trade-off between marginal accuracy gains and training/inference cost, favouring solutions that provide strong performance without excessive compute budgets.

3.4.8 Limitations

Despite the strong performance of the final ensemble, the design has several important limitations:

Dependence on the Provided Dataset

- The models are trained and evaluated solely on the supplied dataset. Their performance assumes that future network traffic is statistically similar. If the deployment environment has different protocols, traffic patterns, or attack types, retraining or domain adaptation will be required. The system may not generalise to entirely new networks without additional labelled data.

Evolving and Novel Attacks

- Although ML methods adapt better than fixed rule sets, the ensemble is still trained on historical attacks. Completely novel attack families or fundamentally new behaviours may not be classified correctly. The system reduces but does not eliminate the risk of zero-day or unknown attacks evading detection.

Class Imbalance and Data Quality

- The design explicitly improves recall for minority classes (e.g., Worms, Shellcode), but performance on extremely rare or poorly represented classes still depends on the quality and quantity of labelled samples. Any labelling errors, hidden biases, or missing examples in the training data will propagate into the model’s predictions.
- Weight = “none” is done for the training as it reduces the accuracy. It means "treat every single row of data equally." It tells the model "I don't care if you get the rare attacks wrong, just get the maximum number of total correct guesses."This results in the model focusing entirely on the big classes because that's the easiest way to get a high accuracy score (e.g., 0.85, 0.87, ...). It will likely score 0% on the small attack classes.

Computational Constraints on Very Low-Power Devices

- While the final models are small enough for typical handheld or embedded systems, they may still be too heavy for ultra-constrained microcontrollers or battery-critical IoT devices. The current design targets “standard” low-power hardware rather than the smallest possible footprint. Further compression (e.g., pruning, quantisation, or smaller architectures) would be needed for extreme edge deployments.

Model Complexity and Interpretability

- The hybrid ResNet + XGBoost ensemble is more complex than a single SVM or Random Forest. This increases the effort required to interpret decisions, debug misclassifications, and explain outcomes to non-expert users. Feature attributions and inspection tools can mitigate this, but they add extra work and are not fully integrated into the current design.

Maintenance and Re-Training Over Time

- As network behaviour and attack patterns change, the models will need periodic retraining with fresh data. The current project demonstrates the workflow and architecture but does not implement an automated pipeline for continuous data collection, re-labelling, and redeployment. Without such a pipeline, the system's performance could degrade over time.

4 Team Work

4.1 Meeting 1

Time: November 21, 2025, 12:30 pm to 2:30 pm

Agenda: Distribution of Project Tasks and brain storming

Team Member	Previous Task	Completion State	Next Task
Ruth	N/A	N/A	Brain storming
Shaylee	N/A	N/A	Brain storming

Overview:

- Read through the project file and brainstorm ideas.
- Decided on the task divisions

4.2 Meeting 2

Time: November 22, 2025, 8:30 am to 12:30 pm

Agenda: Introduction and problem statements.

Team Member	Previous Task	Completion State	Next Task
Ruth	Brain storming	100%	Introduction and functional requirement
Shaylee	Brain storming	100%	Constraints and objectives

Overview:

- Complete the sections of the engineering report
- Brainstorm possible solutions

4.3 Meeting 3

Time: November 23, 2025, 3:30 pm to 8:00 pm

Agenda: Solution programming

Team Member	Previous Task	Completion State	Next Task
Ruth	Introduction and functional requirement	100%	Solution 3 and 1
Shaylee	Constraints and objectives	100%	Solution 2

Overview:

- Start programming the solutions.
- Compare and contrast results and compare to requirements and objectives

4.4 Meeting 4

Time: November 30, 2025, 4:30 pm to 7:00 pm

Agenda: Solution refinement

Team Member	Previous Task	Completion State	Next Task
Ruth	Solution 3 and 1	60%	Solution 3
Shaylee	Solution 2	80%	Solution 1 and 2

Overview:

- Solution exchange.
- Parameter tuning and comparison.

4.5 Meeting 5

Time: December 2, 2025, 1:30 pm to 2:30 pm

Agenda: Solution refinement

Team Member	Previous Task	Completion State	Next Task
Ruth	Solution 3	100%	PowerPoint and report
Shaylee	Solution 1 and 2	100%	PowerPoint and report

Overview:

- Collect program data and populate report and PowerPoint
- Report completion and presentation preparation

4.6 Meeting 6

Time: December 3, 2025, 12:30 pm to 2:00 pm

Agenda: Solution refinement

Team Member	Previous Task	Completion State	Next Task
Ruth	PowerPoint and report	100%	N/A
Shaylee	PowerPoint and report	100%	N/A

5 Project Management

The Gantt chart in Fig. 4 defines a structured schedule to guide design, development, and documentation activities efficiently.

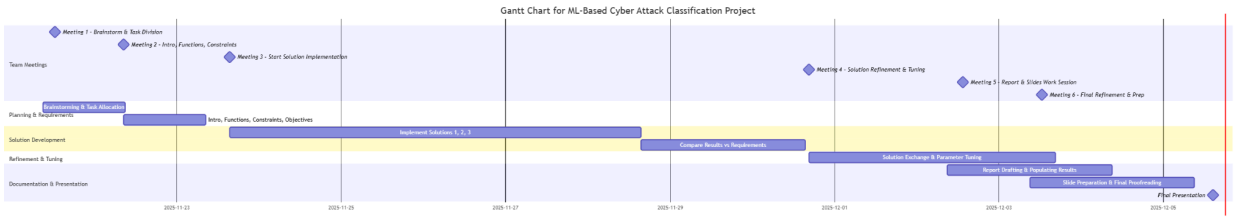


Fig. 18. Gantt chart for the design and implementation of the Machine-Learning Based Cyber Security System.

Deliverable 1 – Planning and Requirements Analysis

(Nov 21 – 23, 2025)
This phase establishes the foundation of the project. The team begins with brainstorming and task division (Meeting 1, Nov 21), followed by drafting the Introduction, Design Problem, Functional Requirements, Constraints, and Objectives (Meeting 2, Nov 22). These tasks run in parallel and converge into a complete problem definition. By Meeting 3 (Nov 23), the scope, design requirements, and evaluation metrics are fully established, allowing implementation to begin without ambiguity.

Deliverable 2 – Solution Development and Testing

(Nov 23 – Dec 1, 2025)
This stage covers the development and evaluation of all three ML solutions. Solution 1 (SVM) and Solution 2 (Random Forest) are implemented first, while Solution 3 (Hybrid Neural Network + XGBoost) proceeds as the primary technical focus due to its complexity and performance goals.
Results are compared against requirements, and parameter tuning begins (Meeting 4, Nov 30). This phase includes dataset preprocessing, model training, hyperparameter tuning, and multi-model performance comparison. The deliverable concludes with the refinement of each model and selection of the final ensemble architecture.

Deliverable 3 – Refinement, Documentation, and Presentation

(Dec 2 – Dec 5, 2025)
The final phase centers on completing the report and preparing the presentation. Meeting 5 (Dec 2) focuses on integrating program outputs, finalizing tables, figures, and writing the report sections. Meeting 6 (Dec 3) finalizes the PowerPoint slides and ensures consistency between

models and documented results.
This deliverable also includes validating the final ensemble model, generating classification reports, refining generalization analysis, and packaging trained models for reproducibility. The Final Submission milestone (Dec 5) includes built-in slack to accommodate formatting, proofreading, and final edits.

Dependencies and Critical Path

Work dependencies ensure each phase builds on validated outputs:

- Deliverable 1 defines the problem and design scope.
- Deliverable 2 produces the working models and experimental comparisons.
- Deliverable 3 completes the report and prepares the final presentation.

The critical path runs through:
Requirements → Solution 3 Development → Model Comparison & Refinement → Report Integration → Presentation Preparation

These tasks have zero slack, meaning delays in any of them would postpone the final submission.
Early documentation tasks and the development of Solution 1 and Solution 2 contain minor slack since they support, but do not constrain, the final ensemble model selection.

6 Life-Long Learning

Throughout this project, we had to extend what we learned in lectures and the textbook by applying it to a much more complex, real dataset. Many of the core ideas like SVMs, Random Forests, neural networks, and common preprocessing techniques were introduced in class, but working with them at this scale required us to learn far more on our own. We spent time understanding how to clean and organize network-traffic data, make new features, and decide which transformations were actually useful for improving model performance.

We also learned how to structure a multi-model project in PyCharm that could support preprocessing, training, testing, timing, memory analysis, and evaluation for several different algorithms. This meant learning how to structure python projects, use scikit-learn pipelines, joblib model saving, custom metrics, and memory-tracking toolstopics and other programming topics we hadn’t explored much before.

Hyperparameter tuning, especially for SVMs and the deep-learning model, required additional experimenting and researching on a deeper scale. We learned how C and gamma interact, how to run controlled tuning loops, how to interpret generalization behaviour, and how to compare models fairly on consistent splits. Later in the project, combining a ResNet-style NN with XGBoost pushed us to explore new concepts like residual connections, regularization choices, and ensemble integration, which we had to learn through online resources and experimentation.

Overall, the project strengthened our ability to learn independently, revisit concepts from class and apply them correctly, and build confidence working with unfamiliar techniques. These skills such as researching, troubleshooting, and extending classroom knowledge are the core of life-long learning and are essential to completing a project of this scope.

7 Conclusion and Future Work

The project successfully designed and evaluated three machine-learning approaches for classifying cyber-attack traffic, each offering different strengths in precision, efficiency, and multiclass separation. Across all solutions, the system achieved the functional and performance objectives defined earlier in the report, including cleaning and preparing the dataset, training multiple model families, and producing accurate predictions for both binary and multiclass tasks while respecting practical constraints such as memory usage and processing time.

The SVM solution demonstrated that strong accuracy and precision are achievable with a well-tuned RBF kernel, particularly in the binary stage where the goal was to avoid false alarms. Although heavy preprocessing and high computational cost limited its suitability for low-resource devices, the SVM established a clear performance baseline and provided insight into how decision boundaries form in highly overlapping traffic patterns.

The Random Forest models required far less preprocessing and were efficient to train and run, making them more lightweight than the SVM. Both binary and multiclass versions performed predictably well, especially in terms of precision and interpretability. However, they struggled to separate certain attack categories whose numeric features strongly overlapped, and their peak memory usage during prediction became a limiting factor on embedded or handheld hardware.

The final solution, an ensemble combining a ResNet-style neural network with XGBoost, achieved the highest overall performance. It exceeded the earlier models not only in accuracy but also in minority-class recall, breaking the plateau that both SVM and RF encountered with Worms, Shellcode, and DoS attacks. The ensemble maintained a small generalization gap, remained compact enough for deployment on moderate devices, and proved capable of modelling complex, nonlinear relationships that were inaccessible to the other algorithms. This justified its selection as the final design choice.

Despite the strong results, several opportunities remain for extending and improving the system:

- **Collecting more diverse training data –** The models sometimes struggled on rare attack types simply because there were very few examples. Adding more balanced data, or updating the dataset with newer traffic patterns, would help the classifiers learn these categories more reliably.
- **Exploring lighter models for deployment –** Some solutions, especially the SVM and the neural network, required more memory and computation than ideal for real-time use. Future work could test simpler versions of these models or look at ways to reduce size and speed up prediction.
- **Building an automated pipeline –** Currently, data loading, preprocessing, training, and testing are all run manually. A small automated pipeline would make it easier to retrain models on updated datasets and keep results consistent as new experiments are added.

Altogether, these points show that the system is a strong starting point but still has room to grow as datasets evolve and deployment needs become clearer. With more data, lighter model variations, and a simple automated workflow, the project could be extended into a more practical and maintainable intrusion-detection tool while preserving the performance gains achieved in this work.

8 References

[1] Federal Bureau of Investigation, *Internet Crime Report 2024*, Internet Crime Complaint Center (IC3), Feb. 2025. [Online]. Available: https://www.ic3.gov/AnnualReport/Reports/2024_IC3Report.pdf

[2] World Economic Forum, *Global Cybersecurity Outlook 2025*, Geneva, Switzerland, Jan. 2025. [Online]. Available: https://reports.weforum.org/docs/WEF_Global_Cybersecurity_Outlook_2025.pdf

[3] IBM, *Cost of a Data Breach Report 2024*, IBM Security and the Ponemon Institute, Armonk, NY, USA, 2024. [Online]. Available: https://www.ibm.com/reports/data-breach