Botnet Detection

Final Report for the Cybersecurity Course [2025/2026]

Gabriele Aprile Simone Tassi

December 2025

Contents

1 Overview

1.1 Objective

The rapid proliferation of connected devices has altered the volume and velocity of global network traffic, creating a fertile environment for cybercriminal operations. Central to this threat landscape is the *botnet*, a coordinated network of compromised nodes controlled by a Command and Control (C&C) server. By infecting thousands of insecure devices, attackers can disrupt services and compromise data with strong efficiency. These networks act as versatile delivery systems for a wide spectrum of attacks, ranging from Distributed Denial of Service (DDoS) campaigns to financial schemes like click fraud and mass spam distribution. The core challenge for security analysts, however, lies not just in stopping the attack, but in the early identification of the infected hosts that make up the network. Accurate classification of network traffic is therefore the first line of defence.

The objective of this work is to pursue the botnet classification task by using different Artificial Intelligence techniques, and to define which are the best strategies in order to enhance the overall security posture of modern networks against these sophisticated, distributed threats.

1.2 Literature and previous work

Historically, network intrusion detection relied on signature-based methods. These systems operated much like traditional antivirus software, comparing network packets against a database of known malicious byte sequences or "fingerprints" [7]. While effective against static, well-documented malware, this approach has effectively reached its operational limit. Modern botnets employ polymorphism, dynamic command-and-control (C&C) domains, and heavy traffic encryption to evade detection. In such scenarios, signature-based inspection fails because it searches for specific static patterns that the attacker has already obfuscated or hidden behind an encrypted layer.

This inability to detect zero-day or encrypted attacks forced a fundamental shift in the field: moving from packet inspection to **behavioural analysis**. Rather than analyzing the *content* of a packet (which may be unreadable), research now focuses on *how* devices communicate. This involves examining flow-level statistics (such as packet timing, volume, directionality, and periodicity) to identify anomalies. Within this framework, Machine Learning has become the primary engine for detection, treating network traffic analysis as a classification problem where the model learns the abstract behavioural patterns of an infection rather than its payload [1] [6].

When selecting a model for this task, the current literature presents a division in viewpoints. On one hand, Deep Learning approaches, such as Hybrid CNN-LSTM architectures, are frequently proposed for their ability to capture complex temporal dependencies in raw traffic data [5]. However, for flow-based analysis, where data is already aggregated into statistical feature, empirical benchmarks often favour traditional "shallow"

learning algorithms. Comparative studies demonstrate that ensemble methods, particularly Random Forests and optimized Decision Trees, frequently outperform complex neural networks in terms of training efficiency and interpretability, while maintaining competitive accuracy on benchmark datasets [2] [4].

Consequently, the critical challenge in this work is not just selecting the best algorithm, but ensuring the validity of the training environment. A recurring issue in botnet research is the reliance on synthetic data, which often lacks the noise and unpredictability of a production network, leading to models that overfit and fail in deployment. By training on mixed, real-world traffic captures rather than simulated environments, this work aims to evaluate how these machine learning classifiers perform when exposed to the genuine variability of legitimate background traffic.

1.3 Datasets

The reference dataset selected for this work is the **CTU-13 Dataset** [1] [3], a widely recognized benchmark in the field of botnet detection captured at the Czech Technical University (CTU) in 2011. Unlike purely synthetic datasets, CTU-13 captures real botnet traffic mixed with safe *normal* and *background* traffic, providing a realistic environment for evaluating machine learning classifiers.

The dataset consists of thirteen distinct "scenarios", each representing a specific botnet infection. In every scenario, a specific malware sample was executed in a controlled environment to generate malicious traffic, while normal and background traffic were simultaneously captured from the university network. These scenarios cover a diverse range of botnet behaviors and protocols, including **Neris** (IRC-based, engaging in DDoS and spam), **Rbot** (TCP/UDP-based scanning and DDoS), **Virut** (HTTP-based spam and data exfiltration), **Murlo**, **Menti**, **Sogou**, and **NSIS.ay**.

The raw data for each scenario originates from **PCAP (Packet Capture)** files, which recorded the complete network traffic during the infection window. To protect user privacy, the PCAP files containing normal and background traffic were truncated to remove payloads, while the captures containing only the botnet traffic remain complete.

For this analysis, the dataset's processed **bidirectional NetFlows** (specifically in `.binetflow` format) are employed (generated using the *Argus* auditing tool). The use of bidirectional flows is a critical enhancement over standard unidirectional NetFlows; it allows for the correct identification of client-server relationships and provides aggregated statistics (such as duration, total packets, and byte counts) for the entire communication session. This bidirectional perspective offers a richer feature set for machine learning algorithms, enabling more accurate discrimination between benign and malicious flow patterns.

[1] https://www.stratosphereips.org/datasets-ctu13; last accessed 12/12/2025

It is important to mention that the CTU-13 dataset is used in its **labeled** form, where each flow is annotated as benign or malicious. This explicit ground-truth labeling enables the formulation of the problem as a **supervised learning task**, allowing machine learning models to learn discriminative patterns between normal and botnet-generated network flows.

A crucial aspect of the CTU-13 dataset lies in the topology of the infections captured within the `.binetflow` files. Depending on the specific scenario, the malicious traffic originates either from a **single compromised node** (a unique IP address acting as a bot) or from **multiple infected nodes** operating simultaneously. This distinction is vital for classification tasks, as it tests the algorithm's ability to detect botnet patterns both in isolated instances and in coordinated, distributed environments.

To facilitate the analysis and referencing of these scenarios throughout this report, a standardized naming convention has been applied to each dataset. The label assigned to each scenario follows the format:

`[Index]-[CaptureID]-[BotName]-[Topology]`

where:

- **Index:** A sequential identifier (1 through 13) assigned for this report.
- **CaptureID:** The original ID assigned by the CTU researchers (e.g., 42, 43, etc.).
- **BotName:** The specific family of the malware employed (e.g., *Neris*, *Rbot*, *Virut*).
- **Topology:** The suffix `single` indicates the attack was generated by a single IP, while `multi` indicates the involvement of multiple infected nodes.

2 Methodology

2.1 Preprocessing

The subsequent sections delineate the comprehensive **preprocessing phases** applied to the datasets. Where specific steps pertain exclusively to a subset of the utilized algorithms, this is explicitly noted.
Effective preprocessing is crucial, as the quality of the input data is directly linked to the performance and reliability of the final detection models.

2.1.1 Data Cleaning

1. **Casting** of `sTos` and `dTos` features in type `Int64` to ensure proper compatibility in the following phases of preprocessing

Scenario Name	IRC	SPAM	CF	PS	DDoS	FF	P2P	HTTP
1-42-neris-single	✓	✓	✓					
2-43-neris-single	✓	✓	✓					
3-44-rbot-single	✓			✓				
4-45-rbot-single	✓				✓			
5-46-virut-single		✓		✓				✓
6-47-donbot-single				✓				
7-48-sogou-single								✓
8-49-murlo-single				✓				
9-50-neris-multi	✓	✓	✓	✓				
10-51-rbot-multi	✓				✓			
11-52-rbot-multi	✓				✓			
12-53-nsis-multi						✓	✓	
13-54-virut-single		✓		✓				✓

Table 1: Characteristics of the botnet scenarios

Abbreviation	Extended Form	Description
IRC	*Internet Relay Chat*	Communication protocol often used by botnets for Command and Control (C&C)
SPAM	*Spamming*	Activity of sending unsolicited messages in bulk
CF	*Click Fraud*	Fraud on advertising clicks
PS	*Port Scan*	Scanning of network ports to find vulnerabilities
DDoS	*Distributed Denial of Service*	Distributed denial of service attack, in which the botnet overloads a target with an enormous amount of traffic
FF	*Fast Flux*	Technique used to obfuscate the locations of malicious servers by rapidly changing Domain Name System (DNS) records
P2P	*Peer-to-Peer*	Decentralized communication model used by botnets
HTTP	*Hypertext Transfer Protocol*	Protocol used by botnets for Command and Control through web traffic

Helper to better understand the characteristics of Scenarios

2. **Checking and fixing labels**: in the `Dir` column, which indicates the direction of the `binetflow`, we identified some broken or incorrect labels. These were fixed and converted into a common terminology for better clarity and consistency. In particular:

- the `who` label, since it could not be correctly interpreted, led to the drop of the corresponding record
- `->`, `?>`, `<-`, `<?` were converted to `mono`, meaning "monodirectional flow"
- `<->`, `<?>` were converted to `bi`, meaning "bidirectional flow"

3. **Dropping duplicated rows**
4. **Dropping not useful columns**
   - `sTos` and `dTos` columns contained mostly 0 values across the entire datasets; non-zero values accounted for less than 0.2% of the total, making these features not useful for the analysis
   - the `StartTime` feature was not considered particularly useful, since it mainly provides absolute timing information, while the `Dur` (Duration) feature is more meaningful for analyzing attack intervals and frequencies
5. **Uniformed labels** into `1` (for botnet flows) and `0` (for background/safe flows)
6. *Only for SVM and MLP* | **Dropped rows with NaN** values (executed after 2.1.2)

2.1.2 Data Encoding

Some of the fields in the dataset required encoding in order to convert categorical data into a numerical format. Below are the two types of encoding adopted.

- **Label encoding**, used for low-cardinality columns, i.e. `Proto` (communication protocol used), `Dir` (binetflow direction), and `State` (a byproduct of the communication protocol)
- **Frequency encoding**, used for high-cardinality columns, where there are many different labels to encode: each category is replaced with the number of times it appears in the dataset. Columns encoded in this way are `SrcAddr` and `DstAddr` (the source and destination IP addresses of the data flow), and `Sport` and `Dport` (the source and destination ports of the data flow)

2.1.3 Data Balancing (*Only for SVM and MLP*)

Since *SVM* is particularly computationally expensive in terms of training time and extremely slow on large datasets, we limited the maximum number of training samples to make the process feasible. This limitation also helped mitigate the issue of heavily unbalanced data, which presents challenges for both SVM and MLP classifiers.

The majority class (*Background*) was **undersampled** to match the minority class (*Botnet*), resulting in a balanced dataset.

If `SVM_MAX_TRAIN_SAMPLES` is set, it ensures that the total dataset size does not exceed this value while maintaining a 50/50 class split.
In the SVM experiments conducted, this *limit was set to 200,000* samples.

2.1.4 Data Scaling (*Only for SVM and MLP*)

Since *SVM* is a *distance-based algorithm*, feature distances have a significant impact on its results and overall performance; for this reason, a feature scaling technique was applied, specifically `StandardScaler`. Feature scaling was also applied to the *MLP* classifier, as it improves general performance in the training phase.

2.1.5 Dimensionality Reduction - PCA (*Only for SVM*)

Dimensionality reduction is the process of reducing the number of input variables in a dataset while retaining the most important information. It helps improve model performance, reduces noise, and makes complex data easier to visualize and interpret.

Since our dataset does not contain an excessive number of features, dimensionality reduction techniques are not strictly necessary. However, in an attempt to improve SVM training time, we also conducted a sub-experiment using a dimensionality reduction technique called **Principal Component Analysis (PCA)**, with `n_components=0.95`, retaining 95% of the original feature variance in the transformed dataset.

2.2 Data exploration of the unbalanced datasets

One of the fundamentals of working with data, in order to analyze it correctly and extract insights that are not immediately apparent to the human eye, is to *know your data*.

After the data cleaning phase, we generated simple bar plots to illustrate the **(un)balance of classes** in the dataset. These images were produced during *Experiment 1*, but they provide a useful visual representation of one of the most important challenges in data analysis and cybersecurity analysis in particular: the availability of high-quality datasets, in sufficient quantity, and derived from real, rather than synthetic, scenarios.

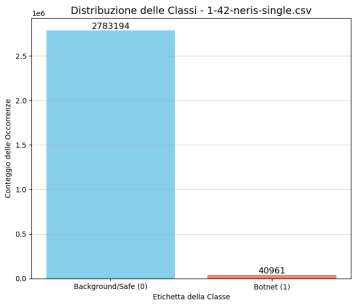
In this project, the situation is quite challenging due to the strong class imbalance, as can be seen in Figure 1.

2.3 Classification models

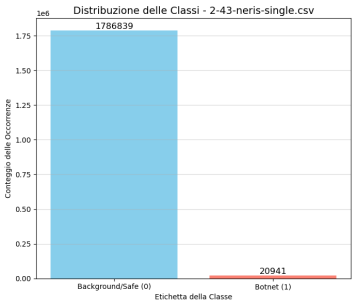
Various AI models are used in the experiments described in the following pages. In this section, we provide a general list of all the models used, along with the parameters applied in the experiments (unless otherwise specified, default settings were used).
All the models are **supervised learning models**. In the Experiments section, when referring to these models, we refer to the exact configurations described here.

The models were chosen primarily based on the existing literature review, which, as previously mentioned in subsection 1.2, identifies these methods as among the most effective for this use case.

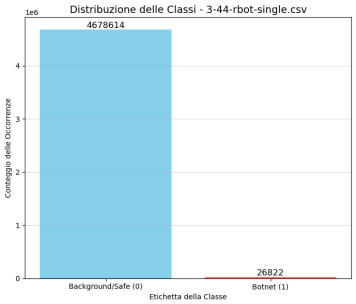
1. **RandomForestClassifier**



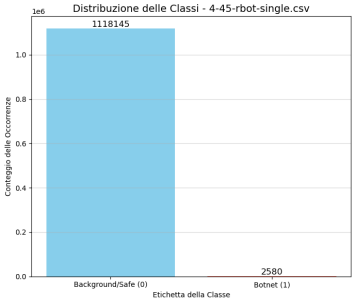
(a) 1.4504%



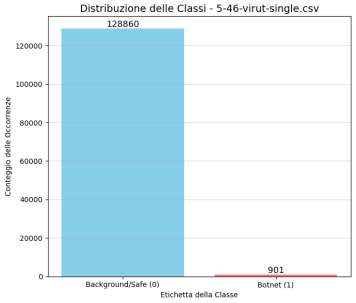
(b) 1.1584%



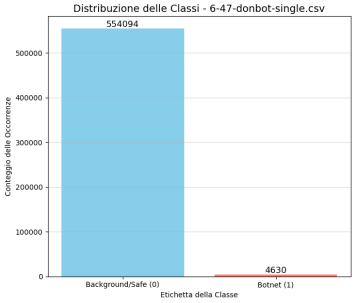
(c) 0.5700%



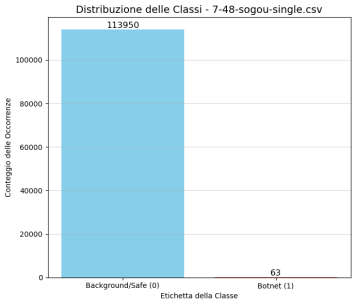
(d) 0.2302%



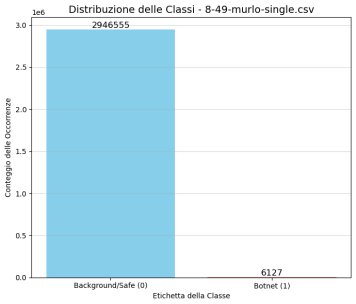
(e) 0.6944%



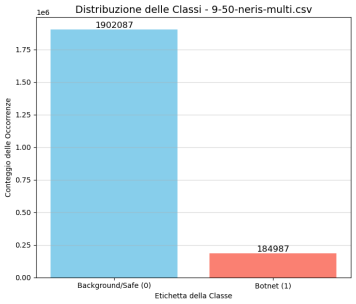
(f) 0.8287%



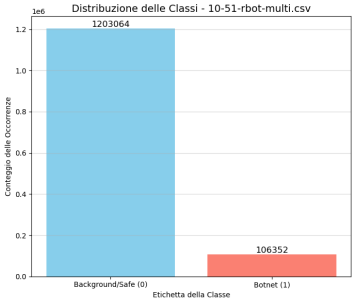
(g) 0.0553%



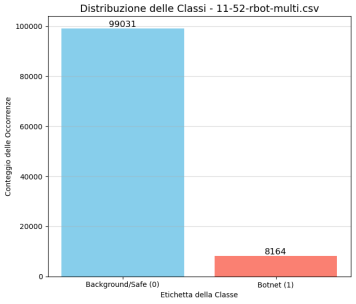
(h) 0.2075%



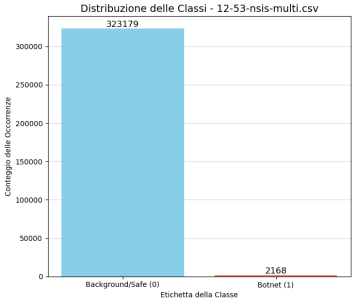
(i) 8.8635%



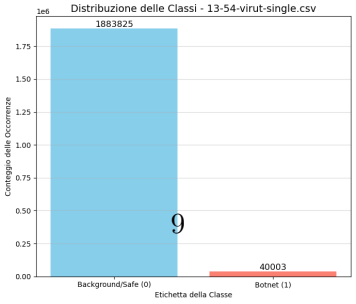
(j) 8.1221%



(k) 7.6160%



(l) 0.6664%



(m) 2.0793%

Figure 1: Visualization of class balance across the 13 datasets in *Experiment 1* after data cleaning, showing the percentage of rows labeled as Botnet in each dataset.

- **Type**: Random Forest (tree-ensamble technique)
- **Parameters**:

```
criterion='gini',
random_state=RANDOM_STATE,
n_jobs=-1
```

2. **DecisionTreeClassifier**
   - **Type**: Decision Tree
   - **Parameters**:

```
criterion='gini',
random_state=RANDOM_STATE,
n_jobs=-1
```

3. **SVC**
   - **Type**: C-Support Vector Classification (Support Vector Machine - SVM)
   - **Parameters**:

```
kernel='rbf',
C=1.0,
gamma='scale',
random_state=RANDOM_STATE,
cache_size=2000
```

4. **MLPClassifier**
   - **Type**: Multilayer perceptron (Neural Network)
   - **Variations**: used both in this configuration and in an alternative version with two hidden layers of sizes `(100, 50)`, as specified in the Experiments section
   - **Parameters**:

```
hidden_layer_sizes=(100),
activation='relu',
solver='adam',
max_iter=1000,
learning_rate_init=0.001,
random_state=RANDOM_STATE,
early_stopping=True
```

2.4 Evaluation of the models

We evaluated and compared the results using multiple sources of support, including logs, various graphs, and reports containing different metrics. In particular, our considerations are based on:

- **Graphs**
  - **Confusion Matrix**, a table showing the number of correct and incorrect predictions for each class, which helps to visualize the performance of the classifier and identify specific misclassifications
  - **Feature Importance**, a measure indicating the impact of each feature on the model's predictions, helping to understand which features contribute most to the decision-making process
  - **Label count distribution**, as mentioned in subsection 2.2 and produced during *Experiment 1*, used to obtain information about the class balance in the dataset
- **Classification Report**, which summarizes the key information of the test for each class, including:
  - **Precision**, defined as
$$\frac{TP}{TP + FP}$$
  - **Recall** (True Positive Rate), defined as
$$\frac{TP}{TP + FN}$$
  - **F1-score**, defined as the harmonic mean between precision and recall
$$2 * \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}}$$
  - **Support**, the number of rows in the test dataset corresponding to each label
- **Other** infos, e.g.
  - Model, datasets used in train and test, number of training samples
  - **Accuracy**, defined as
$$\frac{TP + TN}{TP + FN + TN + FP}$$
    (usually considered a good measure only if there is an equal number of samples for each class)

2.5 Other notes

To ensure uniformity and reproducibility of the experiments, all algorithms requiring a `random_state` seed were initialized with the **seed 42** (`RANDOM_STATE = 42`).

When needed, the `train_test_split` ratio was fixed at **0.2** (80% training set, 20% test set), always **stratified** according to the class labels to maintain the same distribution of classes in both training and test sets. This is particularly important given the strong class imbalance, as it helps preserve the representativeness of each class.

3 Experiment 1: Baseline Assessment

3.1 Overview

This experiment, the first and most elementary of the three, evaluates the internal consistency and performance of machine learning classifiers on individual datasets. It is conducted by **splitting each dataset into training and test** sets using the same procedure, and it's run independently for all 13 scenarios using the *RandomForestClassifier*, as described in subsection 2.3.

The primary objective of this experiment is to **establish a performance baseline** by assessing how well the model performs within each dataset under controlled and consistent conditions.

3.2 Results

As can be observed from the results reported in Table 2, the *Random Forest* technique **performs well** on the available datasets. None of the reported metrics (precision, recall, or F1-score) for either the Botnet or the Background/Safe flow classes falls below 80%. However, the presence of several results equal to 1.00 could be a potential concern as it may indicate a tendency of the algorithm to learn the training data too closely, leading to possible **overfitting**.
Training times for Random Forest are satisfactory: from this perspective, it can be well suited for continuous real-time deployment in a realistic operational environment.

4 Experiment 2: Cross-Dataset Generalization

4.1 Overview

This experiment **evaluates the generalization capabilities and robustness** of machine learning classifiers, enforcing a strict separation of environments by **using different capture files for training and testing**, instead of splitting a single dataset as done in Experiment 1 (section 3).

The primary objective is to assess how well detection algorithms perform when applied to network traffic that differs from their training distribution. The experiment is run starting from a matrix of test scenarios that vary the relationship between the training and the test sets based on two key dimensions:

- **Attack typology:** Evaluating the impact of changing and preserving the specific type of attack (e.g. spam, DDoS, port scanning, ...)
- **Infection scale:** Analyzing the transition between datasets containing single infected hosts and those containing multiple infected hosts.

	Background/Safe			Botnet		
Dataset	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
`1-42-neris-single`	1.00	1.00	1.00	1.00	1.00	1.00
`2-43-neris-single`	1.00	1.00	1.00	1.00	1.00	1.00
`3-44-rbot-single`	1.00	1.00	1.00	1.00	1.00	1.00
`4-45-rbot-single`	1.00	1.00	1.00	1.00	0.81	0.90
`5-46-virut-single`	1.00	1.00	1.00	1.00	0.99	0.99
`6-47-donbot-single`	1.00	1.00	1.00	1.00	1.00	1.00
`7-48-sogou-single`	1.00	1.00	1.00	1.00	0.85	0.92
`8-49-murlo-single`	1.00	1.00	1.00	1.00	0.99	1.00
`9-50-neris-multi`	1.00	1.00	1.00	1.00	1.00	1.00
`10-51-rbot-multi`	0.99	1.00	0.99	1.00	0.86	0.93
`11-52-rbot-multi`	1.00	1.00	1.00	1.00	0.97	0.99
`12-53-nsis-multi`	1.00	1.00	1.00	1.00	0.97	0.98
`13-54-virut-single`	1.00	1.00	1.00	1.00	1.00	1.00

Table 2: Performance Metrics - Random Forest - 80/20 Split (Experiment 1)

By systematically **pairing different subsets of the CTU-13 dataset**, this experiment simulates real-world deployment challenges where a detection system must identify malicious behaviours in unseen network environments. While the investigation was initially prioritized around the *Random Forest* algorithm (as suggested by the literature in 1.2), the observation of non-optimal results in specific scenarios necessitated a wider architectural comparison. Consequently the experiment relies on **four distinct algorithms**, *Decision Trees*, *Random Forest*, *Support Vector Machine (SVM)* and *Multi-Layer Perceptrons (MLP)*, in order to determine their respective abilities to learn abstract malicious patterns rather than memorizing specific artifacts of a single network capture.

4.2 Datasets used

For this experiment both `single` and `multi` datasets (1.3) are employed in different combinations. The configuration matrix, detailed in Table 3, isolates specific variables, such as the botnet family or the attack type, to determine which factors most significantly influence model generalization.

By varying the source and the destination of the learning process, the experiment assesses whether the classifiers are learning the characteristics of network anomalies or just overfitting to the traffic patterns of a single dataset.

Test ID	Training Set	Testing Set	Relationship & Objective
0	`1-42-neris-single`	`2-43-neris-single`	**Baseline:** Same botnet, same spamming behavior. Single-to-Single infection.
1	`3-44-rbot-single`	`4-45-rbot-single`	**Intra-Family Variance:** Same botnet family, but different attack types (Port Scan vs. DDoS).
2	`5-46-virut-single`	`6-47-donbot-single`	**Behavioral Similarity:** Different botnet families, but both executing Port Scan attack.
3	`1-42-neris-single`	`8-49-murlo-single`	**Disjoint:** Different botnet families and different attack types (Spam vs. Port Scan).
4	`1-42-neris-single`	`10-51-rbot-multi`	**Scale & Family:** Single-host training (Spam) vs. Multi-host testing (DDoS).
5	`10-51-rbot-multi`	`4-45-rbot-single`	**Downscaling:** Multi-host training vs. Single-host testing within the same botnet family.
6	`10-51-rbot-multi`	`6-47-donbot-single`	**Complex Generalization:** Multi-host DDoS training vs. Single-host Port Scan testing (different families).
7	`10-51-rbot-multi`	`11-52-rbot-multi`	**Scale Consistency:** Multi-host to Multi-host transfer within the same botnet family.
8	`9-50-neris-multi`	`13-54-virut-single`	**Cross-Family Scale:** Multi-host Neris training vs. Single-host Virut testing.

Table 3: Dataset Pairs for Cross-Dataset Generalization (Experiment 2)

4.3 Results

4.3.1 Tree-Based Models

The tree-based architectures demonstrated the **highest robustness** across the majority of test scenarios. While the single *Decision Tree* (DT) achieves high precision in baseline scenarios (e.g., Test 0: Neris → Neris), it exhibits symptoms of overfitting, failing to generalize in complex transfer tasks. In contrast, the *Random Forest* (**RF**) algorithm consistently **outperforms the DT**, particularly in terms of Recall (Table 4). The ensemble nature of RF, which aggregates predictions from multiple decorrelated trees, appears to mitigate the variance inherent in single-capture training sets, allowing the **model to learn more abstract signatures** of malicious behavior rather than memorizing specific artifacts of the training file.

Test ID	Scenario		Decision Tree (DT)				Random Forest (RF)			
	Train	Test	*Background*		*Botnet*		*Background*		*Botnet*	
			Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
0	Neris	Neris	1.00	1.00	1.00	0.97	1.00	1.00	1.00	0.99
1	Rbot	Rbot	1.00	1.00	0.90	0.11	1.00	1.00	1.00	0.00
2	Virut	Donbot	1.00	1.00	0.86	0.99	1.00	1.00	1.00	0.99
3	Neris	Murlo	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98
4	Neris	Rbot (M)	0.93	1.00	0.95	0.09	0.93	1.00	0.95	0.09
5	Rbot (M)	Rbot	1.00	1.00	0.49	0.11	1.00	1.00	1.00	0.34
6	Rbot (M)	Donbot	0.99	1.00	0.60	0.00	0.99	1.00	1.00	0.00
7	Neris (M)	Virut	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
8	Rbot (M)	Rbot (M)	0.93	1.00	1.00	0.08	1.00	1.00	1.00	0.98

Table 4: Tree-Based Models Results (Experiment 2)

4.3.2 Support Vector Machines

The Support Vector Machine (SVM) classifiers yields the **poorest performance** metrics across the experimental matrix. In high dimensional network flow data, the separation margin between "Background" and "Botnet" classes is rarely linear. While the RBF kernel was employed to handle non-linearity, the model **struggles to construct a decision boundary** that remains valid across different network environments. The application of Principal Component Analysis (**PCA**) for dimensionality reduction **doesn't yield a significant improvement** in stability.

The practical viability of SVM for this task is low. The algorithm has a prohibitive training cost which necessitates down-sampling the datasets significantly (still resulting in a time consuming training process, compared to the other models). Despite this high computational investment, the overall performance is negligible.

Test ID	Scenario		SVM (No PCA)				SVM (with PCA)			
	Train	Test	*Background*		*Botnet*		*Background*		*Botnet*	
			Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
0	Neris	Neris	1.00	0.99	0.46	0.83	1.00	0.99	0.48	0.91
1	Rbot	Rbot	1.00	0.96	0.03	0.46	1.00	0.95	0.02	0.46
2	Virut	Donbot	1.00	0.87	0.06	0.99	1.00	0.87	0.06	0.99
3	Neris	Murlo	1.00	0.98	0.08	0.95	1.00	0.98	0.08	0.95
4	Neris	Rbot (M)	0.92	0.99	0.00	0.00	0.92	0.99	0.00	0.00
5	Rbot (M)	Rbot	1.00	1.00	0.11	0.08	1.00	0.99	0.02	0.08
6	Rbot (M)	Donbot	0.99	1.00	0.00	0.00	0.99	0.99	0.00	0.00
7	Neris (M)	Virut	1.00	0.82	0.11	1.00	1.00	0.82	0.11	1.00
8	Rbot (M)	Rbot (M)	0.93	1.00	0.68	0.06	0.93	0.97	0.16	0.07

Table 5: SVM Results (Experiment 2)

4.3.3 Multi-Layer Perceptrons

The *neural network models (MLP)* occupy a middle ground, **outperforming SVMs** but **failing to match the generalization capability of Random Forests**. Increasing the depth from a single hidden layer to two hidden layers provided marginal improvements in complex scenarios (e.g., Test 3 and Test 8 in Table 6). This suggests that while a deeper architecture allows for a higher level of feature abstraction, the sparsity of the training data limits the network's ability to learn robust representations. The MLP models exhibit a tendency to overfit the training distribution, leading to high training accuracy but sharp degradation when applied to unseen test files.

The application of Neural Networks faces the "small data" constraint. While the CTU-13 dataset contains millions of packets, the number of distinct flows (the unit of classification) representing botnet behavior is relatively small (often $< 5{,}000$ samples - see Figure 1). Neural approaches typically require massive datasets to converge on generalized weights. Moreover, *the extreme class imbalance can severely inhibit the gradient descent process*, biasing the network towards predicting the safe class to minimize global loss.

Test ID	**Scenario**		**MLP (Single Layer)**				**MLP (2 Layers)**			
	Train	**Test**	*Background*		*Botnet*		*Background*		*Botnet*	
			Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
0	Neris	Neris	1.00	1.00	0.98	0.97	1.00	1.00	0.99	1.00
1	Rbot	Rbot	1.00	1.00	0.19	0.48	1.00	1.00	0.01	0.00
2	Virut	Donbot	1.00	0.85	0.05	0.99	1.00	0.87	0.06	0.99
3	Neris	Murlo	1.00	1.00	0.78	0.96	1.00	1.00	0.93	1.00
4	Neris	Rbot (M)	0.92	1.00	0.04	0.00	0.92	1.00	0.05	0.00
5	Rbot (M)	Rbot	1.00	0.99	0.03	0.08	1.00	1.00	0.22	0.11
6	Rbot (M)	Donbot	0.99	0.99	0.00	0.00	0.99	1.00	0.03	0.00
7	Neris (M)	Virut	1.00	0.99	0.65	1.00	1.00	1.00	0.90	1.00
8	Rbot (M)	Rbot (M)	0.93	0.98	0.18	0.06	0.93	1.00	0.79	0.06

Table 6: Multi-Layer Perceptrons Results (Experiment 2)

4.3.4 Systematic errors

A distinct pattern of failure emerged across all model architectures in Tests 1, 4, 5, and 6. In these specific scenarios, the classifiers suffered from either critically low Recall (inability to detect the attack) or unstable Precision (high false alarm rates). The universality of these errors, affecting all models, suggests the issue lies within the data characteristics rather than the algorithmic approach.

- **Behavioral Disjointedness (Tests 1 & 6)**: In Test 1 (Rbot $\rightarrow$ Rbot) and Test 6 (Rbot $\rightarrow$ Donbot), the models exhibit near-zero Recall. This indicates that the

traffic profile of the botnet in the training set is effectively invisible in the testing set. Even in Test 1, where both sets contain the *Rbot* family, but the specific malicious actions differ (Port Scan vs DDoS). Consequently, the models learn to recognize specific actions rather than a generic "malicious" signature. When those specific actions are absent in the test set, the botnet traffic looks indistinguishable from background activity to the model.

- **Scale and Context Mismatch (Test 4)**: Test 4 involves training on a single infected host (Neris) and testing on a network with multiple infected hosts (Rbot). The models achieve extremely low recall. This suggests that it completely misses the noisier attack patterns generated by the multi-bot Rbot network. Probably, the features defining the "Multi" environment are absent in the "Single" training environment, leading to a failure to generalize.
- **Ambiguity and Feature Overlap (Test 5)**: In this scenario (Rbot Multi → Rbot Single), the Random Forest model manages the highest Recall with respect to the other models, with a high precision rate (1.00). This implies that when the model does classify a flow as malicious, it is correct 100% of the time. The low Recall, however, indicates that the model is failing to identify a substantial portion of the actual malicious Rbot traffic in the testing set. This phenomenon suggests that the model, trained on the complex multi-host dataset, learned a very strict signature of Rbot traffic. When faced with the simpler single-host dataset, only the most distinct and aggressive flow patterns are classified as Rbot. The majority of the Rbot traffic in the test set, likely including less intense flows, is conservatively classified as Background, resulting in high precision but poor detection coverage.

4.3.5 Expected and Observed Behavior

Expected Performance

The models perform exceptionally well in scenarios where the training and testing data shared concrete traffic behaviors, such as Test 0 (Neris → Neris) and Test 3 (Neris → Murlo). This success was expected because the underlying network flows were consistent. It was also expected, but interesting to notice, that models perform well on Test 2, where train and test datasets belong to different botnet families but execute the same Port Scan attack.

Expected Failures

The failure in Test 6 (Rbot → Donbot) was a predictable outcome. The training set consists of Rbot traffic (DDoS), while the test set contains Donbot traffic (Port Scan). These two behaviors generate fundamentally different feature footprints: DDoS creates bursts of short, high-frequency packets, while spam creates longer, established TCP connections. Since the models never see "Port Scan features" during training, the Donbot traffic was espected to be classified as "Background", resulting in the observed breakdown in detection.

Unexpected Findings

The most significant anomaly is the failure of Test 1 (Rbot → Rbot). The hypothesis made was that training and testing on the same malware family would guarantee a baseline level of success. The observed failure (Recall $<$ 0.12 across all models) contradicts the assumption that a "Botnet Family" label implies a consistent traffic signature. Therefore, a model trained on a "sleeping" Rbot is useless against an "attacking" Rbot, highlighting that **effective detection requires training on behavioral classes** (e.g., "Port Scanning," "DDoS") **rather than malware family** names.

5 Experiment 3: Multi-Class Identification

5.1 Overview

The third phase of the study focuses on evaluating the robustness of the **Random Forest** (1) detection model against a diverse set of malicious network traffic.

To achieve this, **multiple independent datasets** (each corresponding to a unique botnet family) are consolidated into a single comprehensive dataset. This merging process introduces network flow records characterized by varied behavioral features and protocols from different sources, challenging the classifier with a **broader spectrum of patterns**. Training and test sets are extracted through the same `train_test_split()` method usage described in subsection 2.5.

Working in this way, the classification objective changes. Unlike the other experiments, where the tasks are binary (botnet vs background/safe), the use of combined datasets establishes a **multi-class classification** problem. The machine learning model is required to simultaneously distinguish between multiple specific botnet labels, in addition to identifying normal background traffic. This setup tests the model's capability to generalize and categorize distinct classes of malicious activities within a single environment.

5.2 Results

Experiment 3 demonstrates the RF model's performance when identifying distinct botnet families within a heterogeneous dataset. The results, summarized in Table 7, show a high overall *accuracy* which strongly depends on the high cardinality of background traffic, but they also present high *recall* values, that indicate the model's ability to pursue correct classification despite the severe class imbalance.

A more detailed analysis on the **precision**, **recall** and **F1-score** for each specific botnet class reveals trends related to dataset composition:

1. **Impact of low cardinality**: The model's performance shows a noticeable decrease when classifying botnet families with a significantly lower number of flow records

compared to other classes. For instance:

- In **Mix 2** test, while *Neris* and *Rbot* maintain good scores, *Virut* shows reduced performance, due to its low sample presence in the dataset (40,961 botnet flows in `1-Neris` vs 901 botnet flows in `5-Virut`).
- In **Mix 4** test, where *Sogou* achieves perfect precision (1.00) but a low recall (0.62), it's clear how model rarely mislabels background and other botnet's traffic, but it fails to identify a portion of *Sogou* traffic, likely due to insufficient training data (26,822 botnet flows in `3-Rbot` vs 63 botnet flows in `7-Sogou`).

2. **Classification of Similar Botnet Families**: **Mix 7** test employs two different *Rbot* datasets performing the same type of attack. The model struggles to separate the two instances of the Rbot botnet family: this is evidenced by the poor performance metrics for the second Rbot class (11), with a very low **recall** and **F1-score**. This difficulty underlines that the behavioral features of the two *Rbot* instances are too similar for the classifier to distinguish between them as separate classes. The model assigns the flow records to the more represented *Rbot* class (106,352 botnet flows in `10-Rbot` vs 8,164 botnet flows in `11-Rbot`).

In summary, the model maintains good performance in discriminating between malicious and normal traffic, and between distinct and well represented botnet families. However, it becomes more difficult to perform the classification when botnet families have limited support in the training set or when required to differentiate between two instances of the same botnet with really similar traffic patterns.

Table 7: Multi-Class Classification Results on Merged Datasets (Experiment 3)

Dataset Composition	Class	Precision	Recall	F1-Score

Mix 1:
`1-42-neris-single`
`3-44-rbot-single`
	Background	1.00	1.00	1.00

	Neris	1.00	1.00	1.00
	Rbot	1.00	1.00	1.00

Mix 2:
`1-42-neris-single`
`3-44-rbot-single`
`5-46-virut-single`
	Background	1.00	1.00	1.00

	Neris	1.00	1.00	1.00
	Rbot	1.00	1.00	1.00
	Virut	0.93	0.82	0.87

Mix 3:
`1-42-neris-single`
`2-43-neris-single`
`3-44-rbot-single`
`5-46-virut-single`
	Background	1.00	1.00	1.00

	Neris (1)	0.96	0.96	0.96
	Neris (2)	0.92	0.93	0.92
	Rbot	1.00	1.00	1.00
	Virut	0.81	0.69	0.74

Continued on next page

Table 7 – continued from previous page

Dataset Composition	Class	Precision	Recall	F1-Score
Mix 4: 1-42-neris-single 3-44-rbot-single 5-46-virut-single 7-48-sogou-single 8-49-murlo-single	Background	1.00	1.00	1.00
	Neris	0.99	1.00	0.99
	Rbot	1.00	1.00	1.00
	Murlo	1.00	0.99	1.00
	Virut	0.88	0.71	0.78
	Sogou	1.00	0.62	0.76
Mix 5: 10-51-rbot-multi 12-53-nsis-multi	Background	1.00	1.00	1.00
	Rbot	1.00	0.94	0.97
	Nsis	1.00	0.98	0.99
Mix 6: 9-50-neris-multi 10-51-rbot-multi 12-53-nsis-multi	Background	1.00	1.00	1.00
	Neris	1.00	1.00	1.00
	Rbot	1.00	1.00	1.00
	Nsis	0.99	0.91	0.95
Mix 7: 9-50-neris-multi 10-51-rbot-multi 11-52-rbot-multi 12-53-nsis-multi	Background	1.00	1.00	1.00
	Neris	1.00	1.00	1.00
	Rbot (10)	0.93	1.00	0.97
	Nsis	1.00	0.93	0.96
	Rbot (11)	0.95	0.09	0.16
Mix 8: 1-42-neris-single 13-54-virut-single 10-51-rbot-multi 12-53-nsis-multi	Background	1.00	1.00	1.00
	Rbot	1.00	1.00	1.00
	Nsis	0.99	0.92	0.96
	Neris	0.93	0.95	0.94
	Virut	0.94	0.92	0.93
Mix 9: 1-42-neris-single 3-44-rbot-single 10-51-rbot-multi 12-53-nsis-multi	Background	1.00	1.00	1.00
	Neris	1.00	1.00	1.00
	Rbot (3)	1.00	1.00	1.00
	Rbot (10)	1.00	1.00	1.00
	Nsis	0.99	0.91	0.95

6 Conclusions

This study evaluated machine learning algorithms for botnet detection across various experimental phases using the CTU-13 datasets. The findings offer critical insights into performance, methodological dependencies, and inherent limitations in this domain.

The *Random Forest Classifier* consistently emerged as the best performing model overall, demonstrating high accuracy and strong individual class metrics. It represents the best trade-off between detection performance and training time, making it suitable for

potential simple real time deployment scenarios. However, the exceptional results in stratified split scenarios raise suspicion of potential overfitting, which necessitates cautious interpretation of the absolute numerical outcomes.

Other tested algorithms generally yielded lower performance, suggesting that more extensive hyperparameter tuning (which was beyond the scope of this comparative evaluation) is required to maximize their effectiveness and remains a key direction for future work.

A central insight is the impact of the *data split strategy on model performance*. In Experiment 1, the dataset is directly split into training and test sets, while in Experiment 3, although addressing a different objective (multiclass classification), the datasets are first aggregated and then split. The splitting process is always performed in a stratified manner, based on class labels, and the overall best results could be observed in these two experiments. The stratified split method contrasts with the cross-scenario (where the training and test sets were drawn entirely from different scenarios) evaluation of Experiment 2 which results in worse outcomes. This performance disparity suggests that machine learning algorithms generalize more effectively when tested on data that exhibits partially known behavioral patterns, a commonality maintained when testing data is structurally similar to the training data. This observation is consistent with trends discussed previously in the analysis section (subsubsection 4.3.5).

A persistent and critical challenge across all experimental phases was the severe *class imbalance* within the datasets, as illustrated in Figure 1. The overwhelming dominance of Background traffic is a fundamental limitation in cybersecurity research. The limited quantity and inherent imbalance of high-quality, real-world derived network flow data significantly constrain the generalization capabilities of machine learning models. Addressing the availability of diverse, voluminous, and balanced datasets remains essential for advancing the robustness of botnet detection systems.

References

[1] Khalid Alissa, Tahir Alyas, Kashif Zafar, Qaiser Abbas, Nadia Tabassum, and Shadman Sakib. Botnet attack detection in iot using machine learning. *Computational Intelligence and Neuroscience*, 2022(1):4515642, 2022. doi: https://doi.org/10.1155/2022/4515642. URL https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/4515642.

[2] Mustafa Alshamkhany, Wisam Alshamkhany, Mohamed Mansour, Mueez Khan, Salam Dhou, and Fadi Aloul. Botnet attack detection using machine learning. In *2020 14th International Conference on Innovations in Information Technology (IIT)*, pages 203–208, 2020. doi: 10.1109/IIT50501.2020.9299061.

[3] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. An empirical comparison of botnet detection methods. *Computers & Security*, 45:100–123, 2014. doi: 10.1016/j.cose.2014.05.011.

[4] V. Kanimozhi and T. P. Jacob. Performance evaluation of botnet detection using machine learning techniques. *International Journal of Electrical and Computer Engineering (IJECE)*, 13(6):6827–6835, 2023. doi: 10.11591/ijece.v13i6.pp6827-6835.

[5] A. Karthick Kumar, S. Rathnamala, T. Vijayashanthi, M. Prabhananthakumar, Alavikunhu Panthakkan, Shadi Atalla, and Wathiq Mansoor. Enhanced hybrid deep learning approach for botnet attacks detection in iot environment. In *2024 7th International Conference on Signal Processing and Information Security (ICSPIS)*, pages 1–6, 2024. doi: 10.1109/ICSPIS63676.2024.10812621.

[6] Celil OKUR and Murat DENER. Detecting iot botnet attacks using machine learning methods. In *2020 International Conference on Information Security and Cryptology (ISCTURKEY)*, pages 31–37, 2020. doi: 10.1109/ISCTURKEY51113.2020.9307994. URL https://ieeexplore.ieee.org/document/9307994.

[7] Ying Xing, Hui Shu, Hao Zhao, Dannong Li, and Li Guo. Survey on botnet detection techniques: Classification, methods, and evaluation. *Mathematical Problems in Engineering*, 2021(1):6640499, 2021. doi: https://doi.org/10.1155/2021/6640499. URL https://onlinelibrary.wiley.com/doi/abs/10.1155/2021/6640499.