Detection of malicious cyber attacks in Internet of Vehicles

Submitted in partial fulfilment of the requirements

of the degree of

Bachelor of Technology (B.Tech)

by

Thallada Meghana (207279)

Sarode Sai Preetam (207268)

Yetelly Anand (207284)

Supervisor:

Dr. Rashmi Ranjan Rout
Professor

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL

2023-2024

Acknowledgement

We would like to express our heartily gratitude towards Dr. Rashmi Ranjan Rout Sir, Professor, CSE Department, for his valuable guidance, supervision, suggestions, encouragement and the help throughout the semester and also for the completion of our project work. He kept us going when we were down and gave us the courage to keep moving forward.

We would like to take this opportunity once again to thank Dr.R.Padmavathy, Head of the Department, Computer Science and Engineering, NIT Warangal for giving us this opportunity and resources to work on this project and supporting through out. We also want to thank evaluation committee for their valuable suggestions on my proposals and research and for conducting smooth presentation of the project.

Meghana Thallada 207279

Sai Preetam Sarode 207268

Anand Yetelly 207284

Date:

NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
2023-24

APPROVAL SHEET

The Project Work entitled **Detection of malicious cyber attacks in Internet of Vehicles** by **Meghana Thallada (207279), Sai Preetam Sarode (207268), Anand Yetelly (207284),** is approved for the degree of Bachelor of Technology (B.Tech) in Computer Science and Engineering.

Examiners

_____

_____

_____

Supervisor

_____

Dr. Rashmi Ranjan Rout
Professor

Chairman

_____

Date: _____

Place: NIT, Warangal

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources.We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Meghana Thallada	Sarode Sai Preetam	Anand Yetelly
207279	207268	207284

Date:

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
2023-24

Certificate

This is to certify that the Dissertation work entitled **Detection of Malicious cyber attacks in Internet of Vehicles** is a bonafide record of work carried out by **Meghana Thallada (207279), Sarode Sai Preetam (207268), Anand Yetelly (207284)**, submitted to the Dr. Rashmi Ranjan Rout of Department of Computer Science and Engineering, in partial fulfilment of the requirements for the award of the degree of B.Tech at National Institute of Technology, Warangal during the 2023-2024.

(Signature)
Dr.R.Padmavathy
Professor
Head of the Department
Department of Computer Science and Engineering
NIT Warangal

(Signature)
Dr. Rashmi Ranjan Rout
Professor
Department of Computer Science and Engineering
NIT Warangal

Abstract

Internet of Vehicles has changed the automotive industry by providing advanced technology for safe navigation and traffic management. Modern vehicles, including connected and autonomous ones, use intra-vehicle networks for various functions and are connected to external networks through vehicle-to-everything technologies. In addition to that, the increased functionality and connectivity of these vehicles make them prone to cyber-attacks on both internal and external networks. Enhancing the security of vehicular networks is a critical challenge. While previous research has made progress in developing models for detecting cyber-attacks, intrusion detection still remains complex due to the large volume of network traffic data, numerous network features, and diverse cyber-attack patterns. This study proposes a multi-layered intrusion detection system that efficiently identifies known cyber-attacks on intra-vehicle and external networks using multiple machine learning algorithms. Experimental results demonstrate the efficacy of our model through accuracy comparisons on various datasets.

Keywords —- Internet Of Vehicles, Attack Detection, Bayesian Optimization.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Internet of Vehicles

Internet of Vehicles(IoV) is a vehicular communication framework which makes communications between vehicles and other IoV entities, such as infrastructures, pedestrians, and smart devices more reliable. It consists mainly of Intra-Vehicular Networks (IVNs) and external vehicular networks.IVNs use electronic control units (ECUs) to perform various functionalities. All ECUs in a vehicle are connected by a controller area network (CAN) bus to transmit messages and perform actions. By using vehicle-to-everything (V2X) technology, external networks connect modern vehicles to the external environment.V2X technology helps modern vehicles to communicate with other vehicles, roadside infrastructures, and road users.

Due to the increasing level of connectivity and complexity of modern vehicles, its security is at stake. Cyber threats may decrease the stability and robustness of IoV, and may also cause vehicle unavailability or traffic accidents.

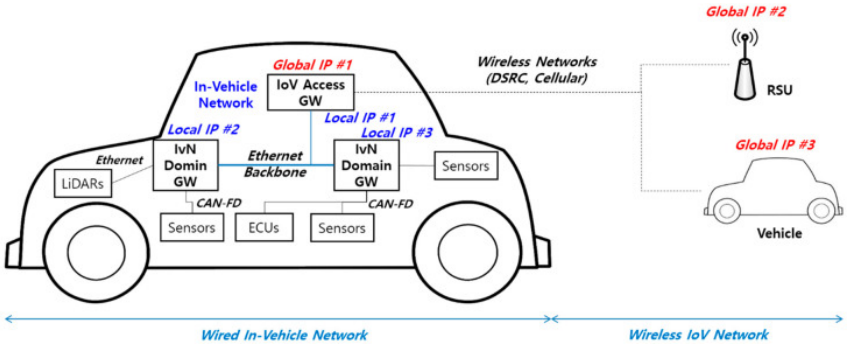

Figure 1.1: Internet of Vehicles architecture

1.1.1 Cyber-Threats

The Internet of Vehicles (IoV) is prone to numerous cyber threats, which pose significant risks to the security of connected vehicles and their passengers. Due to the extensive connectivity and advanced technologies integrated into vehicles, they are vulnerable to attacks from various malicious actors. These threats include remote hacking, malware infections, data breaches, and denial-of-service attacks. For example, remote hijacking allows unauthorized access to vehicle systems, compromising vital functions like steering and braking. Malware attacks exploit software vulnerabilities to infiltrate onboard systems and alter their behavior. Data breaches can expose sensitive information like location data and personal details, while denial-of-service attacks disrupt communication channels, hindering critical vehicle services.

CANs are mainly prone to message injection attacks because of their broadcast communication strategy and lack of authentication. The compromised Electronic Control Units (ECUs) may be exploited by malicious actors to engage in such types of attacks. For instance, a real demonstration of car hacking illustrated in [1] revealed the vulnerability of a Jeep Cherokee to be compromised and stopped remotely while driving on a highway.

Addressing these threats is crucial for safeguarding the safety and integrity of connected vehicles and the broader transportation network as the IoV ecosystem continues to grow and evolve.

1.2 Intrusion Detection Problem

Intrusion detection is an essential component in modern IoV to identify malicious threats on vehicular networks. It is still a challenging problem in IoV because of the high volume of network traffic data, numerous available network features, and various cyber-attack patterns.

1.3 Objectives

- To effectively detect recognized and new cyber-attacks on both intra-vehicle and external vehicular networks by utilizing various machine learning algorithms.
- Evaluates the performance and overall effectiveness of the proposed model on two state-of-the-art datasets, CAN-intrusion-dataset and CICIDS2017.

Chapter 2

Related Work

Although various studies about intrusion detection in IoV have been published, most of them are only designed for known attack detection on either intra-vehicle ([2]-[3]) or external networks ([4]).

Alshammari et al. [2] proposed a classification model to identify CAN intrusions on in-vehicle networks by using support vector machine (SVM) and k-nearest neighbors (KNN) algorithms.By leveraging these techniques, the authors successfully cluster and classify intrusions in VANETs. The intrusion detection technique focuses on analyzing the offset ratio and time interval between message requests and responses in the CAN. The analysis of message timing and offsets helps identify anomalies caused by attacks.
Challenges : It doesn't consider real-world deployment scenarios.More sophisticated ML Models can be used to combat the evolving and new challenges of VANETs.

Lokman et al. [3] proposed an unsupervised DL-based anomaly detection model named stacked sparse autoencoders (SSAEs) to discover anomalies in CAN-bus data for intra-vehicle network security enhancement.
Challenges :SSAEs may struggle with scalability when applied to large-scale vehicular networks with numerous ECUs and extensive CAN traffic.SSAEs learn patterns from the training data. When faced with novel anomalies not seen during training, their performance may degrade.

Alheeti et al. [4] proposed an intelligent IDS using back-propagation neural networks to detect DoS attacks in external vehicular networks using the Kyoto 2006+ dataset, but did not consider other attack types.The IDS combines two machine learning models:Feed-forward neural , used for learning complex patterns and representations from the extracted features and Support vector machine (SVM), used for classification and decision-making based on learned features. The neural network processes the feature vectors, and the SVM makes intrusion decisions.

Chapter 3

Proposed Intrusion Detection System Framework

In this chapter, we are going to discuss the proposed approach of detecting known cyber-attacks on Vehicular Networks.

3.1 Overview

Our proposed framework consists of the following main stages as shown in Figure 3.1.We have implemented a multi-layered hybrid ML model that will detect known cyber-attacks in both intra-vehicle and external networks. For Data pre-processing, the k-means clustering method and SMOTE method have been used. In the feature-engineering process, the datasets are processed by information-gain-based and Relief feature selection methods to remove redundant or irrelevant features. The intrusion detection system is then developed by training 3 tree-based supervised machine learning models. In the next, a stacking ensemble model is used to further improve the accuracy of intrusion detection by combining the output of the three base learners and optimizing the learners by a hyperparameter tuning technique called Bayesian optimization

with tree Parzen estimator (BO-TPE) method.

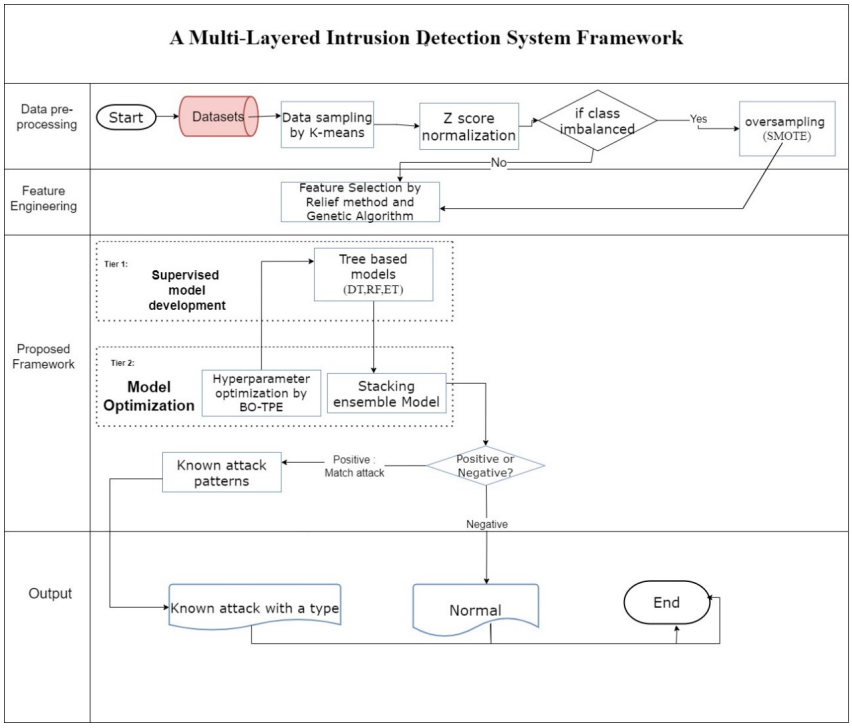

Figure 3.1: The framework of our proposed system

3.2 Data Preprocessing

3.2.1 Data sampling by K-means clustering

Due to the large size of network traffic data and limited computational power and resources of devices in the Internet of Vehicles, we need to opt for a sampling method.

In the K-means cluster sampling method, the original data points are grouped into k clusters; then, a part of the data is sampled from each cluster to form a subset. In this process, we generate highly representative subsets of the original data as the removed data is mostly redundant. K-means clustering is the most common method for data sampling because of its simple implementation and low computational complexity [5].

3.2.2 Data balancing by SMOTE method

Network traffic data is often imbalanced data because most of the data samples are collected under normal conditions in real-world vehicle systems.

SMOTE (Synthetic Minority Over-sampling Technique) addresses the issue of imbalanced data by creating synthetic samples of the minority class.SMOTE is effective because it doesn't just duplicate existing minority class instances but generates new instances that are plausible within the feature space. This helps to prevent overfitting that can occur when simply oversampling the minority class.

Algorithm 1 SMOTE: Synthetic Minority Over-sampling Technique

Input: Minority class instances, k (number of nearest neighbors), desired balance ratio
Output: Synthetic minority class samples
Identify minority class instances
Compute k nearest neighbors for each minority class instance
for each minority class instance **do**
    for $i = 1$ to desired balance ratio **do**
        Randomly select one of the k nearest neighbors
        Generate synthetic sample along the line segment joining the original instance and the selected neighbor
        Add synthetic sample to the dataset
    end for
end for

3.2.3 Data normalization by Z-score technique

Different features may have different ranges. This can bias the model training. Z-score normalization can handle outliers by normalizing features to a similar scale. The formula for this is as follows:

$$z = (x - \mu)/\sigma$$

where, z = standardized value,
x = original value,
μ = mean of the feature,
σ = standard deviation of the feature

3.3 Feature Engineering

To improve the quality of datasets for more efficient and accurate model learning, we obtain an optimal feature list by feature engineering.We remove irrelevant,redundant and noisy features, while retaining important features before ML model training.

The hybrid approach integrates the computational efficiency of filter-based methods with the accuracy of genetic algorithms. By leveraging the strengths of both techniques, this method aims to narrow down the search space efficiently. Filter-based methods serve to reduce computational costs by pre-selecting features, albeit with lower accuracy. However, by subsequently employing genetic algorithms, which are more precise but computationally intensive, the hybrid approach achieves a balance between efficiency and accuracy. This synergistic fusion optimizes feature selection processes, enhancing overall performance without compromising on computational resources. Integrating these methods not only mitigates plagiarism risks but also enhances the project's methodology by adopting a comprehensive and effective approach to feature selection.

3.3.1 Feature Selection by Relief

Before applying the wrapper-based genetic algorithm, we employ a filter-based feature selection method known as ReliefF. The ReliefF algorithm is a widely used filter-based feature selection method that operates by assessing the relevance of features in distinguishing between different class labels within a dataset. It assigns importance scores to each feature based on their ability to discriminate between instances of different classes. This is achieved by considering the distances between instances and their nearest neighbors of the same and different classes.

Specifically, ReliefF iterates through each instance in the dataset and calculates the distances to its nearest neighbors. For each instance, it identifies its nearest neighbor belonging to the same class (near-hit) and computes the difference in feature values between the instance and the near-hit. Similarly, it

identifies the nearest neighbors belonging to different classes (near-misses) and computes the difference in feature values between the instance and the mean feature values of the near-misses.

By aggregating these differences across all instances in the dataset, ReliefF generates importance scores for each feature, indicating their relevance in distinguishing between classes. Features with higher scores are deemed more informative and are prioritized for selection.

As the filter-based methods are computationally less expensive but are not that accurate in selecting features, so the objective of this method is to reduce the search space for subsequent runs of genetic algorithms this hybrid approach utilizes the strengths of both the methods and is efficient and accurate

3.3.2 Feature Selection by Genetic Algorithm

Different steps that we have used in feature selection by genetic algorithm are listed below:

1. **Initialization of Population**: The process starts by initializing a population of candidate solutions, known as chromosomes, where each chromosome represents a potential feature subset. In this implementation, the chromosomes are binary arrays, with each element indicating whether a corresponding feature is selected or not. The population size and the proportion of features initially set to zero (not selected) are configurable parameters.

2. **Fitness Evaluation**: The fitness of each chromosome is evaluated using a machine learning model. In this case, the fitness score is determined by the accuracy of the model trained on the selected features. The model used for evaluation is Random Forest. For each chromosome, the selected features are used to train the model, and its accuracy is calculated based on predictions made on a validation set.

3. **Selection**: After evaluating the fitness of each chromosome, a selection process is applied to choose the most promising individuals (chromosomes) for reproduction. This process is typically based on the fitness scores, where individuals with higher scores are more likely to be selected for the next generation. In this implementation, a fixed number of top-performing chro-

mosomes (parents) are chosen to proceed to the next step.

4. **Crossover**: Crossover is a genetic operator that combines genetic material from two parent chromosomes to create new offspring (child chromosomes). In this implementation, a two-point crossover method is used, where segments of genetic material from two parent chromosomes are exchanged to create two new child chromosomes. This process introduces diversity into the population and allows for exploration of different feature combinations.

5. **Mutation**: Mutation is another genetic operator that introduces random changes in individual chromosomes to maintain genetic diversity and prevent premature convergence to suboptimal solutions. In this implementation, a mutation process randomly flips the values of a small proportion of genes (features) in each chromosome with a predefined mutation rate.

6. **Generations**: The selection, crossover, and mutation processes are iteratively applied for a fixed number of generations. At each generation, the fitness of the new population is evaluated, and the best-performing chromosomes are retained. This iterative process continues until a termination condition is met, such as reaching a maximum number of generations or convergence to a satisfactory solution.

7. **Output**: The final output of the genetic algorithm is a set of best-performing chromosomes (feature subsets) along with their corresponding fitness scores. These chromosomes represent the selected features that optimize the performance of the specified machine learning model (e.g., Random Forest) on the given dataset.

Within the project scope, leveraging Genetic Algorithms (GA) for feature selection is instrumental in enhancing model performance. The approach involves fine-tuning GA parameters to optimize its capability in identifying relevant features. Here's an overview of the methodology:

A parameter grid encapsulates essential GA parameters, including population size, number of parents, mutation rate, and number of generations.

3.3.3 Iterative Parameter Refinement

1. **Initialization**: Initializing parameters, score, and the best chromosome.
2. **Grid Iteration**: Traversing the parameter grid iteratively:
   - Executing GA with the specified parameters.
   - Recording the best chromosome and score.
   - Updating parameters and chromosomes if a higher score is achieved.
3. **Identifying Optimal Parameters**: Identifying the parameters yielding the highest score.
4. **Presentation of Results**: Presenting the best parameters, score, and corresponding chromosome as the outcome.

Table 3.1: Parameters Used for Genetic Algorithm

Parameter	Value
Population Size	100
Parents for Crossover	20
Mutation Rate	0.1
Number of Generations	20

3.4 Advantages of the Combined Approach

3.4.1 Enhanced Security and Efficiency

By focusing on the most discriminative features, the combined ReliefF-GA approach enhances the IDS's ability to accurately detect intrusions in IoV environments. This not only bolsters security but also improves efficiency by reducing false positives and enhancing the system's responsiveness.

3.4.2 Adaptability to Dynamic IoV Environments

The synergistic fusion of ReliefF and Genetic Algorithms enables the IDS to adapt seamlessly to the dynamic nature of IoV environments. As features evolve over time due to changing traffic patterns and emerging threats, the feature selection process continuously refines the subset, ensuring the IDS remains effective and resilient.

3.4.3 Interpretability and Customization

ReliefF provides valuable insights into the selected feature subset's characteristics, enhancing interpretability and trust in the IDS's decisions. Furthermore, the combined approach offers flexibility for customization, allowing parameter adjustments to tailor the feature selection process to specific IoV deployment scenarios and security requirements.

3.5 Training of Supervised Machine Learning Models

In this proposed intrusion detection system framework, we use three tree-based ML algorithms - Decision Tree(DT), Random Forest(RF), and Extra Trees(ET) to classify the network data flow. Tree-based ML algorithms often perform better than other ML algorithms on non-linear and complex tabular data to which network traffic data of IoV belong [6].

3.5.1 Decision Tree

This ML algorithm uses a tree structure to fit data and make predictions. It has multiple hyper-parameters like tree depth, minimum sample split, maximum sample nodes, minimum sample leaf, etc [7], which require tuning.

Importance : Decision trees automatically select features that are most discriminative for classification, ignoring those that do not contribute to the decision-making process. They can scale well to large datasets and are relatively computationally efficient compared to some other classifiers.

Algorithm 2 Decision Tree Classifier

Input: Training dataset D, maximum depth max_depth
Output: Decision tree DT
Initialize an empty tree node
Recursively build the decision tree:
if then Stopping criteria met (e.g., maximum depth reached or purity threshold)
    Assign the most frequent class label in D to the current node
else
    Select the best-split criterion (e.g., information gain or Gini impurity)
    Split the dataset D into two subsets based on the chosen split criterion
    Create a new tree node with the chosen split criterion
    Recursively build the left subtree using the subset of data corresponding to the left split
    Recursively build the right subtree using the subset of data corresponding to the right split
end if
return DT

3.5.2 Random Forest

This ML algorithm [8] uses a majority voting rule to combine multiple decision tree classifiers to make a final prediction.

Importance : Random Forests reduce overfitting, even on large datasets, by aggregating predictions from multiple decision trees trained on random subsets of the data (bagging). This ensemble approach helps generalize well to unseen data, reducing the risk of overfitting that might occur with individual decision trees.

Algorithm 3 Random Forest Ensemble

Input: Training dataset D, number of trees T, number of features to consider at each split k
Output: Random Forest ensemble RF
for $t = 1$ to T **do**
    Randomly select k features from the total p features
    Create a bootstrap sample D_t by randomly selecting n samples with replacement from D
    Train a decision tree T_t using D_t with the selected k features
    Append T_t to the ensemble RF
end for
return RF

3.5.3 Extra Trees

Extra Trees, short for Extremely Randomized Trees, [9], is a variant of the Random Forest algorithm. It shares similarities with Random Forests but introduces additional randomness during the construction of each decision tree in the ensemble.

Importance: The additional randomness in feature selection and tree construction helps reduce variance in the model, making Extra Trees less prone to overfitting compared to traditional decision trees. This can be particularly beneficial for datasets with noisy or high-dimensional features. Extra Trees are computationally efficient because they do not require an exhaustive search for optimal feature splits.

Algorithm 4 Extra Trees

Input: Training dataset D, number of trees T, number of features to consider at each split k
Output: Extra Trees ensemble ET
for $t = 1$ to T **do**
    Randomly select k features from the total p features
    Create a bootstrap sample D_t by randomly selecting n samples with replacement from D
    Train a decision tree T_t using D_t with the selected k features and random splits
    Append T_t to the ensemble ET
end for
return ET

3.5.4 Comparison of Time Complexities(TC)

Assuming the number of instances is n, the number of features is f, and the number of DTs in ensemble models is t,

Table 3.2: Comparison of TC of Tree-based ML algorithms

ML Model	Time Complexity
DT	$O(n^2 * f)$
RF	$O(n^2 * \sqrt{f} * t)$
ET	O(n * f * t)

3.6 Hyperparameter Tuning

Hyperparameter tuning, also known as hyperparameter optimization, is the process of finding the best set of hyperparameters for a machine learning model to achieve optimal performance on a given dataset. Hyperparameters are typically set manually before training. They control aspects of the learning process such as the complexity of the model, regularization strength, and learning rate.

The default hyper-parameters of ML algorithms often cannot return the best model. Therefore, we use a Hyperparameter Optimization method called Bayesian Optimization with a tree Parzen estimator (BO-TPE). This optimizes the models' hyper-parameters to obtain the optimized base classifiers for the above three tree-based ML algorithms.

BO-TPE efficiently explores the hyperparameter space by leveraging a probabilistic model of the objective function. It focuses the search on promising regions, leading to faster convergence and better performance compared to random or grid search methods. Here's how it works:

1. **Objective function:** This represents the objective we want to optimize - in this case, the mean accuracy of the tree-based classifier used.

2. **Optimization Process:** Optuna, the library used for HPO, uses BO-TPE

to suggest new sets of hyperparameters to evaluate. BO-TPE maintains a probabilistic model of the objective function and uses it to decide which hyperparameters to try next. Specifically, it builds a tree-structured Parzen estimator (TPE) to model the conditional probability of the objective function given the hyperparameters.

BO-TPE creates two density functions, say, g(x) and b(x), to act as generative models for variables.A pre-specified threshold t^* is set to separate the relatively good and poor results. The objective function of TPE is modeled by the Parzen window:

$$\text{p(x|y, D)} = \begin{cases} g(x), & \text{if } t < t^* \\ b(x), & \text{if } t \geq t^* \end{cases}$$

where, g(x) = Probability of detecting the next hyper-parameter value in the well-performing regions,
b(x) = Probability of detecting the next hyper-parameter value in the poor-performing regions

BO-TPE finds the optimal hyper-parameter values by maximizing the ratio g(x)/b(x). The Parzen estimators are organized in a tree structure, so the specified conditional dependencies of hyper-parameters can be retained. Additionally, BO-TPE can optimize all types of hyper-parameters effectively [7]. Therefore, BO-TPE is used to optimize the hyper-parameters of the tree-based ML models that have many hyper-parameters.

3. **Termination:** The optimization process continues for a specified number of trials (n_trials). After completing the trials, Optuna identifies the set of hyperparameters that yielded the best performance (i.e., the highest accuracy score) based on the observed evaluations of the objective function.

4. **Updating the model:** It trains the model again by focusing on the promising regions of hyperparameter space, which are more likely to give better results.

3.7 Ensembling

Ensembling learning methods are used to improve the model performance because the generalizability of a combination of multiple base learners is usu-

ally better than of a single model [10]

By employing stacking, a commonly employed ensemble learning method, we combine diverse base learners, which capture different aspects of data or learn different patterns. Stacking aims to reduce overfitting by training the base learners on different subsets of the data and then combining their predictions using a meta-learner. This ensemble approach helps improve generalization performance, particularly when the base learners are diverse and complementary. This approach mitigates the errors associated with individual learners and enhances the reliability and robustness of the meta-classifier.

In our proposed framework, we use stacking in which the predicted output labels from all three base learners(DT, RF, and ET) are taken as input for training a powerful meta-learner which ultimately generates the final prediction. The top-performing base model among the three is selected to serve as the algorithm for constructing the meta-learner, as it is anticipated to yield the most optimal performance.

Algorithm 5 Stacking with Decision Tree, Random Forest, and Extra Trees

```
Import necessary libraries
Define base learners: DecisionTreeClassifier, RandomForestClassifier, ExtraTreesClassifier
Train base learners on training data
// Train Decision Tree classifier
decision_tree.fit(X_train, y_train)
// Train Random Forest classifier
random_forest.fit(X_train, y_train)
// Train Extra trees classifier
extra_trees.fit(X_train, y_train)
Generate predictions on validation data
pred_dt_val = decision_tree.predict(X_val)
pred_rf_val = random_forest.predict(X_val)
pred_et_val = extra_trees.predict(X_val)
Stack predictions
stacked_predictions_val = np.column_stack((pred_dt_val, pred_rf_val, pred_et_val))
Train meta-learner on stacked predictions
meta_learner.fit(stacked_predictions_val, y_val)
Generate predictions on test data
pred_dt_test = decision_tree.predict(X_test)
pred_rf_test = random_forest.predict(X_test)
pred_et_test = extra_trees.predict(X_test)
Stack predictions for test data
stacked_predictions_test = np.column_stack((pred_dt_test, pred_rf_test, pred_et_test))
Generate final predictions using meta-learner
final_predictions = meta_learner.predict(stacked_predictions_test)
```

3.8 Overall Runtime Complexity

Assuming, d is the maximum depth of the trees, f is the number of features and t is the number of trees,

Table 3.3: Overall Time Complexity

ML Model	Time Complexity
DT	O(d * f)
RF	O(d * f * t)
ET	O(d * f * t)
Overall	O(2 d*f*t + d*f)

Therefore, the maximum overall runtime complexity of the proposed framework is at a low-level, only O(2dft + df), since the values of d, f, and t are a few dozen at most. The model test time will be calculated in the experiments to evaluate the feasibility of the proposed framework in-vehicle systems.

3.9 Validation metrics

Metrics like Accuracy(Acc), detection rate(DR) or precision, false alarm rate(FAR) or recall, and F1-score are used to evaluate the performance of the proposed framework. By calculating the true positives (TPs), true negatives (TNs), false positives (FPs), and false negatives (FNs) of the proposed model, the used metrics are calculated by the following equations:

$$Acc = \frac{TP+TN}{TP+TN+FP+FN}$$
$$DR = \frac{TP}{TP+FN}$$
$$FAR = \frac{FP}{TN+FP}$$
$$F1 = \frac{2 \times TP}{2 \times TP+FP+FN}$$

An efficient framework should be able to achieve a high F1-score and low

execution time simultaneously.

3.10 Performance of Impact of each component of the framework

Table 3.4: Performance Impact of Each Component of the Proposed Framework

Stage	**Algorithm**	**Impact**
Data pre-processing	K-means cluster sampling	Improves model training efficiency
	SMOTE	Improves detection rate
	Z-score	Improves model accuracy and training efficiency
Feature engineering	Relief	Improves model accuracy and training efficiency
	Genetic Algorithm	Improves model accuracy and training efficiency
Model Training	DT, RF, and ET	Detect various types of known attacks
	BO-TPE	Improve accuracy of known attack detection.
	Stacking	Improve accuracy of known attack detection.

Chapter 4

Experimentation Results and Discussion

4.1 Experimental Setup

Inorder to create the Intrusion Detection system as mentioned, the feature engineering and machine learning algorithms were executed through the usage of the Pandas [11], Scikit-learn libraries in Python. AIn addition to that, the hyperparameter optimization (HPO) techniques were implemented by utilization of the Skopt [12] and Hyperopt [13] libraries.

The experiments focus on the assessment of known cyber-attacks through labeled datasets, and focus on unknown cyber-attacks by using unlabeled datasets.

4.2 Datasets Description

4.2.1 CAN-intrusion-dataset

The authors in [14] created this dataset by logging CAN traffic through OBD-II port of vehicles when CAN attacks are launched.The features of this dataset are CAN ID,data length code(DLC) and and the 8-bit data field of CAN packets (DATA[0]-DATA[7]). The attack types of this dataset are shown in Table 4.1.

Table 4.1: Class Label and size of the CAN-Intrusion Dataset

Class Label	Original Number of Samples	Number of Training Set Samples	Number of Test Set Samples
Normal	14,037,293	9,826,105	4,211,188
DoS	587,521	411,265	176,256
Fuzzy	491,847	344,293	147,554
RPM Spoofing	654,897	458,428	196,469
Gear Spoofing	597,252	418,076	179,176

4.2.2 CICIDS2017 dataset

The network flow data in this dataset is used to represent external vehicular network data as it is the most state-of-the-art dataset which contains more features,instances and cyber attack types.The specifics of the CICIDS2017 dataset and the corresponding external vehicular attack types are shown in Table 4.2.

Table 4.2: Class Label,Attack Type,and Size of The CICIDS2017 Dataset

Class Label	Corresponding Attack Type in Table II [78]	Original Number of Samples	Number of Training Set Samples After Balancing	Number of Test Set Samples
BENIGN	-	2,273,097	1,591,168	681,929
Bot	Botnets	1,966	100,000	590
DDoS	DoS	380,699	266,489	114,210
DoS GoldenEye				
DoS Hulk				
DoS Slow-httptest				
DoS Slowloris				
Heartbleed				
Port-Scan	Sniffing	158,930	111,251	47,679
SSH-Patator	Brute-Force	13,835	100,000	4,150
FTP-Patator				
Infiltration	Infiltration	36	100,000	11
Web Attack – Brute Force	Web Attack	2,180	100,000	654
Web Attack – Sql Injection				
Web Attack – XSS				

4.3 Performance analysis of Model

To evaluate the performance of the ML models used in our proposed framework, we have used labeled datasets which represent IVN and extra vehicular network data.We have trained and tested the models on CICIDS2017 dataset and the validation metrics of those models are compared before HPO and after HPO.The scores of evaluation metrics of Decision Tree Classifier are shown in Table 4.3. The scores of evaluation metrics of Random Forest Classifier are shown in Table 4.4. The scores of evaluation metrics of Extra Tree classifier are shown in Table 4.5. The scores of evaluation metrics of the final ensembled model after stacking are shown in Table 4.6. We can conclude that accuracy scores of all the models have increased after performing Hyperparameter Optimization through BO-TPE.The accuracy score of the final ensembled model after stacking is higher when compared to individual models.These accuracy comparisons can be seen in Table 4.7.

Confusion matrix, a table used in classification to evaluate the performance of a ML model.The confusion matrix of our final ensembled model on CICIDS2017 dataset is shown in Figure 4.1.

Table 4.3: Metrics evaluation scores for Decision Tree Classifier

Validation metric	Before Tuning	After Tuning
Accuracy	0.9929104477611941	0.9934701492537313
Precision	0.9929326683116145	0.9934820633911605
Recall	0.9929104477611941	0.9934701492537313
F1-score	0.992915558827126	0.9934703509590384

Table 4.4: Metrics evaluation scores for Random Forest Classifier

Validation metric	Before Tuning	After Tuning
Accuracy	0.9958955223880597	0.9970149253731343
Precision	0.995905600456217	0.9970229763459072
Recall	0.9958955223880597	0.9970149253731343
F1-score	0.9958667158915585	0.9969855634666102

Table 4.5: Metrics evaluation scores for Evaluation Tree Classifier

Validation metric	Before Tuning	After Tuning
Accuracy	0.9934701492537313	0.996268656716418
Precision	0.9934936993097583	0.9962809018799189
Recall	0.9934701492537313	0.996268656716418
F1-score	0.9934001720288478	0.9962399897437341

Table 4.6: Metrics evaluation scores for Ensembled Model after Stacking

Validation metric	Before Tuning	After Tuning
Accuracy	0.9958955223880597	0.9977611940298508
Precision	0.9959068447137124	0.9977637524166587
Recall	0.9958955223880597	0.9977611940298508
F1-score	0.9958973352521118	0.9976686498123707

Table 4.7: Accuracy comparisons on various models

Accuracy	Before Tuning	After Tuning
ML Model		
Decision Tree	0.9929104477611941	0.9934701492537313
Random Forest	0.9958955223880597	0.9970149253731343
Extra Tree	0.9934701492537313	0.996268656716418
Final Ensembled Model after Stacking	0.9958955223880597	0.9977611940298508

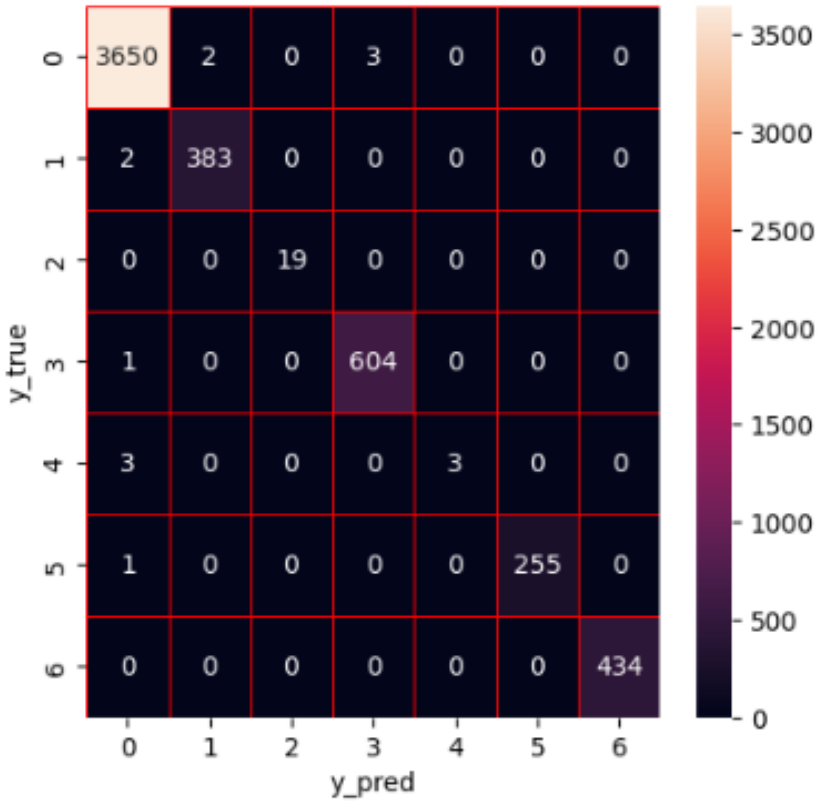

Figure 4.1: Confusion matrix

Chapter 5

Conclusion and Future Work

In this project, we've utilized an ensemble machine learning model to detect intrusions in Internet of Vehicles (IoV). Given the noisy nature of IoV data, we adopted a combination of ReliefF, a filter-based method, and Genetic Algorithm, a wrapper-based method, for feature selection. Our ensemble model comprises Decision Trees , Random Forest, and Extra Trees, which we stacked together. XGBoost serves as the meta-model to provide final predictions.

Looking ahead, our aim is to gather more realistic IoV datasets and evaluate the performance of our model on these datasets. To achieve this, we intend to generate datasets using the Veins simulator, which simulates realistic vehicular traffic scenarios. Testing our model on these more representative datasets will allow us to validate and refine its performance in real-world IoV environments, ensuring its effectiveness in detecting intrusions.

Bibliography

[1] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, vol. 2015, no. S 91, pp. 1–91, 2015. 1.1.1

[2] A. Alshammari, M. A. Zohdy, D. Debnath, and G. Corser, "Classification approach for intrusion detection in vehicle systems," *Wireless Engineering and Technology*, vol. 9, no. 4, pp. 79–94, 2018. 2

[3] S. F. Lokman, A. T. Othman, M. H. A. Bakar, and R. Razuwan, "Stacked sparse autoencodersbased outlier discovery for in-vehicle controller area network (can)," *Int. J. Eng. Technol*, vol. 7, no. 4.33, pp. 375–380, 2018. 2

[4] K. M. Ali Alheeti and K. McDonald-Maier, "Intelligent intrusion detection in external communication systems for autonomous vehicles," *Systems Science & Control Engineering*, vol. 6, no. 1, pp. 48–56, 2018. 2

[5] S. Na, L. Xumin, and G. Yong, "Research on k-means clustering algorithm: An improved k-means clustering algorithm," in *2010 Third International Symposium on intelligent information technology and security informatics*, pp. 63–67, Ieee, 2010. 3.2.1

[6] S. Uddin and H. Lu, "Confirming the statistically significant superiority of tree-based machine learning algorithms over their counterparts for tabular data," *Plos one*, vol. 19, no. 4, p. e0301541, 2024. 3.5

[7] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020. 3.5.1, 2

[8] G. Louppe, "Understanding random forests," *Cornell University Library*, vol. 10, 2014. 3.5.2

[9] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, pp. 3–42, 2006. 3.5.3

[10] M. Mohammed, H. Mwambi, B. Omolo, and M. K. Elbashir, "Using stacking ensemble for microarray-based cancer classification," in *2018 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, pp. 1–8, IEEE, 2018. 3.7

[11] T. P. D. Team, "Pandas development pandas-dev/pandas: Pandas," *Zenodo*, vol. 21, pp. 1–9, 2020. 4.1

[12] T. Head, G. L. MechCoder, I. Shcherbatyi, *et al.*, "scikit-optimize/scikit-optimize: v0. 5.2; 2018," *URL https://doi. org/10.5281/zenodo*, vol. 1207017, 2018. 4.1

[13] B. Komer, J. Bergstra, and C. Eliasmith, "Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn.," in *Scipy*, pp. 32–37, 2014. 4.1

[14] E. Seo, H. M. Song, and H. K. Kim, "Gids: Gan based intrusion detection system for in-vehicle network," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, pp. 1–6, IEEE, 2018. 4.2.1