國立臺北科技大學

資訊安全碩士學位學程

碩士學位論文

Master of Science in Information Security

Master Thesis

TransBoost-DoS: A Hybrid Detection System for Denial-of-Service Attacks in VANETs

研究生：曾祥語

Researcher: Hsiang-Yu Tseng

指導教授：陳香君博士

Advisor: Shiang-Jiun Chen, Ph.D.

July 2025

國立臺北科技大學
研究所碩士學位論文口試委員會審定書

本校 資訊安全碩士學位學程 研究所 曾祥語 君

所提論文，經本委員會審定通過，合於碩士資格，特此證明。

學位考試委員會

委　　員： 馬奕葳

吳和庭

陳香君

指導教授： 陳香君

所　　長： [illegible]

中 華 民 國　一百一十四　年　七　月　二十二　日

Abstract

Keywords: VANETs, Denial-of-Service Attack, Hybrid Learning, Machine Learning, Deep Learning, Cybersecurity, Intelligent Transportation Systems, XGBoost, Transformer, SUMO Simulation.

With the continued advancement of vehicular communication technologies and the integration of 5G into Software-Defined Vehicles (SDVs) and Vehicular Ad-hoc Networks (VANETs), both vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications have experienced substantial improvements. While these innovations enable critical functionalities such as safety services, traffic coordination, and over-the-air (OTA) software updates, they also introduce new security risks—most notably Denial-of-Service (DoS) attacks—that can severely disrupt communication and endanger transportation safety.

To mitigate these threats, we propose a hybrid learning-based detection pipeline, TransBoost-DoS. Our method employs a Transformer-based feature extractor to learn temporal dependencies from vehicular message sequences, effectively modeling dynamic vehicle behaviors. Extracted features are then classified by an XGBoost model to detect and distinguish among five types of DoS attacks and benign traffic.

Evaluated using 5-fold cross-validation on a balanced VeReMi dataset, our proposed model reaches a high accuracy of 99.67%, demonstrating robust outcome in identifying abnormal behaviors with minimal false positives.

Acknowledgements

在本論文即將完成之際，回顧這段求學與研究的旅程，心中充滿無限感謝之情。若沒有眾多師長與親友的協助與鼓勵，我無法獨自完成這項挑戰。謹藉此機會，向所有曾經支持、陪伴與指導我的人致上最誠摯的謝意。

首先，我要由衷感謝我的指導教授陳香君博士。感謝老師在整個研究過程中耐心指導與悉心教誨，不僅在學術方向上提供我清晰的指引，也在寫作、思考與實作細節上給予我莫大的幫助。老師嚴謹的學術態度與對學生的用心關懷，是我在研究路上最重要的啟發與支撐。

我也要感謝在研究所期間結識的夥伴 Mick、Ricky、Sam、璟霆、劭睿、正承、建璋、子瑜、玟萱、德劭、楊越、柏勳、承諺、仔君。謝謝你們一路相伴，無論是在課堂討論、論文交流，或是日常生活的分享中，都給予我深刻的啟發與正面的影響。你們的陪伴與支持，讓我的研究生生活豐富而精彩，是我珍貴的回憶。

我更要深深感謝我的家人。感謝父母長期以來的支持、包容與理解，無論順境或逆境，您們始終是我最堅強的後盾。感謝你們從未要求我成為完美，只希望我盡力、快樂地做自己。這份信任與自由，是我勇敢面對挑戰的重要來源。

最後，我想要特別感謝我的女朋友，是她在我遇到困難沒有自信的時候，給予我鼓勵與支持。她的陪伴讓我在研究過程中不再孤單，並且在我需要時提供了無私的幫助。她的愛與理解是我最大的動力來源。

謹以本論文獻給所有曾經幫助我、相信我與陪伴我的人。謝謝你們。

曾祥語 謹誌於

臺北科技大學資訊安全碩士學位學程班

中華民國 114 年 7 月 31 日

Table of Contents

Abstract i
Acknowledgements ii
Table of Contents iii
List of Figures v
List of Tables vii
Chapter 1 Introduction 1
Chapter 2 Related Work 4
2.1 VANETs 4
2.2 Vehicle Communication Technologies 5
2.2.1 DSRC 6
2.2.2 C-V2X 6
2.2.3 Basic Safety Messages (BSMs) [17] 7
2.2.4 Vulnerabilities in Vehicular Communication Technologies 7
2.3 Attack Simulation and Dataset Generation 8
2.3.1 Attack Simulation 8
2.3.2 Dataset Generation 9
2.4 Machine Learning and VANETs 9
2.5 Transformer 12
2.6 XGBoost 13
Chapter 3 Methodology 15
3.1 System Architecture 15
3.1.1 Topology Generation Module 16
3.1.2 Behavior Packages Generation Module 16
3.1.3 Attack Detection Module 17
3.1.4 Report Module 17
3.2 Model Architecture 17
3.2.1 Data Preprocessing 19

3.2.2 Transformer-based Feature Extraction [31] 19
3.2.3 XGBoost-based Attack Classification [34] 20
3.2.4 Evaluation 21
Chapter 4 Implementation 22
4.1 Setup 22
4.1.1 Hardware and Software Requirements 22
4.1.2 Environment Setup 23
4.2 Model Implementation Details 30
4.2.1 Data Reduction & Balancing 30
4.2.2 Data Split 31
4.2.3 Model Training 31
4.2.4 Model Evaluation 32
4.3 Attack Detection Scenario 33
4.4 Simulation Implementation Details 35
4.4.1 Simulation Configuration 35
4.4.2 SUMO Network and Route File Creation 36
Chapter 5 Result & Analysis 38
5.1 Evaluation Metrics 38
5.2 Results 39
5.2.1 Performance Analysis of TransBoost-DoS 39
5.2.2 Comparison with Baseline Models 40
5.2.3 Discussion 43
5.3 Simulation Results 44
5.4 Comparison with existing works 46
Chapter 6 Conclusion & Future Work 48
6.1 Conclusion 48
6.2 Future Work 48
References 50

List of Figures

2.1 VANETs Security Requirements [10] 5

3.1 TransBoost-DoS system architecture 15

3.2 TransBoost-DoS model architecture 18

3.3 Raw dataset 20

3.4 Data preprocessing flowchart 20

4.1 Anaconda official website 24

4.2 Anaconda for creating new environment 24

4.3 Anaconda for activating environment 24

4.4 Anaconda for installing PyTorch 25

4.5 Anaconda for installing xgboost 25

4.6 Anaconda for installing packages 26

4.7 Anaconda for listing installed packages 26

4.8 SUMO official website 27

4.9 SUMO installer Initial screen 27

4.10 SUMO installer License Agreement 28

4.11 SUMO installer destination folder 28

4.12 SUMO ready to install 29

4.13 SUMO installer progress 29

4.14 SUMO finished installation 30

4.15 Data split process 31

4.16 5-fold cross-validation 33

4.17 DoS attack detection scenario 34

4.18 DoS attack detection flowchart 34

4.19 SUMO netedit for creating network file 36

4.20 SUMO netedit for creating route file 37

4.21 SUMO GUI for visualizing the simulation 37

5.1 Confusion matrix of TransBoost-DoS 40
5.2 Classification report of TransBoost-DoS 40
5.3 Confusion matrix of XGBoost-only model 42
5.4 Confusion matrix of Transformer-only model 42
5.5 Classification report of XGBoost-only model 43
5.6 Classification report of Transformer-only model 43
5.7 Simulation log of prediction and message dropping 44
5.8 Simulation log of prediction and message forwarding 45
5.9 Simulation confusion matrix 45
5.10 Simulation classification report 46

List of Tables

3.1 Complementarity of Transformer and XGBoost in hybrid architecture 18

4.1 Hardware requirements 22

4.2 Software requirements 23

4.3 Main python packages used 23

4.4 Simulation parameters 35

5.1 Performance comparison of baseline models 41

5.2 Comparison with methods from [29] 46

Chapter 1 Introduction

The rapid advancement of vehicular communication systems, coupled with the emergence of 5G-powered Software-Defined Vehicles (SDVs) [1], is reshaping the landscape of the automotive sector. By incorporating Vehicular Ad-hoc Networks (VANETs) [2] with 5G connectivity and SDVs architectures, substantial improvements have been achieved in both V2V and V2I communications [3]. These developments support a wide range of intelligent functions, including enhanced road safety, dynamic traffic coordination, and seamless OTA software updates [4]. Nonetheless, this increasing reliance on wireless technologies also brings heightened exposure to cyber threats, particularly DoS attacks [5], which pose significant challenges to the resilience and trustworthiness of vehicular networks.

Recent market forecasts indicate that the global market for SDVs is poised to expand significantly—from USD 213.5 billion in 2024 to an estimated USD 1,237.6 billion by 2030, reflecting a robust compound annual growth rate (CAGR) of 34.0% [6]. This rapid growth is largely fueled by the widespread implementation of OTA update mechanisms, which enable vehicles to receive timely software upgrades and security patches. Additionally, automakers benefit from this paradigm shift through reduced costs associated with product recalls, enhanced real-time system diagnostics, and the ability to perform remote maintenance, thereby optimizing vehicle lifecycle efficiency.

Despite the advantages of SDVs and 5G-Vehicle-to-everything (5G-V2X) communication, the growing dependence on software-driven vehicular functions introduces new security risks. DoS attacks [7], in particular, exploit the open nature of VANETs by overwhelming the network with excessive or malicious data traffic, disrupting communication between vehicles and infrastructure. Such attacks can degrade network performance, cause significant delays in message delivery, disrupt critical safety applications, and even lead to road hazards. Traditional cryptographic authentication mechanisms alone are insufficient to mitigate these large-scale disruptions, necessitating the enhancement of machine learning (ML) based detection techniques to identify abnormal traffic patterns and proactively defend against DoS threats in vehicular networks.

To address the evolving nature of DoS attacks in vehicular networks, we implement a hybrid learning-based detection pipeline, termed TransBoost-DoS. The system first employs a Transformer-based feature extractor to learn temporal dependencies from sliding windows of vehicular message sequences, effectively capturing complex patterns of vehicle behavior. These semantic feature representations are then passed to an XGBoost classifier, which performs multi-class classification across five types of DoS attacks and Normal class, enabling both attack detection and differentiation from benign traffic.

Transformers excel at modeling sequential dependencies and extracting high-level temporal features from time-series data, making them suitable for representing dynamic vehicle behaviors over time. On the other hand, XGBoost is a powerful gradient boosting framework known for its robustness, efficiency, and interpretability in tabular classification tasks. By using Transformer for deep temporal encoding and XGBoost for final decision-making, the hybrid model achieves both high representational capacity and strong classification performance, while mitigating overfitting and enhancing generalization.

To determine the proposed model's performance capabilities, we conducted 5-fold cross-validation [8] on the balanced version of the VeReMi dataset. The evaluation results indicate that the hybrid approach achieves an impressive average accuracy of 99.67%, highlighting its strong capability in identifying anomalous patterns while preserving a low false alarm rate for legitimate traffic. To further verify its robustness in practical settings, we carried out supplementary simulations using the Simulation of Urban MObility (SUMO) framework [9]. The results from these simulations reinforce the model's reliability in detecting misbehavior under dynamic, real-world-like traffic conditions, consistently maintaining high precision and minimal false positives.

The remainder of this paper is structured as follows: Chapter 2 reviews prior research related to VANETs security, including developments in inter-vehicle communication, attack modeling, dataset creation, and the use of ML for anomaly detection. Chapter 3 outlines the methodology behind the proposed TransBoost-DoS framework, covering data preprocessing, model design, and overall system architecture. Chapter 4 discusses implementation specifics, such as the software and hardware configurations, training procedures, and deployment within the SUMO simulation environment. Chapter 5 presents experimental findings, highlighting the model's detection accuracy across various attack scenarios. Lastly, Chapter 6 concludes with a summary of key findings and suggests future work, focusing on expanding the detection scope to include additional attack types, incorporating multi-modal sensor data, and optimizing the model for real-time deployment on vehicular edge devices in practical VANETs environments.

Chapter 2 Related Work

As VANETs continue to evolve and become integral to intelligent transportation systems, their associated security risks have drawn increasing attention from the research community. While these networks enable real-time interactions among vehicles and with roadside infrastructure, they are concurrently vulnerable to numerous cybersecurity threats, including message falsification, replay attacks, and DoS attacks. In order to comprehensively understand these vulnerabilities and assess existing defense strategies, this section presents a survey of related work covering VANETs security fundamentals, vehicular communication protocols, and methodologies for simulating attacks and constructing relevant datasets.

2.1 VANETs

VANETs were initially conceptualized in 2001 as a means for enabling dynamic, peer-to-peer communication among vehicles in a mobile ad-hoc environment [2]. According to [10], the security of such networks relies on eight fundamental principles, which are visually summarized in Figure 2.1. A cornerstone of secure communication in VANETs is the establishment of mutual authentication between participating nodes prior to any message exchange. Nonetheless, this security measure can be compromised in scenarios involving DoS attacks. Specifically, an entity possessing valid cryptographic credentials—such as a digitally signed certificate—can still initiate malicious activities post-authentication. This highlights a critical limitation of conventional signature-based methods, which are insufficient on their own to meet the comprehensive authentication demands described in Figure 2.1.

Message falsification constitutes a serious security risk in VANETs, undermining both data integrity and node authentication. In such attacks, adversaries exploit valid digital certificates to masquerade as trusted participants within the network. Once authenticated, they transmit misleading or fabricated messages to distort situational awareness. These falsified broadcasts may involve incorrect location data, manipulated speed values, or fabricated traffic events, all of which can disrupt normal vehicular coordination and potentially lead to dangerous driving decisions or unfair resource utilization [11]. These attacks may lead to route reconfiguration and, in severe cases, may even result in collisions. Therefore, as illustrated in Figure 2.1, it is crucial to ensure both data integrity and authentication.

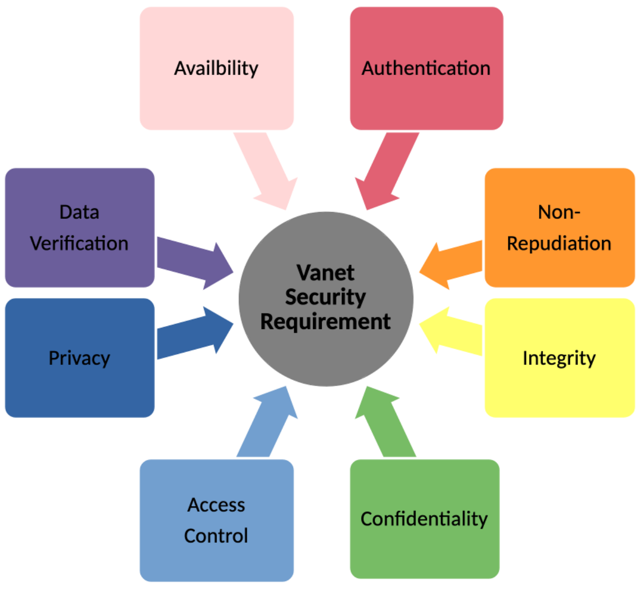


Figure 2.1: VANETs Security Requirements [10]

2.2 Vehicle Communication Technologies

Vehicle-to-Everything (V2X) communication [12] serves as a foundational technology for enabling real-time, low-latency data exchange between vehicles and with surrounding infrastructure. This communication paradigm aims to improve both traffic safety and operational efficiency by supporting timely coordination and hazard awareness. In recent years, V2X technology has advanced, rapidly primarily encompassing two major communication standards: Dedicated Short-Range Communications (DSRC) and Cellular Vehicle-to-Everything (C-V2X). These two standards exhibit individual characteristics in terms of technical architecture and application scenarios.

2.2.1 DSRC

The DSRC wireless communication system operates on the IEEE 802.11p protocol framework, specifically engineered for automotive communication scenarios that demand minimal latency and proximity-based data transmission capabilities. In support of vehicular safety applications, the Federal Communications Commission designated a dedicated 75 MHz spectrum allocation from the 5.850-5.925 GHz frequency band exclusively for DSRC communication systems [13]. Operating in the 5.9 GHz frequency band, DSRC facilitates direct communication between V2V and V2I.

A notable strength of DSRC lies in its independence from traditional cellular networks. This autonomy allows it to function reliably even in regions with limited or no cellular coverage, making it well-suited for latency-sensitive safety applications such as pedestrian alerts and collision avoidance. Prior research indicates that DSRC can achieve latencies on the order of milliseconds, enabling the prompt delivery of emergency messages and contributing significantly to road safety [14].

2.2.2 C-V2X

The Cellular Vehicle-to-Everything (C-V2X) protocol, established by the 3rd Generation Partnership Project (3GPP) [15], facilitates instantaneous data exchange among automobiles, roadway infrastructure, pedestrians, and communication networks. This technology functions through dual operational pathways: the PC5 interface providing direct, minimal-delay connectivity independent of cellular network infrastructure, and the Uu interface enabling extensive coverage communication through 4G LTE or 5G cellular systems.

By utilizing the advantages of 5G networks, including low latency and high data transmission rates, C-V2X can meet the demands of data-intensive applications such as high-definition map updates and real-time visual analysis. These capabilities significantly expand the scope of vehicular communication applications, enhancing both safety and efficiency in intelligent transportation systems [16].

2.2.3 Basic Safety Messages (BSMs) [17]

BSM is a standardized wireless message format used in VANETs that enables vehicles to broadcast critical safety information to nearby vehicles and infrastructure. Transmitted via DSRC or C-V2X technologies, BSMs contain data including vehicle position, speed, direction, acceleration, and dimensions. This communication system supports collision warnings, intersection assistance, and hazard alerts, significantly enhancing road safety by allowing vehicles and drivers to detect potential dangers beyond the capabilities of traditional sensors.

2.2.4 Vulnerabilities in Vehicular Communication Technologies

In [18], Sedar, Kalalas, Vázquez-Gallego, *et al.* present a comprehensive analysis of potential vulnerabilities in V2X, particularly in the context of increasing connectivity and the development of autonomous driving. As V2X technology expands to include V2V, V2I, and vehicle-to-pedestrian (V2P) communication, its unique system characteristics and diverse application scenarios introduce an extensive attack surface. Common security threats include, but are not limited to, message falsification, replay attacks, location spoofing, Sybil attacks, and DoS attacks. These threats can lead to vehicle navigation errors, traffic congestion, and even traffic accidents. Furthermore, privacy concerns in V2X communication have become increasingly prominent, as attackers may exploit or misuse driver location data and identity information. This study also explores advanced detection and defense mechanisms based on artificial intelligence (AI) and ML and categorizes and evaluates existing security measures. The research highlights that as V2X technology continues to be widely adopted, security requirements will become more stringent, necessitating innovative solutions to counter the escalating cybersecurity threats in future intelligent transportation systems.

In [19], Masood, Saeed, Ali, *et al.* propose a holistic framework to strengthen VANETs security by addressing both the detection of fake nodes and the identification of falsified messages in an integrated manner. In contrast to prior approaches that typically handle these issues in isolation, their method combines vehicle profile validatio leveraging attributes such as geographical location, vehicle ID and a reward-penalty mechanism to ensure message and node authenticity. By leveraging a mesh network structure and Roadside Units (RSUs) based verification, their system mitigates security risks, reduces network disconnection, and strengthens communication reliability. Validated through Opportunistic Network Environment simulator experiments, their results demonstrate effective detection and prevention of malicious entities, contributing to safer and more trustworthy vehicular communications for applications in traffic management, emergency response, and smart cities.

2.3 Attack Simulation and Dataset Generation

2.3.1 Attack Simulation

In [11], Lastinec and Keszeli investigate three major attack scenarios in V2V communication: replay attacks, message falsification, and Sybil attacks. To mitigate these threats, they propose the use of digital signatures to make sure message completeness and introduce a novel Sybil node detection method based on signal strength measurements. Their evaluation, conducted in the OMNeT++ [20] simulation environment, demonstrates the effectiveness of these mitigation strategies, providing practical approaches to enhancing the security of V2V communication.

In [21], Gyawali and Qian introduced a deep learning (DL) driven Misbehavior Detection System (MDS) tailored for VANETs. Their approach focuses on identifying abnormal behaviors in Connected and Autonomous Vehicles (CAVs) by examining two major attack scenarios. The first involves generating false alerts, malicious nodes disseminate deceptive messages, such as emergency braking signals, hazard reports, or collision warnings. The second pertains to location spoofing, wherein adversaries inject counterfeit positional data to impersonate multiple virtual entities with fabricated coordinates. To estimate the effectiveness of their model, the authors employed simulation-derived datasets, utilizing the Greenshield traffic model [22] to assess false alert scenarios and incorporating the VeReMi dataset [23] for identifying location falsification attacks. The experimental findings revealed that their method surpasses prior approaches in classification accuracy, recall, and its ability to handle multi-class attack types, showcasing its potential to bolster VANETs security.

2.3.2 Dataset Generation

In [24], Van Der Heijden, Lukaseder, and Kargl introduced the Vehicular Reference Misbehavior Dataset (VeReMi), an open-access dataset intended to support reproducible attack scenarios, benchmarking of detection techniques, and the development of ML models for misbehavior identification in VANETs. The dataset features position falsification attacks and includes simulations involving attackers of various densities across diverse traffic environments and scenario settings. Additionally, the authors evaluated several detection strategies such as Sudden Appearance Warning (SAW), Simple Speed Check (SSC), Acceptance Range Threshold (ART), and Distance Moved Verifier (DMV) and offering a detailed analysis of their detection capabilities and performance.

In [23], Kamel, Wolf, Van Der Hei, *et al.* offer an extended version of the VeReMi dataset aimed at advancing anomalous behavior detection in VANETs. This enhanced dataset incorporates a broader range of sophisticated attack types, a more practical physical error model, and an expanded volume of data. To establish a baseline, the research includes performance evaluations of several basic detection methods. The authors explore three primary attack vectors—replay attacks, message falsification, and Sybil attacks—and propose corresponding countermeasures. These include leveraging digital signature techniques for data integrity and a novel approach to Sybil node identification that utilizes signal strength analysis.

2.4 Machine Learning and VANETs

In [25], Feukeu and Mbuyu present a ML based approach to enhance link adaptation (LA) in VANETs environments. Given the highly dynamic characteristics of VANETs—such as fluctuating vehicle speeds, rapidly changing network topologies, and inconsistent wireless channel conditions—ensuring reliable and efficient communication becomes a significant challenge. To tackle this, the authors propose a multivariate linear regression model capable of predicting the optimal modulation and coding scheme (MCS) in real time. The model relies on key input features, including signal-to-noise ratio (SNR) and relative mobility speed. Simulation results reveal that the approach offers strong adaptability across diverse mobility patterns and communication conditions, delivering notable improvements in link reliability over conventional methods. The study also emphasizes the critical role of data preprocessing, particularly normalization and feature engineering, in maintaining the predictive accuracy and resilience of the model.

In [26], Abi Singh, Kamboj, and Kaur investigate the role of ML and DL in strengthening the outcome of VANETs, particularly within the context of Intelligent Transportation Systems (ITS). Their work highlights how various learning paradigms, such as supervised, unsupervised, and reinforcement learning, can be leveraged to address key challenges, including traffic congestion detection, route planning, and cybersecurity. The study also provides a comparative overview of frequently used VANETs datasets, pinpointing their limitations and underscoring the need for more comprehensive and representative data to drive future research. To enhance system adaptability and robustness, the authors advocate for hybrid learning approaches and outline potential directions for developing more efficient and scalable VANETs solutions.

In [27], Suganyadevi, Swathi, Santhiya, *et al.* present a ML driven traffic management framework for VANETs, targeting the economic and environmental issues arising from urban traffic congestion. The research examines the applicability of ML techniques to various aspects of vehicular networks, such as predicting traffic flow, forecasting accidents, and monitoring vehicle behaviors. By utilizing communication data exchanged between vehicles and Roadside Units (RSUs), the study assesses the effectiveness of multiple ML models including Naïve Bayes (NB), Random Forest (RF), K-Nearest Neighbor (KNN), and Support Vector Machine (SVM). Among these, RF is identified as the most effective for traffic flow prediction. The proposed system is capable of dynamically adjusting traffic light schedules, identifying congestion hotspots, and enhancing overall traffic fluidity. Moreover, the authors stress the importance of employing real-world datasets for model training and validation, offering practical insights to support the development of more efficient and intelligent transportation systems.

In [28], Kadam and Krovi introduced a hybrid classification approach termed KSVM that integrates SVM with KNN for the purpose of detecting Distributed Denial-of-Service (DDoS) attacks in VANETs environments. This approach combines the strong feature separation capabilities of SVM with the neighborhood-based decision logic of KNN to improve classification performance. The proposed model was trained on a DDoS dataset sourced from Kaggle and estimated using various key metrics, including sensitivity, precision, recall, accuracy, and error rate. Experimental findings demonstrated that KSVM achieved superior results compared to conventional ML algorithms, indicating its effectiveness in identifying malicious IP traffic and its potential utility in bolstering VANETs security through intelligent detection techniques.

In [29], Hameed and Saini introduced a hybrid detection architecture that integrates DL with conventional ML methods to classify multiple types of DoS attacks within vehicular networks. Leveraging the extended VeReMi dataset, they compiled a comprehensive dataset covering five distinct DoS variants and implemented composite models such as CNN with Decision Tree (CNN+DT) and CNN+RF for multiclass classification. The proposed framework is specifically designed to be lightweight, ensuring minimal execution time and memory consumption —an essential consideration for deployment on resource-limited Onboard Units (OBUs). Of the evaluated models, CNN+DT achieved the most favorable trade-off between classification accuracy, F1-score, and computational efficiency, positioning it as a practical solution for real-time DoS detection in VANETs scenarios.

In [30], Kumar, Shahid, Jaekel, *et al.* introduce a ML-based detection framework aimed at identifying replay attacks in VANETs environments. The study specifically addresses attacks targeting BSMs and proposes a Misbehavior Detection System (MDS) designed to complement existing cryptographic defenses by providing an additional security layer. Utilizing simulation data derived from the VeReMi extension dataset [23], the framework demonstrates strong performance in distinguishing genuine messages from those retransmitted with malicious intent. Key evaluation metrics such as recall, accuracy, F1-score, and precision reflect the system's high effectiveness. The research also investigates multiple ML algorithms, including RF and XGBoost, and refines feature selection strategies to minimize dependency on sender identity information. Comprehensive testing across various traffic scenarios further confirms the framework's potential to enhance VANETs security in dynamic conditions.

2.5 Transformer

In [31], Vaswani, Shazeer, Parmar, *et al.* presented the Transformer model, a novel architecture for sequence modeling that eliminates the need for recurrence and convolution by relying solely on attention mechanisms. Through the use of self-attention and multi-head attention, the Transformer achieves efficient parallelization and excels at modeling long-range dependencies in sequential inputs. Experimental evaluations demonstrated that the model not only surpassed previous approaches in tasks like machine translation but also established new performance standards in terms of BLEU scores. Additionally, the integration of positional encodings allows the model to maintain awareness of token order, compensating for the lack of recurrent structure and enabling it to capture both syntactic and semantic relationships effectively.

In [32], Ullah, Ullah, Srivastava, *et al.* proposed a refined Transformer-based intrusion detection model, referred to as IDS-INT, designed specifically for intelligent transportation system environments. By integrating multi-head self-attention with a tailored deep architecture, IDS-INT is capable of effectively learning spatiotemporal patterns within network traffic, thereby enabling accurate identification of diverse attack types. The model was validated using standard benchmark datasets and exhibited outstanding outcome across multiple metrics, including recall, accuracy, and F1-score, outperforming conventional ML and DL techniques. This study highlights the promise of Transformer-based architectures in advancing the robustness and flexibility of intrusion detection mechanisms within vehicular networks.

In [33], Shuvro, Khan, Rahman, *et al.* introduced a Transformer-based forecasting model aimed at predicting traffic flow within Software Defined Networking enabled VANETs (SDN-VANETs). The proposed method leverages the self-attention mechanism of the Transformer to capture intricate temporal patterns and long-range dependencies within traffic data, while avoiding the structural constraints typically associated with recurrent and convolutional networks. Evaluation results revealed that this model surpasses conventional methods, such as RNN and LSTM, in both predictive accuracy and resilience, underscoring the effectiveness of

Transformer architectures in handling the ever-evolving and varied conditions present in vehicular communication systems.

2.6 XGBoost

In [34], Chen and Guestrin presented XGBoost, a highly efficient and scalable gradient boosting library tailored for large-scale ML tasks. This framework is engineered to deliver both high computational efficiency and strong predictive performance. Key features include support for parallelized tree construction, pruning techniques, and built-in regularization to reduce the risk of overfitting. Owing to its robustness, accuracy, and versatility, XGBoost has become a popular choice across a wide range of use cases such as regression, classification, and ranking problems.

In [35], Amaouche, AzidineGuezzaz, Benkirane, *et al.* proposed IDS-XGbFS, an intelligent intrusion detection system tailored for VANETs security, built upon the XGBoost learning algorithm. To improve detection performance and address class imbalance, the framework integrates Boruta for feature selection and applies the ADASYN oversampling technique. The model's effectiveness was assessed using two benchmark datasets—NSL-KDD and 5Routing-Metrics. Experimental evaluations revealed that IDS-XGbFS consistently outperformed competing ML models, including CNN and CatBoost, in terms of precision, accuracy, and recall, confirming its suitability for practical deployment in vehicular network intrusion detection scenarios.

In [36], Li, Song, Zheng, *et al.* presents FEXGBIDS, a privacy-preserving intrusion detection system for in-vehicle networks that combines federated XGBoost with cryptographic optimization techniques. The system employs BatchCrypt algorithm to batch encrypt gradients, reducing computational overhead by 65% compared to traditional Paillier encryption, while using BLS signatures for secure parameter aggregation and DDSketch for efficient feature bucketing. Evaluation on the Car Hacking dataset demonstrates that FEXGBIDS attains a detection accuracy of 97.26%, closely matching the performance of centralized XGBoost models. Moreover, the framework exhibits strong resilience against adversarial attacks and is well-suited for scalable deployment within Internet of Vehicles (IoV) environments.

Chapter 3 Methodology

This chapter presents the methodology of our hybrid learning-based detection model for DoS attacks in VANETs. It outlines the simulation setup, data preprocessing pipeline, feature selection, and model architecture. Using SUMO as the simulation framework, vehicular behavior is logged and exported as structured data, which is then analyzed externally to detect abnormal message patterns representing potential DoS attacks.

3.1 System Architecture

The TransBoost-DoS system is designed to DoS-related attacks in VANETs using a combination of realistic traffic simulation, structured behavioral logging, and a hybrid deep learning detection pipeline. The system is composed of four modular components: the topology generation module, the behavior packets generation module, the attack detection module and report module. Each component plays a distinct role in enabling an end-to-end workflow from synthetic behavior generation to post-simulation threat identification. The entire simulation and logging pipeline is implemented using the SUMO platform, python and Traffic Control Interface (TraCI), while detection is performed using a pre-trained TransBoost-DoS model. Figure 3.1 illustrates the complete system workflow from SUMO simulation to TransBoost-DoS detection.

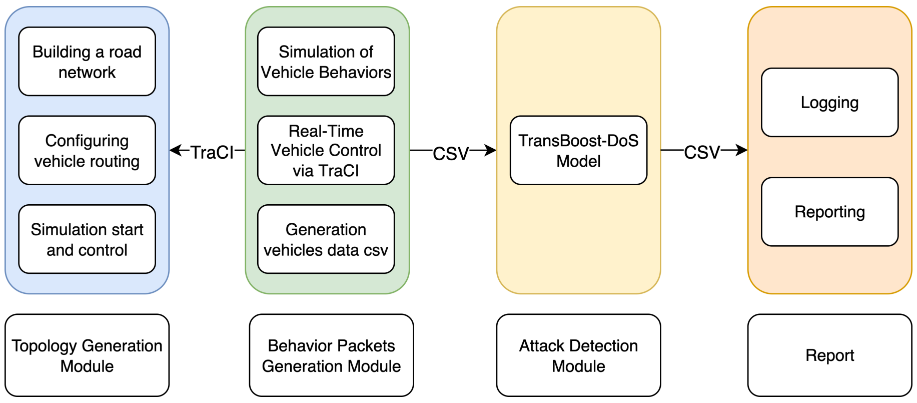

Building a road network
Configuring vehicle routing
Simulation start and control
TraCI
Simulation of Vehicle Behaviors
Real-Time Vehicle Control via TraCI
Generation vehicles data csv
CSV
TransBoost-DoS Model
CSV
Logging
Reporting
Topology Generation Module
Behavior Packets Generation Module
Attack Detection Module
Report

Figure 3.1: TransBoost-DoS system architecture

3.1.1 Topology Generation Module

The topology generation module is responsible for creating the road network and defining vehicle routes for the simulation scenario. This module is implemented using SUMO, which supports microscopic-level mobility control and dynamic vehicle behavior customization. The road network is defined in a `.net.xml` file and consists of six parallel, unidirectional roads, each represented by a single edge. Each road is 150 meters long, contains one lane, and supports a maximum speed of 13.89 m/s.

Each vehicle flow is configured in the `.rou.xml` file. The simulation uses six flow definitions, each lasting for 120 seconds with a generation rate of 1200 vehicles per hour. Distinct color values are applied to each flow for visualization, aiding post-simulation analysis.

This design provides a highly controlled yet behaviorally diverse environment, enabling precise analysis of each attack type's impact on communication and mobility patterns. Additionally, the clear lane separation ensures that feature extraction and classification are based purely on behavior, not collision or congestion interference.

3.1.2 Behavior Packages Generation Module

The Behavior Packets Generation Module is responsible for simulating vehicle behavior and generating communication-related data packets under both normal and attack conditions. This module acts as the core of scenario dynamics, directly influencing the quality and diversity of training data for the detection system.

To emulate realistic vehicular network interactions, the module integrates SUMO simulation with TraCI for real-time vehicle control. Through Python scripts and application-level logic, each vehicle's behavior can be dynamically adjusted based on its role.

All vehicle status data, including timestamp, vehicle ID, speed, position, heading, lane ID, and other dynamic fields, are recorded at each simulation step and exported in CSV format. These logs form the basis of the feature set used for downstream classification by the detection model.

3.1.3 Attack Detection Module

After the simulation is completed, the logged vehicle behavior data is segmented into overlapping time-series samples using a sliding window of fixed size. These samples are standardized using a pre-fitted `StandardScaler`, which transforms each feature by removing the mean and scaling to unit variance. This normalization step ensures that features with varying magnitudes are on a comparable scale, thereby enhancing the stability and performance of the detection model. The standardized samples are then fed into a pretrained TransBoost-DoS model for inference.

The prediction results for each window are saved into a detection report file in CSV format, enabling post-analysis, visualization, and forensic inspection. This offline detection architecture ensures modularity, allowing new detection models to be integrated seamlessly without altering the simulation pipeline.

This inference-only architecture eliminates the need for online training and guarantees consistent, reproducible results across various simulation scenarios. A detailed explanation of the model architecture is provided in the next section.

3.1.4 Report Module

The report module is responsible for generating a comprehensive detection report based on the results from the attack detection module. It compiles the predictions for each sliding window, including the predicted class label and confidence score, into a structured CSV file. This report serves as the final output of the TransBoost-DoS system, providing a clear summary of the detection results. The report can be used for further analysis, visualization, and forensic inspection of the detected attacks.

3.2 Model Architecture

The overall architecture of the proposed TransBoost-DoS model is illustrated in Figure 3.2. To effectively DoS attacks in highly dynamic vehicular environments, the proposed TransBoost-

DoS system adopts a hybrid model architecture that combines a Transformer [31] temporal feature extractor with an XGBoost [34] classifier. This architectural approach takes advantage of the synergistic benefits offered by combining deep learning with conventional ML techniques.

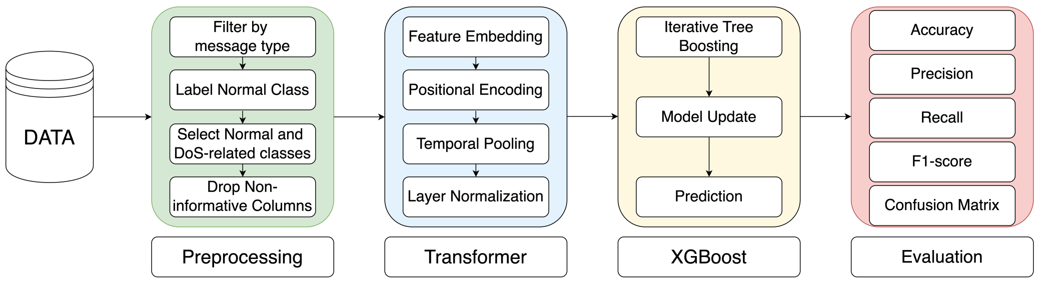


Figure 3.2: TransBoost-DoS model architecture

Transformers excel at handling sequential data by employing self-attention mechanisms, which allow them to model long-range dependencies and intricate temporal dynamics in vehicular behavior. This capability facilitates the extraction of informative representations from communication sequences, aiding in the identification of anomalous patterns.

XGBoost complements this by efficiently handling structured data with strong generalization and interpretability. As a second-stage classifier, it delivers fast, accurate decisions and offers feature importance insights.

Table 3.1 summarizes the key motivations behind integrating these two models. It highlights their respective weaknesses, how one complements the other, and the resulting advantages offered by the hybrid approach.

Table 3.1: Complementarity of Transformer and XGBoost in hybrid architecture

Weakness	**Model Affected**	**Complement by**	**Hybrid Advantage**
Cannot model temporal dependencies	XGBoost	Transformer	Enables modeling of sequential behaviors and time-dependent attack patterns
Needs manual feature engineering	XGBoost	Transformer	Learns high-level representations directly from raw sequences, reducing preprocessing efforts
High computational cost and less interpretability	Transformer	XGBoost	Offers fast, lightweight, and interpretable decisions using learned Transformer features
Difficult to deploy on edge devices due to model size	Transformer	XGBoost	Reduces resource demands via shallow trees, enabling real-time inference on OBUs

To offer a thorough explanation of the proposed TransBoost-DoS detection system, this section details each component of the model architecture. We begin by describing the data input and preprocessing pipeline, which prepares vehicular communication messages for temporal feature extraction. Subsequently, we present the Transformer-based feature extractor and the XGBoost-based classification module, outlining their respective roles and functionalities within the hybrid learning framework.

3.2.1 Data Preprocessing

To initiate the training process, we first loaded the raw dataset `Veremi_dataset.csv` [37] and performed data cleaning and filtering. Figure 3.3 illustrates the raw dataset. Initially, only rows with `type = 3` were retained, as these entries represent messages received from other vehicles, which are more indicative of attack-related behaviors. Subsequently, all entries with `attack = 0` were relabeled as `attack_type = "Normal"`, designating them as benign communication instances.

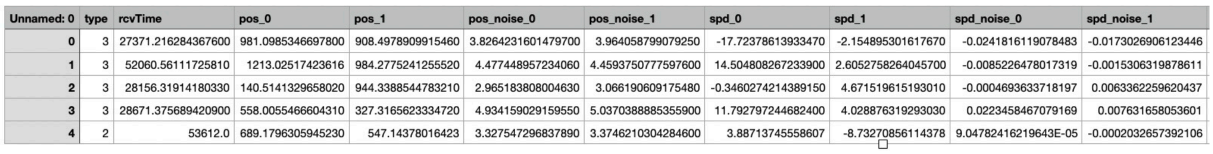
After labeling, we filtered the dataset to retain only samples with `attack_type` belonging to six target categories: `Normal`, `DoS`, `DoS disruptive`, `DoS disruptive sybil`, `DoS random`, and `DoS random sybil`. This ensures that the final dataset includes both malicious and benign samples specifically relevant to the DoS detection task.

In terms of feature selection, we removed non-informative or redundant columns such as `Unnamed: 0`, `type`, and `attack`, which do not contribute to the learning objective. This preprocessing stage ensures that the model is trained and evaluated using only the most meaningful and discriminative features. The final column, `attack_type`, serves as the ground-truth label indicating the class of each instance.

Figure 3.4 illustrates the preprocessing flowchart, which outlines the steps taken to prepare the dataset for training.

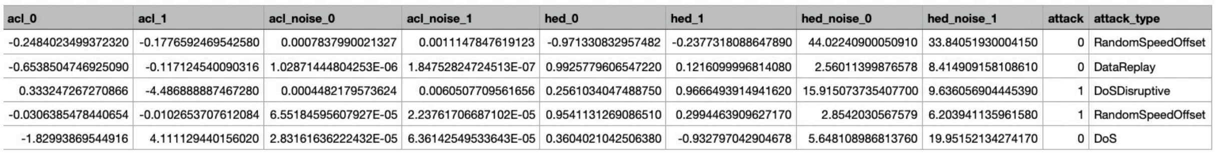
3.2.2 Transformer-based Feature Extraction [31]

This module is responsible for encoding the temporal dependencies in the BSMs sequences. It consists of the following stages:



Unnamed: 0	type	rcvTime	pos_0	pos_1	pos_noise_0	pos_noise_1	spd_0	spd_1	spd_noise_0	spd_noise_1
0	3	27371.216284367600	981.0985346697800	908.4978909915460	3.8264231601479700	3.964058799079250	-17.72378613933470	-2.154895301617670	-0.0241816119078483	-0.0173026906123446
1	3	52060.56111725810	1213.02517423616	984.2775241255520	4.477448957234060	4.4593750777597600	14.504808267233900	2.6052758264045700	-0.0085226478017319	-0.0015306319878611
2	3	28156.31914180330	140.5141329658020	944.3388544783210	2.965183808004630	3.066190609175480	-0.3460274214389150	4.671519615193010	-0.0004693633718197	0.0063362259620437
3	3	28671.375689420900	558.0055466604310	327.3165623334720	4.934159029159550	5.0370388885355900	11.792797244682400	4.028876319293030	0.0223458467079169	0.007631658053601
4	2	53612.0	689.1796305945230	547.14378016423	3.327547296837890	3.3746210304284600	3.88713745558607	-8.73270856114378	9.04782416219643E-05	-0.0002032657392106

Figure 3.3 (a)



acl_0	acl_1	acl_noise_0	acl_noise_1	hed_0	hed_1	hed_noise_0	hed_noise_1	attack	attack_type
-0.2484023499372320	-0.1776592469542580	0.0007837990021327	0.0011147847619123	-0.971330832957482	-0.2377318088647890	44.02240900050910	33.84051930004150	0	RandomSpeedOffset
-0.6538504746925090	-0.117124540090316	1.02871444804253E-06	1.84752824724513E-07	0.9925779606547220	0.1216099996814080	2.56011399876578	8.414909158108610	0	DataReplay
0.333247267270866	-4.486888887467280	0.0004482179573624	0.0060507709561656	0.2561034047488750	0.9666493914941620	15.915073735407700	9.636056904445390	1	DoSDisruptive
-0.0306385478440654	-0.0102653707612084	6.55184595607927E-05	2.23761706687102E-05	0.9541131269086510	0.2994463909627170	2.8542030567579	6.203941135961580	1	RandomSpeedOffset
-1.82993869544916	4.111129440156020	2.83161636222432E-05	6.36142549533643E-05	0.3604021042506380	-0.932797042904678	5.648108986813760	19.95152134274170	0	DoS

Figure 3.3 (b)

Figure 3.3: Raw dataset

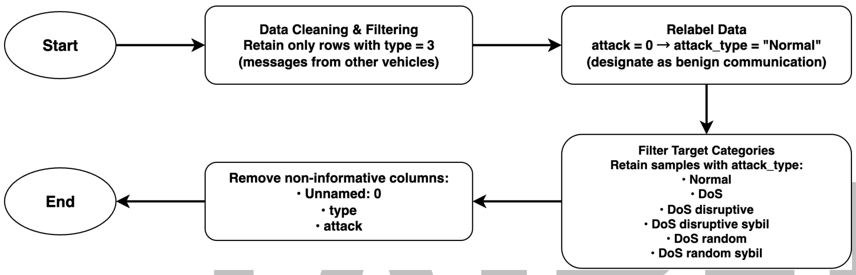

Start
Data Cleaning & Filtering
Retain only rows with type = 3
(messages from other vehicles)
Relabel Data
attack = 0 → attack_type = "Normal"
(designate as benign communication)
Filter Target Categories
Retain samples with attack_type:
• Normal
• DoS
• DoS disruptive
• DoS disruptive sybil
• DoS random
• DoS random sybil
Remove non-informative columns:
• Unnamed: 0
• type
• attack
End

Figure 3.4: Data preprocessing flowchart

- Feature Embedding: Projects raw input features into a higher-dimensional space for subsequent processing.
- Positional Encoding: Adds learnable position embeddings to retain temporal order within the input sequence.
- Temporal Pooling: Aggregates the transformer outputs using mean pooling across the time dimension.
- Layer Normalization: Applies normalization to stabilize and generalize the final encoded vector.

The output is a compact, fixed-size semantic vector representing the behavioral patterns in the input BSMs window.

3.2.3 XGBoost-based Attack Classification [34]

The semantic feature vector is passed to the XGBoost classifier, which performs:

- Iterative Tree Boosting: Forms a sequence of trees, each tree focuses on minimizing the errors made by its predecessors.
- Model Update: Refines the predictions using gradient and hessian information derived from classification loss.
- Prediction: Outputs a final predicted class label that corresponds to a specific classes.

This hybrid design leverages the sequence modeling power of Transformers and the structured classification strength of XGBoost to deliver effective and interpretable detection of complex attacks in VANETs.

3.2.4 Evaluation

To facilitate analysis and benchmarking, the model architecture is designed to allow modular evaluation. Both the Transformer and XGBoost components can be independently assessed and compared with baseline variants. This structure supports consistent evaluation across experiments. A detailed performance analysis is provided in Section 4.2.4.

Chapter 4 Implementation

This chapter presents the detailed implementation of the proposed TransBoost-DoS system. We begin with the environment setup, covering both the hardware specifications and the software tools required to support the simulation and machine learning components. Next, we delve into the model implementation details, including the development of the Transformer and XGBoost-based detection pipeline, along with the data preprocessing, training, and evaluation procedures. Following this, we introduce thethe design of the attack detection scenario, which defines the road topology, vehicle behaviors, and attack patterns used throughou the simulation. Finally, we elaborate on the integration of our system into the SUMO simulation for realistic vehicular network testing.

4.1 Setup

4.1.1 Hardware and Software Requirements

Table 4.1 outlines our hardware requirements. The computer we use to execute the code is furnished with a 12th Gen Intel(R) Core(TM) i7-12700 CPU, an NVIDIA RTX 3080Ti 12GB GPU, and 64GB of DDR4-3400MHz memory.

Table 4.1: Hardware requirements

Hardware Type	Name
CPU	12th Gen Intel(R) Core(TM) i7-12700
GPU	NVIDIA RTX 3080Ti 12GB
Memory	DDR4-3400MHz 64GB

Table 4.2 outlines our software requirements. Our code execution environment is Windows 10, and we use Python 3.13.2 as the programming language. The simulation framework used is SUMO 1.23.1 , which is a widely used open-source tool for simulating vehicular networks.

Table 4.3 outlines the main Python packages used in our implementation.

Table 4.2: Software requirements

Software Type	Software Name	Version
Operating System	Windows [38]	10
Programming Language	Python [39]	3.13.2
Simulation framework	SUMO [9]	1.23.1

Table 4.3: Main python packages used

Package	Version	Purpose	License
pytorch-gpu [40]	2.6.0	Deep learning framework used for Transformer model	BSD
xgboost-gpu [34]	2.1.4	Gradient boosting classifier used for final classification.	Apache 2.0
seaborn [41]	0.13.2	Statistical data visualization.	BSD
matplotlib [42]	3.10.0	Data Visualization.	PSF
scikit-learn [43]	1.6.1	Preprocessing,evaluation and cross-validation.	BSD
pandas [44]	2.2.3	Data manipulation and CSV reading.	BSD
numpy [45]	2.2.5	Numerical operations and array handling.	BSD
json [46]	Built-in	JSON encoding and decoding for saving reports.	PSF
os [47]	Built-in	File handling and system operations.	PSF

4.1.2 Environment Setup

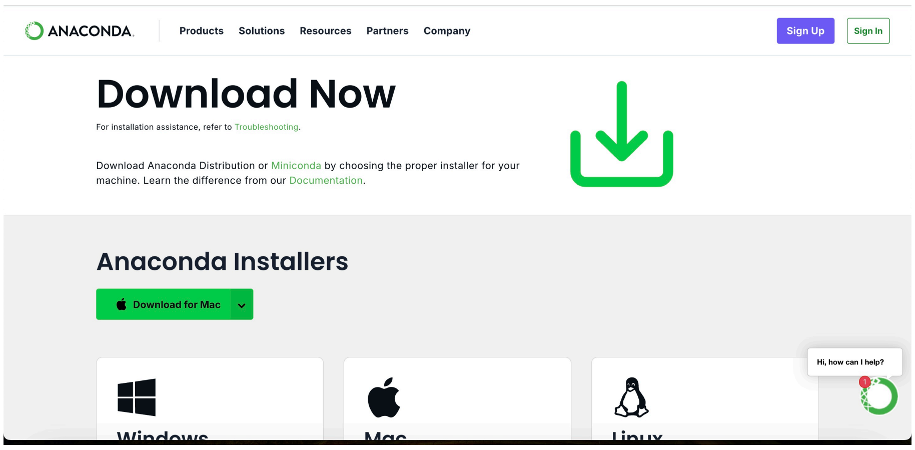
This section outlines the proper setup of the environment used for our experiment.To begin, we utilized Anaconda as our package management tool for the implementation. Therefore, you should first visit the official Anaconda website (`https://www.anaconda.com/download`) to download the software. As illustrated in Figure 4.1, click the ”Download” button to initiate the download.

When the installation is complete, a new environment must be created to maintain the independence of all packages. To do this, open the Anaconda Prompt and enter

```
conda create --name environment-name
```

to create a new environment, as illustrated in Figure 4.2.
Then to activate the environment by entering similar to the first command shown in Figure 4.3.

```
conda activate environment-name
```



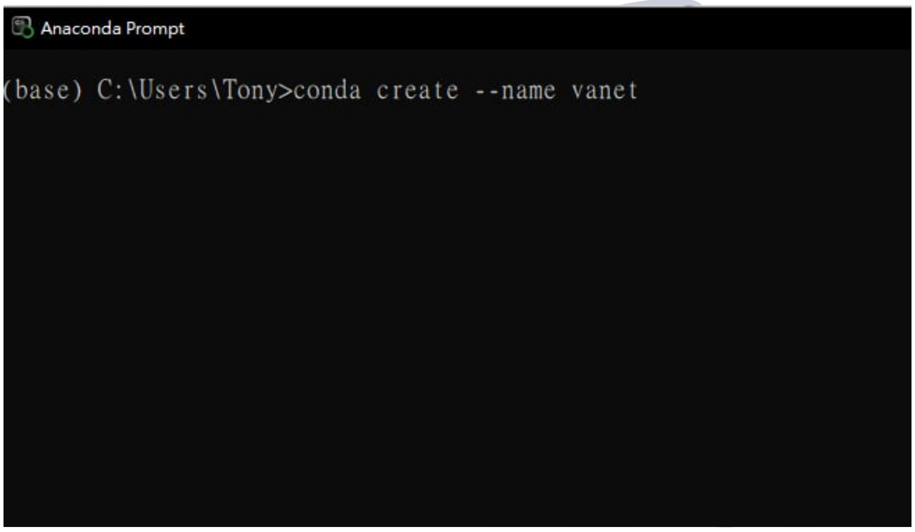
Download Now

For installation assistance, refer to Troubleshooting.

Download Anaconda Distribution or Miniconda by choosing the proper installer for your machine. Learn the difference from our Documentation.

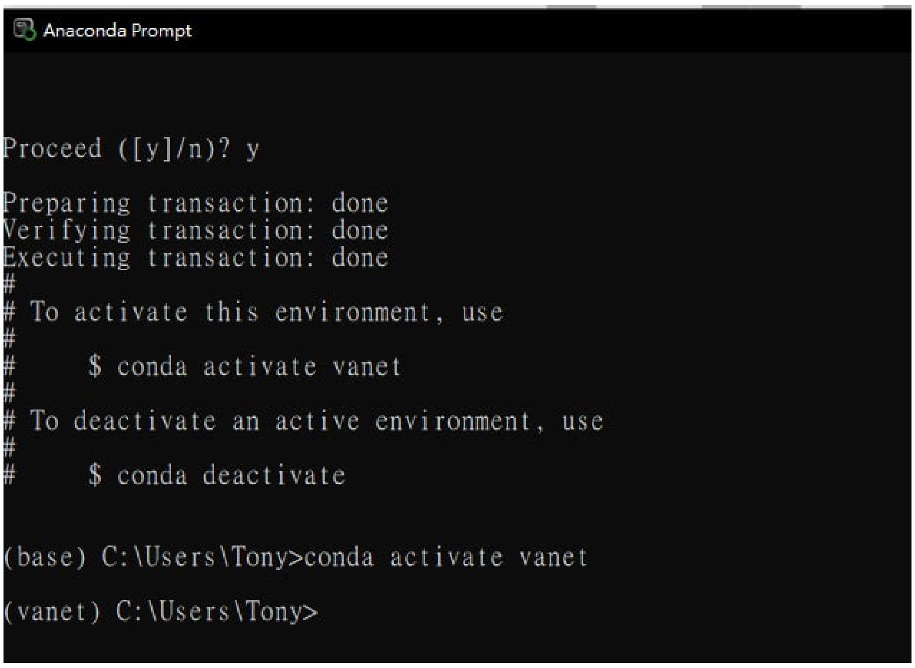
Anaconda Installers

Figure 4.1: Anaconda official website



```
(base) C:\Users\Tony>conda create --name vanet
```

Figure 4.2: Anaconda for creating new environment



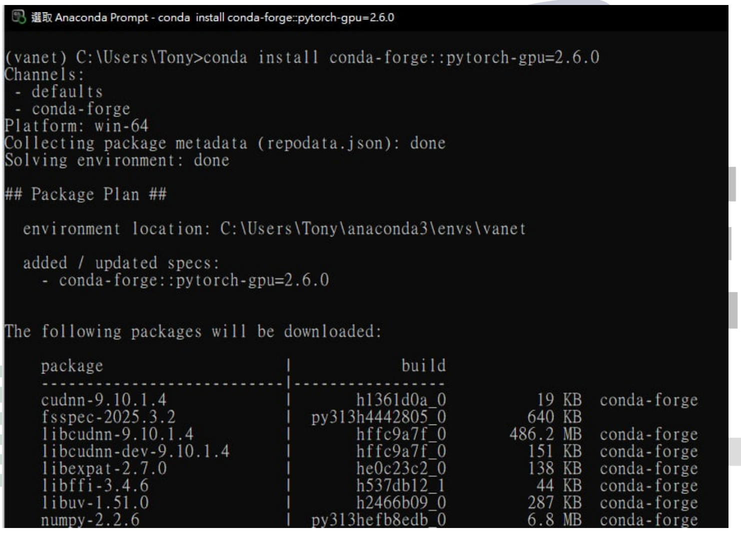
```
Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate vanet
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(base) C:\Users\Tony>conda activate vanet

(vanet) C:\Users\Tony>
```

Figure 4.3: Anaconda for activating environment

Next, use the following command to install the remaining packages in the enabled environment:

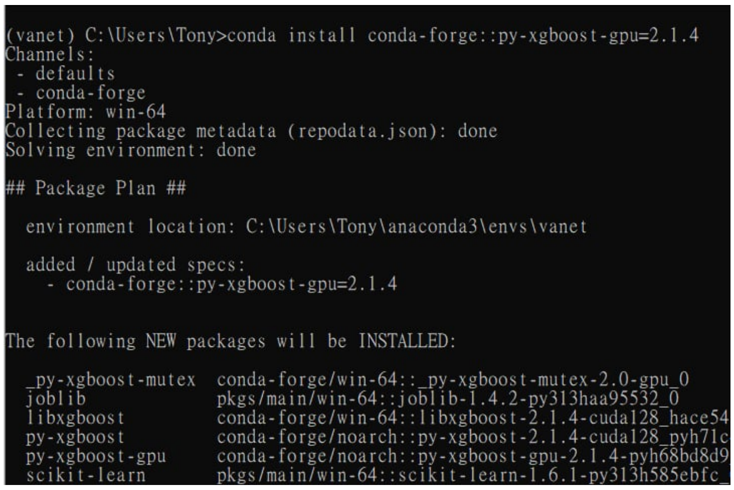
```
conda install conda-forge::pytorch-gpu=2.6.0

conda install conda-forge::py-xgboost-gpu=2.1.4

conda install seaborn=0.13.2 matplotlib=3.10.0 pandas=2.2.3
```

Figure 4.4 , 4.5 and 4.6 illustrates the installation process.



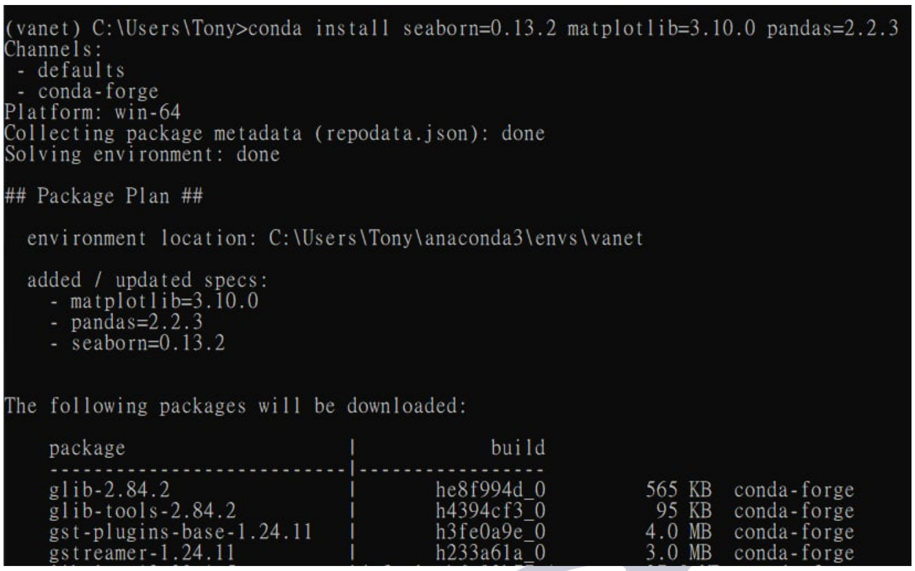
```
選取 Anaconda Prompt - conda  install conda-forge::pytorch-gpu=2.6.0

(vanet) C:\Users\Tony>conda install conda-forge::pytorch-gpu=2.6.0
Channels:
 - defaults
 - conda-forge
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Tony\anaconda3\envs\vanet

  added / updated specs:
    - conda-forge::pytorch-gpu=2.6.0


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    cudnn-9.10.1.4             |       h1361d0a_0          19 KB  conda-forge
    fsspec-2025.3.2            |  py313h4442805_0         640 KB
    libcudnn-9.10.1.4          |       hffc9a7f_0       486.2 MB  conda-forge
    libcudnn-dev-9.10.1.4      |       hffc9a7f_0         151 KB  conda-forge
    libexpat-2.7.0             |       he0c23c2_0         138 KB  conda-forge
    libffi-3.4.6               |       h537db12_1          44 KB  conda-forge
    libuv-1.51.0               |       h2466b09_0         287 KB  conda-forge
    numpy-2.2.6                |  py313hefb8edb_0         6.8 MB  conda-forge
```

Figure 4.4: Anaconda for installing PyTorch



```
(vanet) C:\Users\Tony>conda install conda-forge::py-xgboost-gpu=2.1.4
Channels:
 - defaults
 - conda-forge
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Tony\anaconda3\envs\vanet

  added / updated specs:
    - conda-forge::py-xgboost-gpu=2.1.4


The following NEW packages will be INSTALLED:

  _py-xgboost-mutex  conda-forge/win-64::_py-xgboost-mutex-2.0-gpu_0
  joblib             pkgs/main/win-64::joblib-1.4.2-py313haa95532_0
  libxgboost         conda-forge/win-64::libxgboost-2.1.4-cuda128_hace54
  py-xgboost         conda-forge/noarch::py-xgboost-2.1.4-cuda128_pyh71c
  py-xgboost-gpu     conda-forge/noarch::py-xgboost-gpu-2.1.4-pyh68bd8d9
  scikit-learn       pkgs/main/win-64::scikit-learn-1.6.1-py313h585ebfc_
```

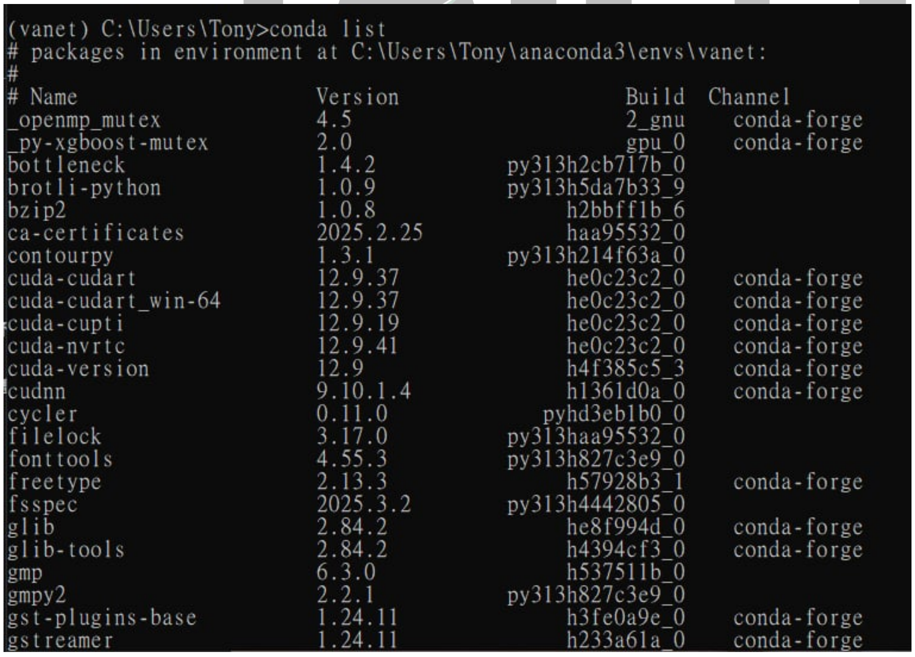
Figure 4.5: Anaconda for installing xgboost



```
(vanet) C:\Users\Tony>conda install seaborn=0.13.2 matplotlib=3.10.0 pandas=2.2.3
Channels:
 - defaults
 - conda-forge
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Tony\anaconda3\envs\vanet

  added / updated specs:
    - matplotlib=3.10.0
    - pandas=2.2.3
    - seaborn=0.13.2


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    glib-2.84.2                |       he8f994d_0         565 KB  conda-forge
    glib-tools-2.84.2          |       h4394cf3_0          95 KB  conda-forge
    gst-plugins-base-1.24.11   |       h3fe0a9e_0         4.0 MB  conda-forge
    gstreamer-1.24.11          |       h233a61a_0         3.0 MB  conda-forge
```

Figure 4.6: Anaconda for installing packages

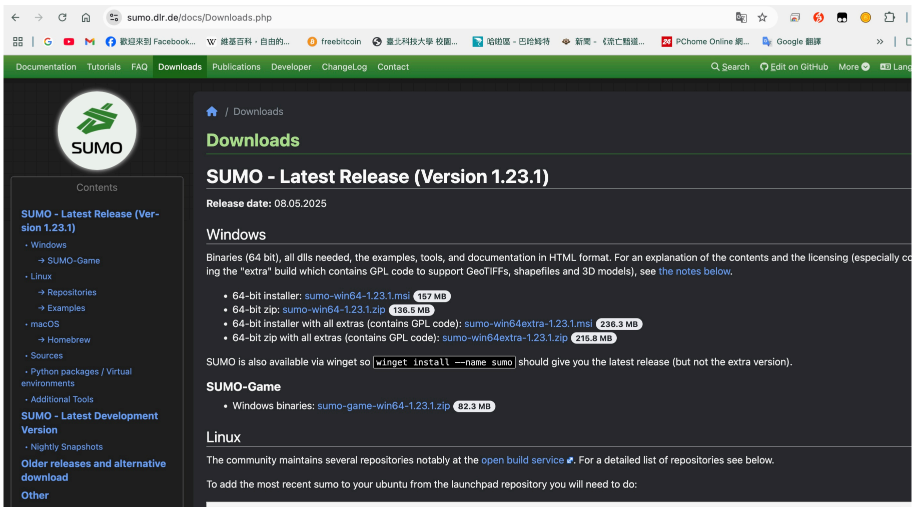
Finally, we use the commmand `conda list` to verify that all packages have been successfully installed. The verification procedure is depicted in Figure 4.7.



```
(vanet) C:\Users\Tony>conda list
# packages in environment at C:\Users\Tony\anaconda3\envs\vanet:
#
# Name                    Version                   Build  Channel
_openmp_mutex             4.5                       2_gnu    conda-forge
_py-xgboost-mutex         2.0                       gpu_0    conda-forge
bottleneck                1.4.2           py313h2cb717b_0
brotli-python             1.0.9           py313h5da7b33_9
bzip2                     1.0.8                h2bbff1b_6
ca-certificates           2025.2.25            haa95532_0
contourpy                 1.3.1           py313h214f63a_0
cuda-cudart               12.9.37              he0c23c2_0    conda-forge
cuda-cudart_win-64        12.9.37              he0c23c2_0    conda-forge
cuda-cupti                12.9.19              he0c23c2_0    conda-forge
cuda-nvrtc                12.9.41              he0c23c2_0    conda-forge
cuda-version              12.9                 h4f385c5_3    conda-forge
cudnn                     9.10.1.4             h1361d0a_0    conda-forge
cycler                    0.11.0             pyhd3eb1b0_0
filelock                  3.17.0          py313haa95532_0
fonttools                 4.55.3          py313h827c3e9_0
freetype                  2.13.3               h57928b3_1    conda-forge
fsspec                    2025.3.2        py313h4442805_0
glib                      2.84.2               he8f994d_0    conda-forge
glib-tools                2.84.2               h4394cf3_0    conda-forge
gmp                       6.3.0                h537511b_0
gmpy2                     2.2.1           py313h827c3e9_0
gst-plugins-base          1.24.11              h3fe0a9e_0    conda-forge
gstreamer                 1.24.11              h233a61a_0    conda-forge
```

Figure 4.7: Anaconda for listing installed packages

To simulate the vehicular environment, we also need to install SUMO. To do this, visit the official SUMO website (`https://sumo.dlr.de/docs/Downloads.php`) and download the software. As illustrated in Figure 4.8, and choose the download method based on specific operating system.


sumo.dlr.de/docs/Downloads.php
Documentation Tutorials FAQ Downloads Publications Developer ChangeLog Contact
Search Edit on GitHub More
SUMO
Contents

SUMO - Latest Release (Version 1.23.1)
- Windows
  - → SUMO-Game
- Linux
  - → Repositories
  - → Examples
- macOS
  - → Homebrew
- Sources
- Python packages / Virtual environments
- Additional Tools

SUMO - Latest Development Version
- Nightly Snapshots

Older releases and alternative download

Other

Downloads

SUMO - Latest Release (Version 1.23.1)

Release date: 08.05.2025

Windows

Binaries (64 bit), all dlls needed, the examples, tools, and documentation in HTML format. For an explanation of the contents and the licensing (especially co
ing the "extra" build which contains GPL code to support GeoTIFFs, shapefiles and 3D models), see the notes below.

- 64-bit installer: sumo-win64-1.23.1.msi 157 MB
- 64-bit zip: sumo-win64-1.23.1.zip 136.5 MB
- 64-bit installer with all extras (contains GPL code): sumo-win64extra-1.23.1.msi 236.3 MB
- 64-bit zip with all extras (contains GPL code): sumo-win64extra-1.23.1.zip 215.8 MB

SUMO is also available via winget so `winget install --name sumo` should give you the latest release (but not the extra version).

SUMO-Game

- Windows binaries: sumo-game-win64-1.23.1.zip 82.3 MB

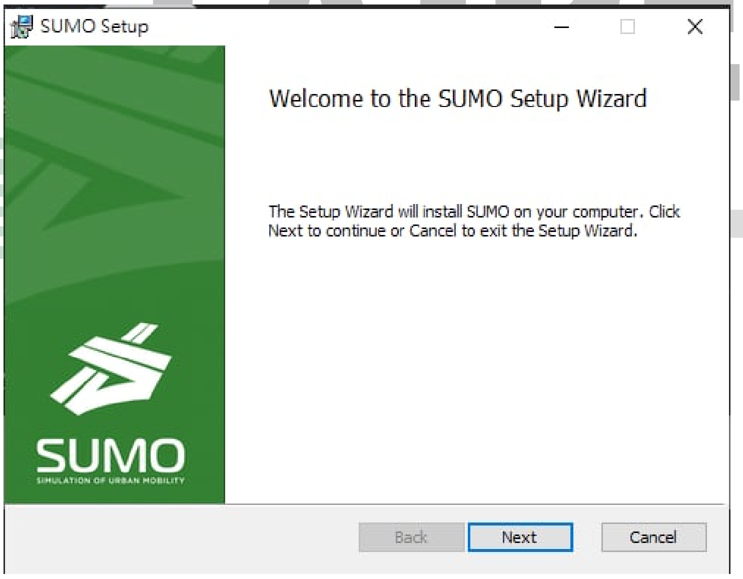
Linux

The community maintains several repositories notably at the open build service. For a detailed list of repositories see below.

To add the most recent sumo to your ubuntu from the launchpad repository you will need to do:

Figure 4.8: SUMO official website

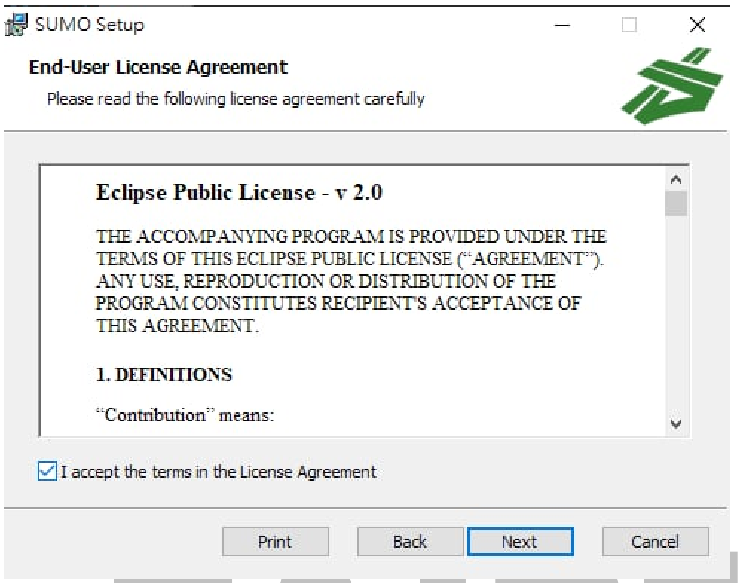
After downloading, follow the installation instructions provided on the SUMO website to complete the setup. Figure 4.9 to Figure 4.14 illustrates the installation process.



Welcome to the SUMO Setup Wizard

The Setup Wizard will install SUMO on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Figure 4.9: SUMO installer Initial screen


SUMO Setup

End-User License Agreement

Please read the following license agreement carefully

Eclipse Public License - v 2.0

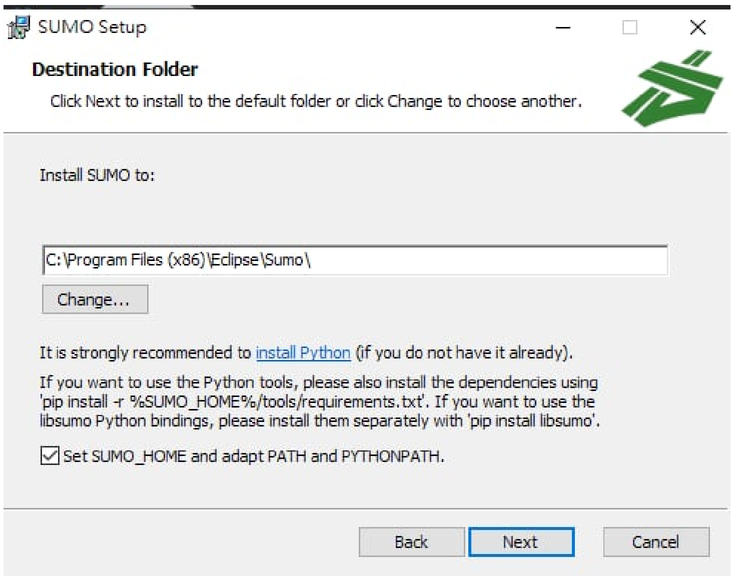
THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

☑ I accept the terms in the License Agreement

Figure 4.10: SUMO installer License Agreement


SUMO Setup

Destination Folder

Click Next to install to the default folder or click Change to choose another.

Install SUMO to:

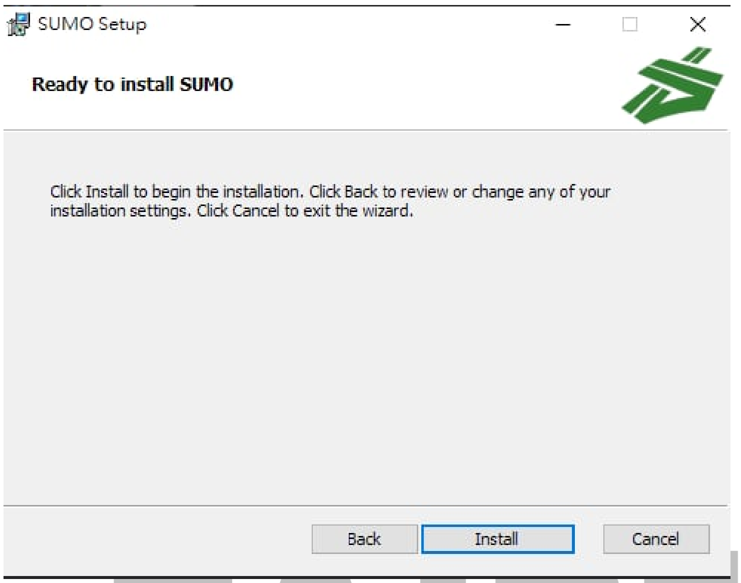
C:\Program Files (x86)\Eclipse\Sumo\

It is strongly recommended to install Python (if you do not have it already).

If you want to use the Python tools, please also install the dependencies using 'pip install -r %SUMO_HOME%/tools/requirements.txt'. If you want to use the libsumo Python bindings, please install them separately with 'pip install libsumo'.

☑ Set SUMO_HOME and adapt PATH and PYTHONPATH.

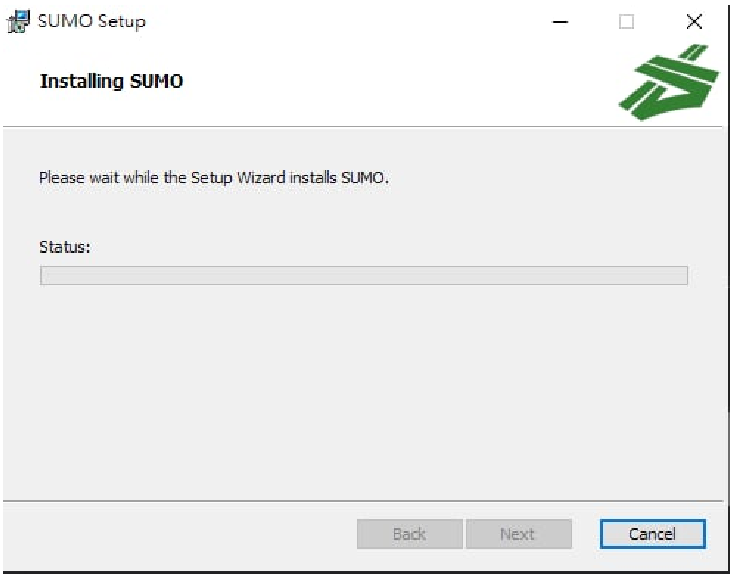
Figure 4.11: SUMO installer destination folder



Ready to install SUMO

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Figure 4.12: SUMO ready to install

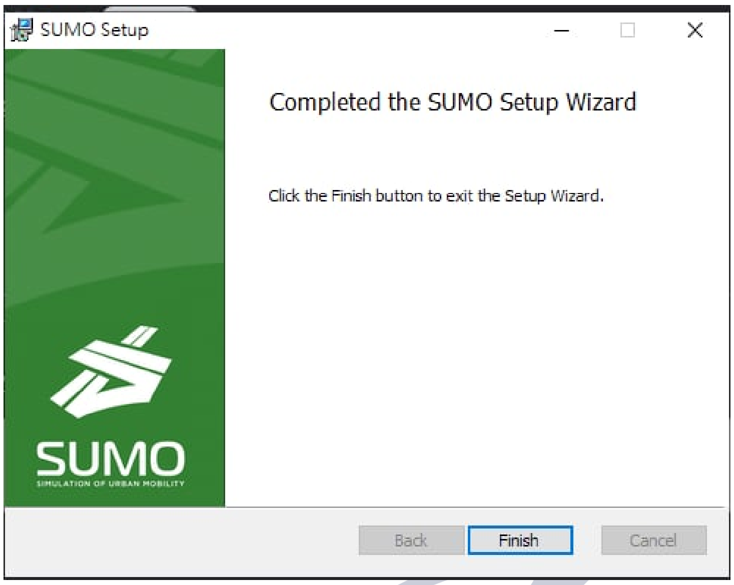


Installing SUMO

Please wait while the Setup Wizard installs SUMO.

Status:

Figure 4.13: SUMO installer progress


SUMO Setup

Completed the SUMO Setup Wizard

Click the Finish button to exit the Setup Wizard.

Figure 4.14: SUMO finished installation

4.2 Model Implementation Details

4.2.1 Data Reduction & Balancing

To prevent the model from becoming biased toward majority classes during training, a data balancing strategy was implemented. The original dataset exhibited significant class imbalance among the various DoS attack types after filtering for `type = 3`. Specifically, the number of samples per attack class was as follows: `DoS` (527,852), `DoSDisruptive` (524,044), `DoSRandom` (522,672), `DoSDisruptiveSybil` (480,073), and `DoSRandomSybil` (480,073).

We capped the number of samples per class at a maximum of 480,000. Classes exceeding this threshold underwent random undersampling to reduce their sizes, while classes with fewer than 480,000 samples retained all available entries. This approach results in a uniformly distributed dataset across all six classes, including the `Normal` class, which is derived from entries labeled with `attack = 0`.

By enforcing this balancing step, we ensure equal representation of each class during training, which is critical for achieving robust and unbiased multi-class classification performance.

4.2.2 Data Split

To evaluate the TransBoost-DoS system, the dataset was first divided into two primary subsets: 80% for training and validation and 20% for final testing. This separation ensures that the final test set remains completely unseen during model development, providing an unbiased assessment of generalization performance. The training and validation portion was later used for cross-validation, which is described in detail in the Model Evaluation subsection. The overall data partitioning strategy is illustrated in Figure 4.15.

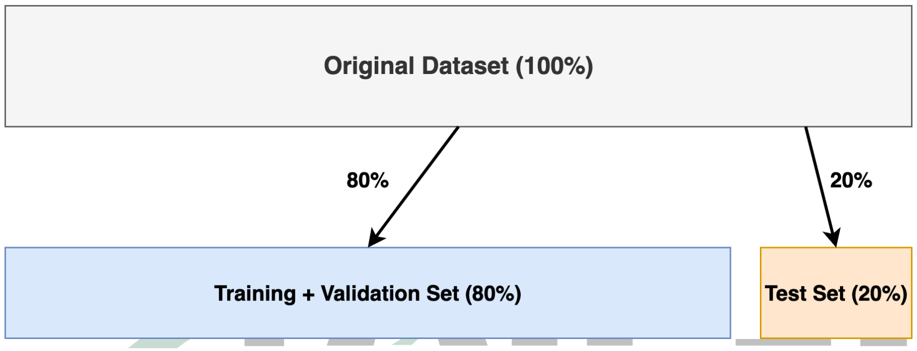


Figure 4.15: Data split process

4.2.3 Model Training

The proposed TransBoost-DoS detection system adopts a hybrid architecture that integrates a Transformer-based temporal feature extractor with an XGBoost classifier for multi-class DoS attack detection in VANETs. This architecture is designed to leverage the Transformer's capability in modeling sequential vehicular behavior while utilizing XGBoost's gradient-boosted decision trees for efficient and interpretable classification.

The selected feature matrix, composed of 17 preprocessed features, is first standardized and transformed into fixed-length time-series sequences using a sliding window of size 30 and stride 3.

We initialize the VANETTransformerFeatureExtractor model with the following configuration: input dimension of 17, hidden dimension of 256, four encoder layers, four self-attention heads, and a dropout rate of 0.6. During training, the Transformer operates in full mode to extract temporal representations; during final feature extraction, it is set to evaluation mode. The model outputs 256-dimensional latent feature vectors by applying linear embedding, learnable positional encoding, multi-head attention, and mean pooling across time steps.

After training the Transformer and softmax classification head on each fold using cross-entropy loss and Adam optimizer, the Transformer is reused to extract high-level features for the entire dataset. These features are then input into an XGBClassifier with 100 boosting rounds, a learning rate of 0.1, and a multi:softmax objective for six-class classification, including Normal and five DoS subtypes.

This two-stage pipeline enhances detection accuracy and generalization by combining deep temporal modeling with ensemble-based decision boundaries, making it well-suited for real-time deployment in vehicular communication environments.

4.2.4 Model Evaluation

During the evaluation stage of the TransBoost-DoS system, a 5-fold cross-validation strategy is employed to enhance robustness and ensure generalizability across varied data partitions. The balanced dataset, comprising normal samples and five distinct DoS attack types, is randomly divided into five equal parts using KFold from scikit-learn, with shuffling enabled and a fixed random seed to maintain reproducibility.

In each iteration, one subset serves as the test set while the remaining four are used for training. Within every fold, the Transformer-based feature extractor and the accompanying softmax classification head are trained jointly using cross-entropy loss, the Adam optimizer, and dynamic learning rate adjustment. Early stopping is applied to avoid overfitting when no improvement in validation loss is observed over consecutive epochs.

Once training is complete, the extracted feature representations are passed to the XGBoost classifier for final prediction. Model outputs are then compared to ground truth labels to compute performance metrics such as recall, accuracy, precision, and F1-score for each fold. Additionally, class-wise accuracy scores are averaged to assess detection consistency across all categories. The best-performing model based on peak validation accuracy is preserved for later deployment or further analysis.

This cross-validation framework enables comprehensive evaluation under diverse conditions. Figure 4.16 illustrates the 5-fold cross-validation process, where each fold is trained and evaluated independently to ensure that the model's performance is not overly dependent on any single data partition.

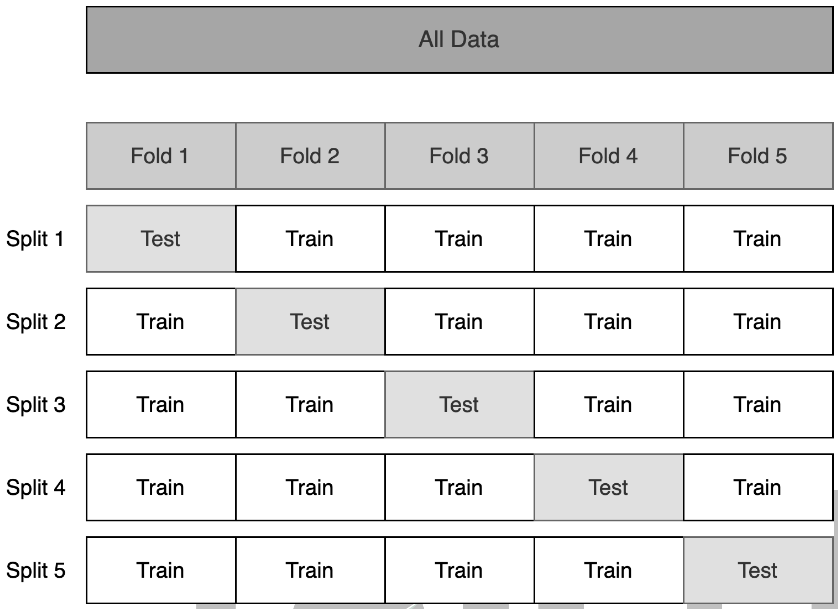


Figure 4.16: 5-fold cross-validation

4.3 Attack Detection Scenario

To evaluate the outcome of the TransBoost-DoS system, we construct two distinct simulation scenarios using SUMO, as illustrated in Figure 4.17.

- Scenario 1 : All vehicles broadcast standard BSMs reflecting legitimate driving behavior. The vehicle-mounted detection model receives and processes these messages, correctly identifying them as normal and forwarding them to downstream vehicular applications.

- Scenario 2 : One vehicle is designated as the attacker and transmits BSMs that mimic different types of DoS attacks (e.g., disruptive, random, or Sybil-based). The detection model on the receiving vehicle analyzes the message patterns, identifies them as attacks, and classifies the attack type. Malicious messages are discarded, and relevant information is logged for further analysis.

Figure 4.18 shows a flowchart of the attack detection process.

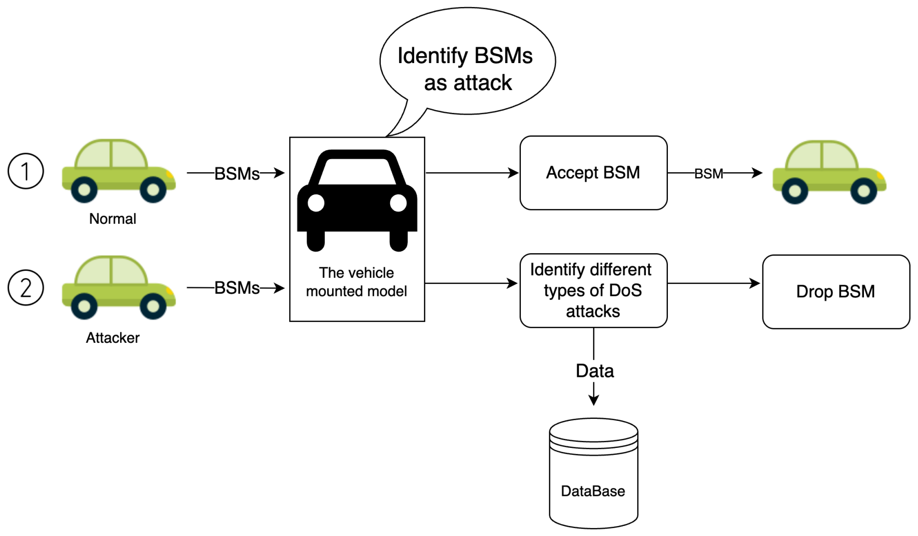


Figure 4.17: DoS attack detection scenario

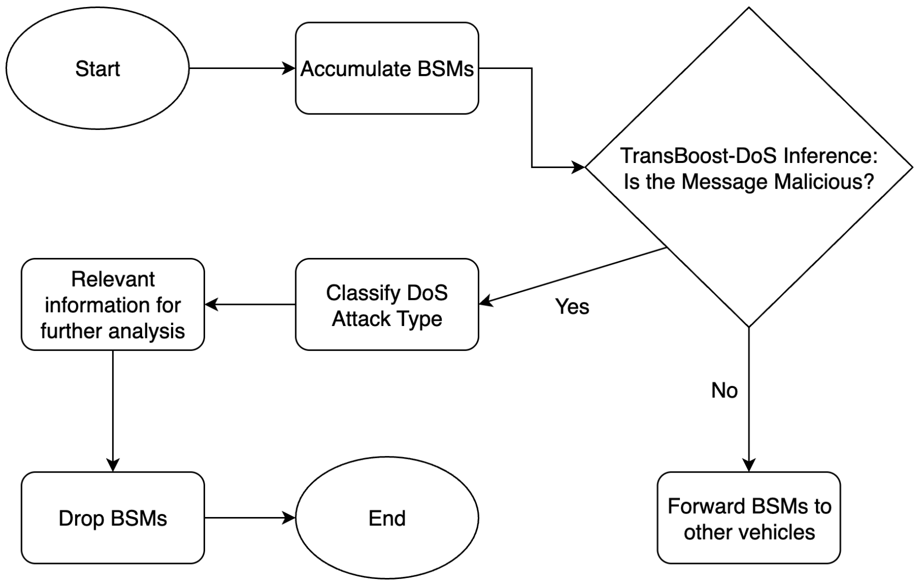

Start
Accumulate BSMs
TransBoost-DoS Inference: Is the Message Malicious?
Yes
Classify DoS Attack Type
Relevant information for further analysis
No
Drop BSMs
End
Forward BSMs to other vehicles

Figure 4.18: DoS attack detection flowchart

4.4 Simulation Implementation Details

4.4.1 Simulation Configuration

To faithfully emulate the dynamics of vehicular communication and assess the performance of the TransBoost-DoS system, we configured a SUMO-based simulation environment with carefully selected parameters. Table 4.4 summarizes the main simulation settings.

Table 4.4: Simulation parameters

Setting	**Description**
Simulation Duration	120 seconds
Road Network Topology	6 parallel unidirectional roads Length: 150 meters per lane
Speed Limit	13.89 m/s (equivalent to 50 km/h)
Traffic Flow Setup	1200 vehicles per hour per lane
Output Format	CSV format for exporting vehicle behavior logs
Receiver Vehicle	Simulated using a Python-based logic for message reception and inference

These parameters were chosen to balance realism, scalability, and computational efficiency. A 120-second simulation window provides adequate runtime to observe the behavioral impact of DoS attacks while maintaining manageable output data size. The six-lane parallel road structure minimizes intersection effects, ensuring that vehicle behavior and message patterns are not confounded by external traffic dynamics. The speed limit of 13.89 m/s reflects typical urban mobility scenarios, and a vehicle generation rate of 1200 vehicles per hour per lane ensures sufficient message density for evaluating the detection system. Output in CSV format simplifies downstream preprocessing and integration with the TransBoost-DoS detection pipeline. Finally, the receiver vehicle is emulated via a Python script that interfaces with the SUMO simulation to receive messages and perform inference using the trained model. The following subsections detail the construction of the road network, route definition, and SUMO GUI visualization process.

4.4.2 SUMO Network and Route File Creation

First we need to create a SUMO network and a route file. In this case, we construct a straight network composed of six parallel roads running from left to right. The network file, saved in XML format, follows the standard structure defined by SUMO. It specifies details such as the number of lanes per road, speed limits, and the geometric layout of each road segment. Each of the six lanes represents an individual route traveling in the same direction, simulating parallel traffic flows. Figure 4.19 illustrates the creation of this six-lane left-to-right road network using SUMO Netedit.

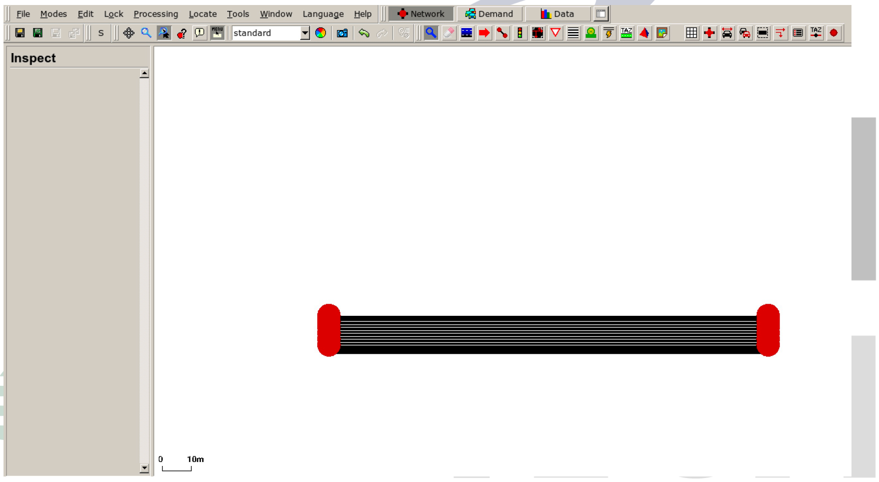


Figure 4.19: SUMO netedit for creating network file

After creating the network file, we need to create a route file that defines the traffic flow in the network. In this case, we build a vehicle that contains a normal vehicle and five different DoS attack vehicles. The route file is saved in the XML format, which is the standard format for SUMO route files. The route file contains information about the vehicles, including their types, speeds, and routes. Figure 4.20 illustrates the creation of the route file using SUMO Netedit.

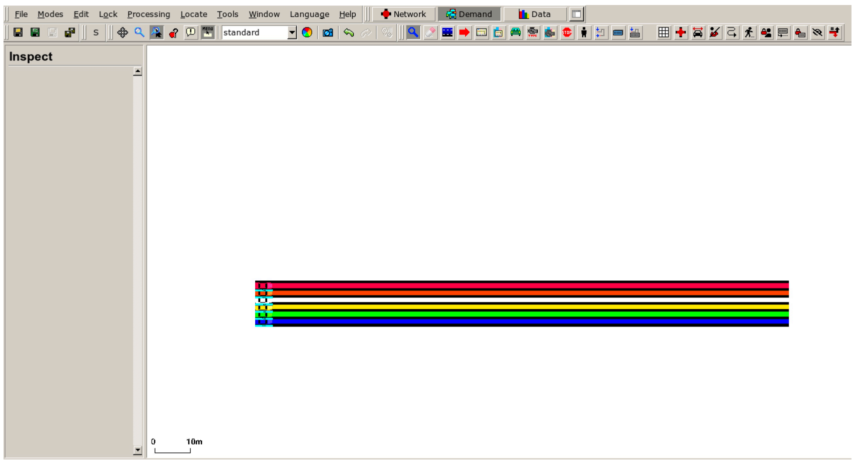

File Modes Edit Lock Processing Locate Tools Window Language Help
Network
Demand
Data
standard
Inspect
0
10m

Figure 4.20: SUMO netedit for creating route file

To visualize the simulation, we can use SUMO's GUI tool, which provides a graphical interface to observe the traffic flow and vehicle behavior in the network. Figure 4.21 illustrates the SUMO GUI tool for visualizing the simulation.

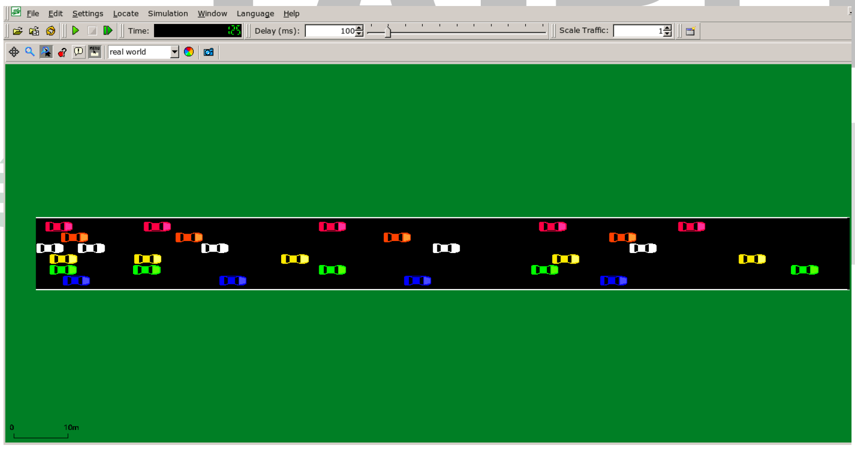


Figure 4.21: SUMO GUI for visualizing the simulation

Chapter 5 Result & Analysis

5.1 Evaluation Metrics

To quantitatively assess the performance of the proposed **TransBoost-DoS** system, we adopt four widely used classification metrics: accuracy, precision, recall, F1-score and confusion matrix [48]. These metrics provide a comprehensive evaluation of the model's ability to correctly identify each class, especially in a multi-class setting involving both normal and various DoS attack types.

- **Accuracy** calculates the percentage of instances that were classified correctly out of all classification attempts.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

- **Precision** calculates the proportion of true positives among all instances classified as positive.

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

- **Recall** calculates how many true positive cases the model successfully detected out of all actual positive instances.

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

- **F1-score** summarizes a model's accuracy by considering both precision and recall, using their harmonic mean to ensure that neither metric is overlooked.

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5.4}$$

- **Confusion Matrix** offers a comprehensive view of the classification model's predictive performance by quantifying true positives, true negatives, false positives, and false negatives for each category.

$$\begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix} \tag{5.5}$$

These metrics are computed for each of the six classes (Normal and five DoS variants), and macro-averaged over all classes to provide an overall performance summary. Additionally, class-wise accuracy is recorded separately to highlight the system's ability to detect specific attack types. A normalized confusion matrix is also generated to visualize prediction distribution and potential misclassification trends.

From a cybersecurity standpoint, recall is crucial to ensure that actual attacks are not missed, while precision helps reduce false alarms that may disrupt normal communications. These metrics, along with the confusion matrix, provide a balanced view of detection effectiveness and reliability in VANETs.

5.2 Results

5.2.1 Performance Analysis of TransBoost-DoS

To thoroughly assess the detection capability and reliability of the proposed TransBoost-DoS system, we performed a comprehensive evaluation using the balanced VeReMi dataset and applied 5-fold cross-validation. The performance was analyzed based on key metrics: accuracy, precision, recall, and F1-score, all of which provide a holistic view of the system's ability to distinguish normal from malicious traffic in VANETs.

The confusion matrix (see Figure 5.1) reveals the distribution of model predictions across all classes. The results show a clear dominance of correct predictions along the diagonal, indicating that the system can accurately differentiate between Normal, DoS, DoS disruptive, DoS random, DoS disruptive sybil, and DoS random sybil attack types. Misclassifications are minimal, underscoring the robustness of the hybrid learning approach.

Further, the classification report (see Figure 5.2) quantifies the detection performance for each class. TransBoost-DoS achieves exceptionally high scores across all evaluated categories,

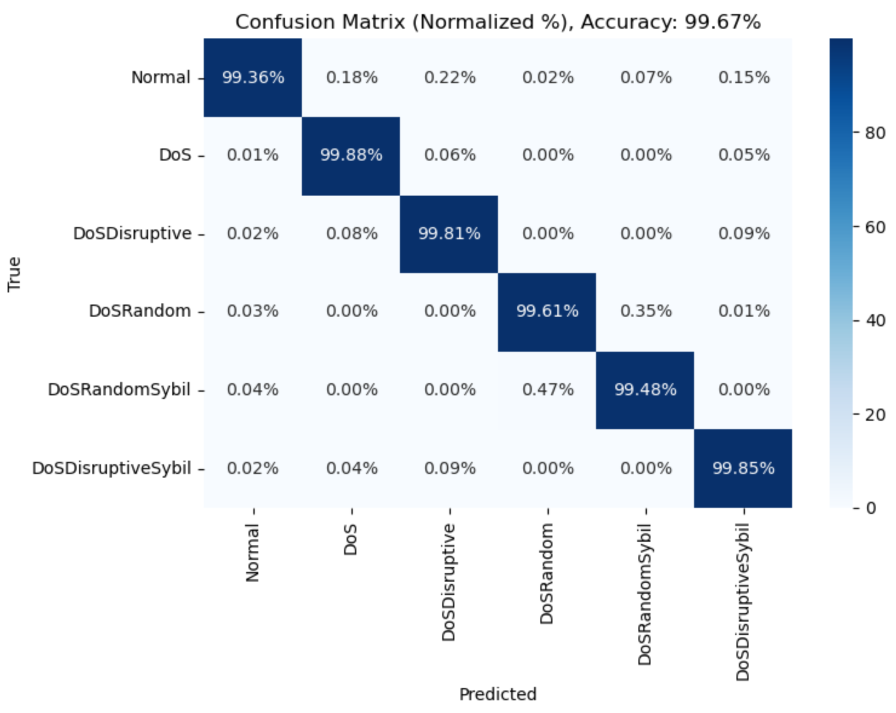
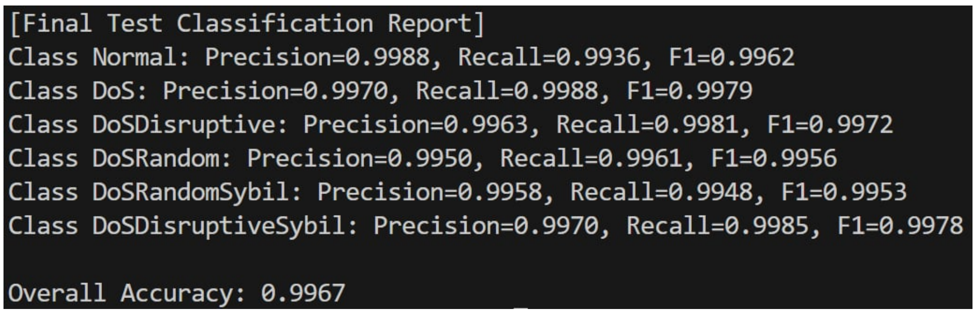


Figure 5.1: Confusion matrix of TransBoost-DoS

with precision, recall, and F1-score values all exceeding 99% for every attack type as well as normal traffic. Specifically, the system attains a macro-averaged accuracy of 99.67%, which demonstrates both the discriminative power and generalizability of the model in diverse VANETs attack scenarios.



```
[Final Test Classification Report]
Class Normal: Precision=0.9988, Recall=0.9936, F1=0.9962
Class DoS: Precision=0.9970, Recall=0.9988, F1=0.9979
Class DoSDisruptive: Precision=0.9963, Recall=0.9981, F1=0.9972
Class DoSRandom: Precision=0.9950, Recall=0.9961, F1=0.9956
Class DoSRandomSybil: Precision=0.9958, Recall=0.9948, F1=0.9953
Class DoSDisruptiveSybil: Precision=0.9970, Recall=0.9985, F1=0.9978

Overall Accuracy: 0.9967
```

Figure 5.2: Classification report of TransBoost-DoS

5.2.2 Comparison with Baseline Models

To reinforce the validation of the proposed TransBoost-DoS system, we carried out a comparative study with two baseline models:

- **Transformer-only Model**: This model utilizes only the Transformer-based feature extractor without the XGBoost classifier.

- **XGBoost-only Model**: This model employs the XGBoost classifier directly on the raw input features without any temporal feature extraction.

The effectiveness of these baseline models was evaluated using the same balanced VeReMi dataset and 5-fold cross-validation, allowing for a fair comparison with TransBoost-DoS. The results, summarized in Table 5.1, indicate that the TransBoost-DoS system significantly outperforms both baseline models across all metrics.

Table 5.1: Performance comparison of baseline models

Models	**Metrics**	**Classes**					
		Normal	**DoS**	**DoSDisruptive**	**DoSRandom**	**DoSRandomSybil**	**DoSDisruptiveSybil**
XGBoost Only	Precision	93.32%	40.35%	39.44%	69.62%	69.84%	44.62%
	Recall	83.75%	47.58%	34.99%	71.88%	63.81%	48.66%
	F1 Score	88.28%	43.67%	37.08%	70.74%	66.69%	46.55%
Transformer Only	Precision	99.10%	82.10%	81.69%	69.61%	75.97%	91.56%
	Recall	90.18%	90.60%	86.71%	85.32%	62.73%	79.99%
	F1 Score	94.43%	86.14%	84.12%	76.67%	68.72%	85.39%
Transformer + XGBoost	**Precision**	**99.88%**	**99.70%**	**99.63%**	**99.50%**	**99.58%**	**99.70%**
	Recall	**99.36%**	**99.88%**	**99.81%**	**99.61%**	**99.48%**	**99.85%**
	F1 Score	**99.62%**	**99.79%**	**99.72%**	**99.56%**	**99.53%**	**99.78%**

Figure 5.3 and 5.4 show the confusion matrices for the XGBoost-only and Transformer-only models. The XGBoost-only model suffers from significant misclassifications across multiple DoS variants, revealing its limited capacity to distinguish complex vehicular behaviors. In contrast, the Transformer-only model exhibits clearer diagonal patterns, suggesting improved temporal learning, though confusion remains among similar attack types such as DoSRandom and DoSRandomSybil.

Corresponding classification reports in Figure 5.5 and 5.6 further support these findings. While Transformer-only outperforms XGBoost-only, still falls short compared to the hybrid TransBoost-DoS system.

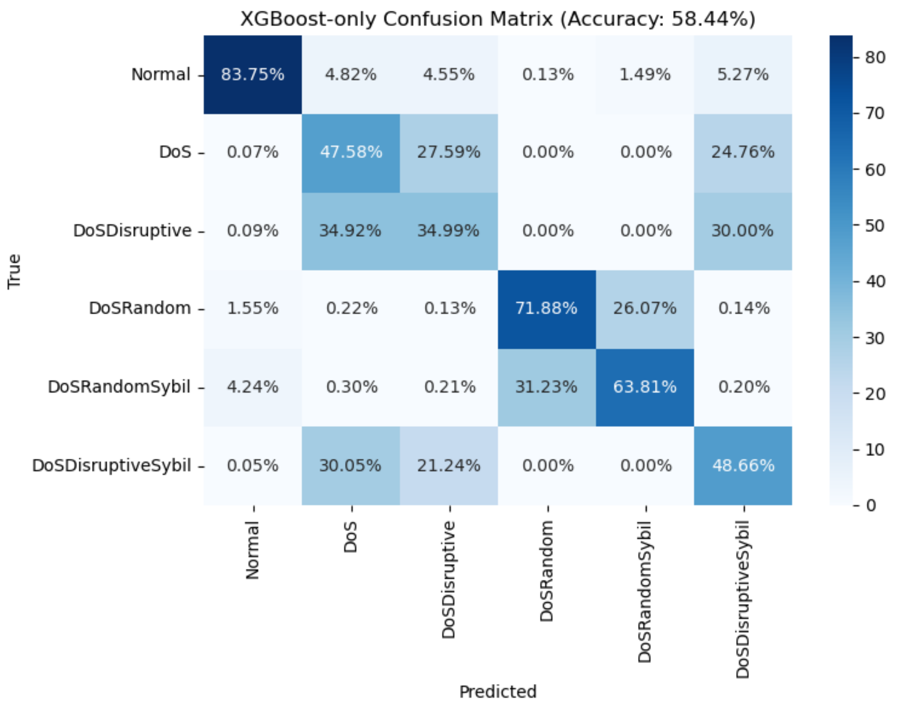


Figure 5.3: Confusion matrix of XGBoost-only model

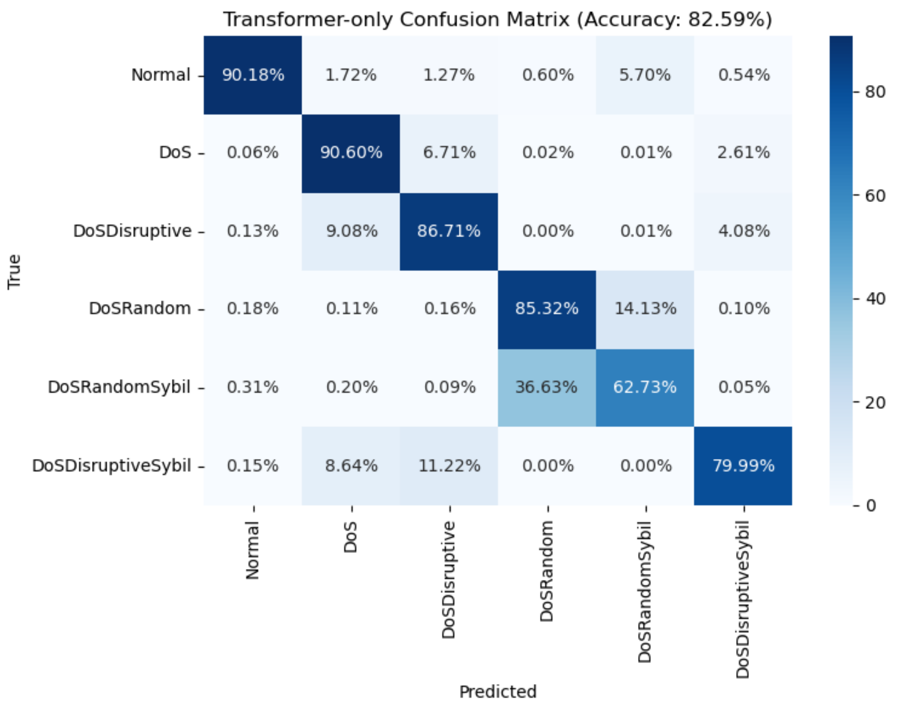
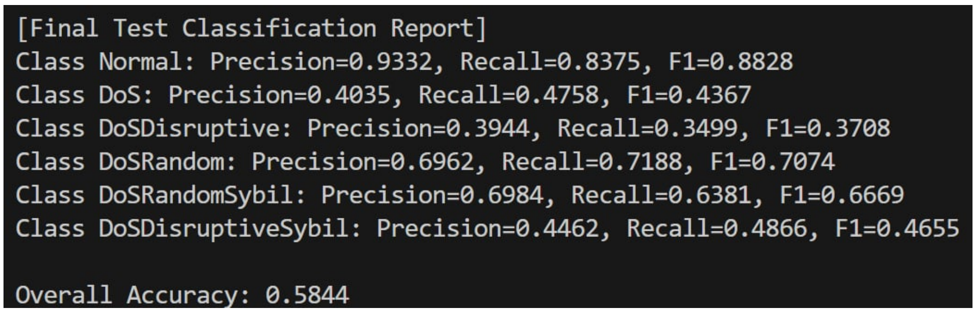
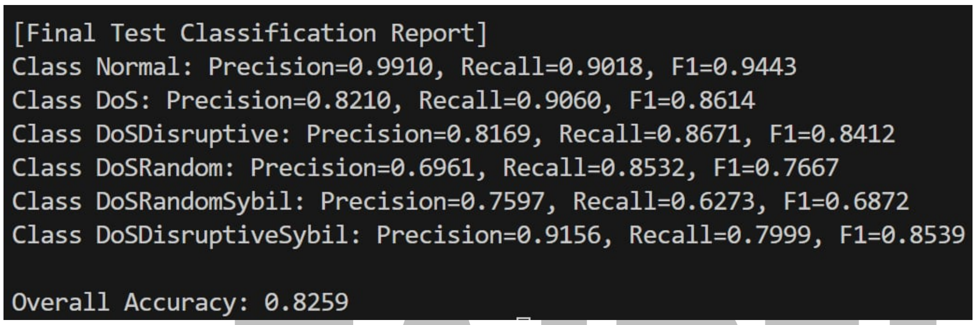


Figure 5.4: Confusion matrix of Transformer-only model



```
[Final Test Classification Report]
Class Normal: Precision=0.9332, Recall=0.8375, F1=0.8828
Class DoS: Precision=0.4035, Recall=0.4758, F1=0.4367
Class DoSDisruptive: Precision=0.3944, Recall=0.3499, F1=0.3708
Class DoSRandom: Precision=0.6962, Recall=0.7188, F1=0.7074
Class DoSRandomSybil: Precision=0.6984, Recall=0.6381, F1=0.6669
Class DoSDisruptiveSybil: Precision=0.4462, Recall=0.4866, F1=0.4655

Overall Accuracy: 0.5844
```

Figure 5.5: Classification report of XGBoost-only model



```
[Final Test Classification Report]
Class Normal: Precision=0.9910, Recall=0.9018, F1=0.9443
Class DoS: Precision=0.8210, Recall=0.9060, F1=0.8614
Class DoSDisruptive: Precision=0.8169, Recall=0.8671, F1=0.8412
Class DoSRandom: Precision=0.6961, Recall=0.8532, F1=0.7667
Class DoSRandomSybil: Precision=0.7597, Recall=0.6273, F1=0.6872
Class DoSDisruptiveSybil: Precision=0.9156, Recall=0.7999, F1=0.8539

Overall Accuracy: 0.8259
```

Figure 5.6: Classification report of Transformer-only model

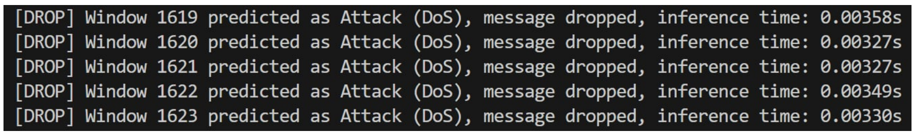
5.2.3 Discussion

The results clearly demonstrate that the TransBoost-DoS system outperforms both baseline models, achieving a macro-averaged accuracy of 99.67% and consistently high precision, recall, and F1-score across all classes. The Transformer-based feature extraction effectively captures the temporal dependencies and complex patterns in vehicular message sequences, while the XG-Boost classifier leverages these rich features to provide fast and interpretable attack detection. This hybrid approach minimizes false positives and maintains high true positive rates, even in the presence of complex or ambiguous attack behaviors. The comparative analysis highlights the limitations of using either Transformer or XGBoost alone for DoS attack detection in VANETs. The Transformer-only model, while better than the XGBoost-only model, still struggles with certain attack types, particularly those with subtle behavioral variations. The XGBoost-only model fails to capture the temporal dynamics of vehicular messages, leading to poor performance across all metrics. In contrast, the TransBoost-DoS system integrates the strengths of both models, achieving highly accurate results in DoS attack detection for vehicular networks. This positions the system as a strong candidate for real-world deployment in intelligent transportation environments, where rapid and precise attack detection is critical for ensuring traffic safety and cybersecurity. The results confirm that TransBoost-DoS offers a reliable and effective solution for detecting DoS attacks in VANETs, successfully integrating high accuracy, resilience, and computational efficiency. Its capability to maintain high detection rates while minimizing false positives highlights its potential for deployment in real-time intelligent transportation systems.

5.3 Simulation Results

The simulation results were obtained using the SUMO framework, which allowed us to create realistic vehicular traffic scenarios and evaluate the performance of the TransBoost-DoS system under dynamic conditions. To evaluate the proposed detection and message forwarding system, we conducted comprehensive simulations involving both normal and attack scenarios in a SUMO-based vehicular network environment. The simulation results are summarized below, with visual evidence shown in Figures 5.7 to 5.10.

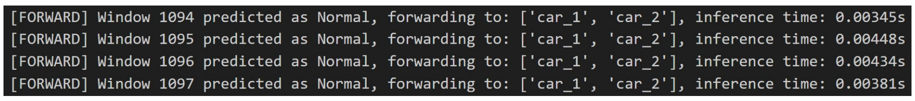
During the simulation, the system dynamically predicted the type of each message window using a hybrid deep learning and XGBoost classifier. As illustrated in the simulation logs (see Figure 5.7), when the model detected an attack type, the corresponding message was immediately dropped demonstrated by the output `[DROP] window N predicted as (attack_type), message dropped` thereby preventing the malicious information from being forwarded within the network. This robust detection and blocking mechanism is essential for mitigating the impact of denial-of-service and Sybil attacks in vehicular environments.



```
[DROP] Window 1619 predicted as Attack (DoS), message dropped, inference time: 0.00358s
[DROP] Window 1620 predicted as Attack (DoS), message dropped, inference time: 0.00327s
[DROP] Window 1621 predicted as Attack (DoS), message dropped, inference time: 0.00327s
[DROP] Window 1622 predicted as Attack (DoS), message dropped, inference time: 0.00349s
[DROP] Window 1623 predicted as Attack (DoS), message dropped, inference time: 0.00330s
```

Figure 5.7: Simulation log of prediction and message dropping

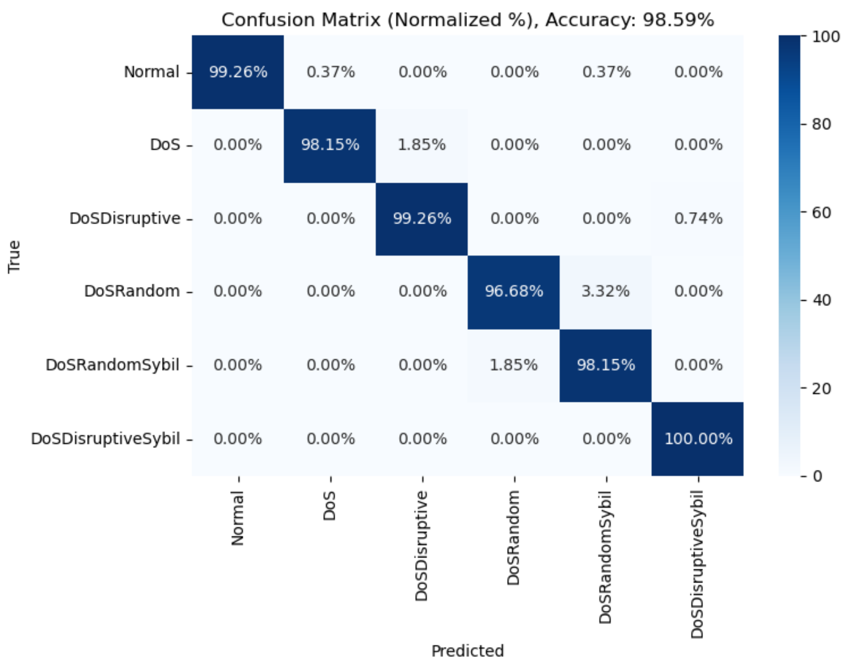
Conversely, when the prediction result was ”Normal” (see Figure 5.8), the message was successfully forwarded to other vehicles, as indicated by the `[FORWARD] window N predicted as Normal, forwarding to: [...]`. This behavior demonstrates that the proposed system is able to distinguish legitimate traffic and allows safe message propagation, thereby maintaining efficient V2V communication under normal circumstances.



```
[FORWARD] Window 1094 predicted as Normal, forwarding to: ['car_1', 'car_2'], inference time: 0.00345s
[FORWARD] Window 1095 predicted as Normal, forwarding to: ['car_1', 'car_2'], inference time: 0.00448s
[FORWARD] Window 1096 predicted as Normal, forwarding to: ['car_1', 'car_2'], inference time: 0.00434s
[FORWARD] Window 1097 predicted as Normal, forwarding to: ['car_1', 'car_2'], inference time: 0.00381s
```

Figure 5.8: Simulation log of prediction and message forwarding

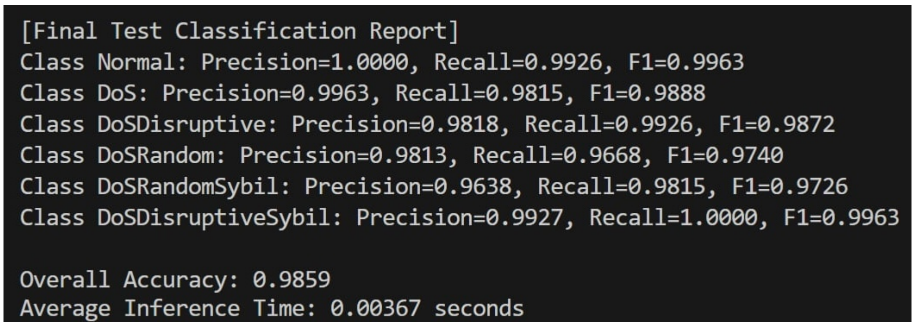
Overall, The confusion matrix generated from the SUMO simulation (see Figure 5.9) reveals that most predictions are correctly aligned along the diagonal, indicating that the system can accurately distinguish normal traffic from various DoS attack types even in dynamic, real-world-like scenarios. Misclassifications are minimal and primarily limited to a few similar attack variants, demonstrating the robustness and practical discrimination ability of the TransBoost-DoS model during real-time Simulation.



	Normal	DoS	DoSDisruptive	DoSRandom	DoSRandomSybil	DoSDisruptiveSybil
Normal	99.26%	0.37%	0.00%	0.00%	0.37%	0.00%
DoS	0.00%	98.15%	1.85%	0.00%	0.00%	0.00%
DoSDisruptive	0.00%	0.00%	99.26%	0.00%	0.00%	0.74%
DoSRandom	0.00%	0.00%	0.00%	96.68%	3.32%	0.00%
DoSRandomSybil	0.00%	0.00%	0.00%	1.85%	98.15%	0.00%
DoSDisruptiveSybil	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%

Figure 5.9: Simulation confusion matrix

The quantitative evaluation report (see Figure 5.10) further supports these findings, showing that the system achieves a high overall accuracy of 98.59% and an average inference time per window of just 3.67 ms. In addition, all class-wise precision, recall, and F1-scores exceed 96%, covering both Normal traffic and multiple DoS variants. Notably, both the detection accuracy and response latency meet real-time requirements for vehicular networks.



```
[Final Test Classification Report]
Class Normal: Precision=1.0000, Recall=0.9926, F1=0.9963
Class DoS: Precision=0.9963, Recall=0.9815, F1=0.9888
Class DoSDisruptive: Precision=0.9818, Recall=0.9926, F1=0.9872
Class DoSRandom: Precision=0.9813, Recall=0.9668, F1=0.9740
Class DoSRandomSybil: Precision=0.9638, Recall=0.9815, F1=0.9726
Class DoSDisruptiveSybil: Precision=0.9927, Recall=1.0000, F1=0.9963

Overall Accuracy: 0.9859
Average Inference Time: 0.00367 seconds
```

Figure 5.10: Simulation classification report

These results demonstrate that the proposed TransBoost detection system not only achieves excellent classification performance, but also enables rapid response and practical attack mitigation, as verified by both per-class metrics and real-time SUMO simulation behaviors.

5.4 Comparison with existing works

To further evaluate the effectiveness of TransBoost-DoS, we compare it with four representative models reported in [29], including both deep learning and hybrid machine learning-based intrusion detection approaches for DoS attacks in VANETs. These models were evaluated on the VeReMi dataset, under comparable multi-class settings.

Table 5.2: Comparison with methods from [29]

Model	**Accuracy**	**Precision**	**Recall**	**F1-score**
CNN + RF	87.94%	87.79%	87.84%	87.85%
RNN + RF	81.96%	81.02%	80.80%	80.90%
CNN + DT	86.86%	86.93%	86.86%	86.83%
RNN + DT	81.56%	81.60%	81.54%	81.56%
TransBoost-DoS (Ours)	**99.67%**	**99.67%**	**99.67%**	**99.67%**

As shown in Table 5.2, TransBoost-DoS significantly outperforms all four baseline models from [29] in terms of accuracy, precision, recall, and F1-score. Notably, while CNN and RNN-based hybrids achieved acceptable performance, their detection capability was still limited by either shallow temporal modeling (in RF/DT classifiers) or insufficient feature richness.

In contrast, TransBoost-DoS leverages the Transformer's strength in modeling long-range dependencies within vehicular message sequences and combines it with XGBoost's robust and interpretable classification power. This hybrid design yields high detection precision with minimal false alarms, making it more suitable for real-time deployment in security-critical VANETs scenarios.

Chapter 6 Conclusion & Future Work

6.1 Conclusion

In this thesis, we proposed TransBoost-DoS, a hybrid detection system designed to accurately identify DoS attacks in VANETs. By integrating a Transformer-based temporal feature extractor with an XGBoost classifier, the system effectively leverages the sequential modeling capability of deep learning and the robustness of gradient boosting for tabular classification. The model was trained and evaluated on a balanced version of the VeReMi dataset, achieving a macro-averaged accuracy of 99.67% across six classes, including five distinct DoS variants and Normal traffic.

To further validate the system under dynamic, real-world-like conditions, we implemented simulation scenarios using the SUMO framework. The simulations successfully demonstrated the system's capacity to detect attacks with high precision and minimal false positives in both controlled and adversarial traffic environments. Moreover, comparative analysis showed that TransBoost-DoS outperforms both Transformer-only and XGBoost-only baselines, confirming the advantages of a hybrid approach.

Overall, TransBoost-DoS presents a robust, interpretable, and efficient solution for DoS attack detection in intelligent vehicular networks, and represents a step forward in the integration of deep and classical machine learning models in the cybersecurity domain.

6.2 Future Work

Although the proposed TransBoost-DoS system has demonstrated promising performance in detecting various DoS attack types, several future directions can be explored to further enhance its effectiveness, scalability, and real-world applicability.

First, we plan to extend the detection coverage to incorporate additional attack types beyond DoS, such as replay attacks, falsified message injection, and location spoofing. These attack scenarios pose significant threats in VANETs environments and require more diverse feature representations and behavioral analysis for reliable detection.

Second, we aim to explore multi-modal fusion techniques by integrating vision and LiDAR data with vehicular communication features. This approach will enable the detection model to leverage spatial, visual, and behavioral cues simultaneously, improving robustness against deceptive or ambiguous attack patterns in complex traffic environments.

Finally, we intend to deploy the system on edge devices, particularly OBUs, to enable real-time inference with minimal latency. By optimizing the model's computational efficiency and memory footprint, the hybrid architecture can be adapted for execution in resource-constrained vehicular platforms, paving the way for practical, on-the-fly intrusion detection in intelligent transportation systems.

References

[1] C. Jiacheng, Z. Haibo, Z. Ning, Y. Peng, G. Lin, and S. X. Sherman, “Software defined internet of vehicles: Architecture, challenges and solutions,” *Journal of communications and information networks*, vol. 1, no. 1, pp. 14–26, 2016.

[2] C. K. Toh, *Ad hoc mobile wireless networks: protocols and systems*. Pearson Education, 2001.

[3] A. Demba and D. P. Möller, “Vehicle-to-vehicle communication technology,” in *2018 IEEE international conference on electro/information technology (EIT)*, IEEE, 2018, pp. 0459–0464.

[4] Wikipedia contributors, *Over-the-air update — Wikipedia, the free encyclopedia*, [Online; accessed 15-July-2025], 2025. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Over-the-air_update&oldid=1298743785`.

[5] Wikipedia contributors, *Denial-of-service attack — Wikipedia, the free encyclopedia*, [Online; accessed 15-July-2025], 2025. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Denial-of-service_attack&oldid=1299471217`.

[6] MarketsandMarkets™, *Software defined vehicle market*, `https://www.globenewswire.com/news-release/2024/07/30/2921107/0/en/Software-Defined-Vehicle-Market-worth-1-237-6-billion-by-2030-Globally-at-a-CAGR-of-34-0-says-MarketsandMarkets.html`, [Online; accessed 24-February-2025], 2025.

[7] M. Arif, G. Wang, M. Z. A. Bhuiyan, T. Wang, and J. Chen, “A survey on security attacks in vanets: Communication, applications and challenges,” *Vehicular Communications*, vol. 19, p. 100 179, 2019.

[8] A. Y. Ng *et al.*, “Preventing” overfitting” of cross-validation data,” in *ICML*, vol. 97, 1997, pp. 245–253.

[9] P. A. Lopez, M. Behrisch, L. Bieker-Walz, *et al.*, “Microscopic traffic simulation using sumo,” in *2018 21st international conference on intelligent transportation systems (ITSC)*, Ieee, 2018, pp. 2575–2582.

[10] Z. Afzal and M. Kumar, “Security of vehicular ad-hoc networks (vanet): A survey,” in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1427, 2020, p. 012 015.

[11] J. Lastinec and M. Keszeli, “Analysis of realistic attack scenarios in vehicle ad-hoc networks,” in *2019 7th International Symposium on Digital Forensics and Security (ISDFS)*, IEEE, 2019, pp. 1–6.

[12] Wikipedia contributors, *Vehicle-to-everything — Wikipedia, the free encyclopedia*, [Online; accessed 15-July-2025], 2025. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Vehicle-to-everything&oldid=1298502144`.

[13] J. B. Kenney, "Dedicated short-range communications (dsrc) standards in the united states," *Proceedings of the IEEE*, vol. 99, no. 7, pp. 1162–1182, 2011.

[14] Y. Yao, L. Rao, X. Liu, and X. Zhou, "Delay analysis and study of ieee 802.11 p based dsrc safety communication in a highway environment," in *2013 Proceedings IEEE INFOCOM*, IEEE, 2013, pp. 1591–1599.

[15] Wikipedia contributors, *3gpp — Wikipedia, the free encyclopedia*, [Online; accessed 15-July-2025], 2025. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=3GPP&oldid=1299275163`.

[16] R. Weber, J. Misener, and V. Park, "C-v2x-a communication technology for cooperative, connected and automated mobility," in *Mobile Communication-Technologies and Applications; 24. ITG-Symposium*, VDE, 2019, pp. 1–6.

[17] M. A. Al-shareeda, M. A. Alazzawi, M. Anbar, S. Manickam, and A. K. Al-Ani, "A comprehensive survey on vehicular ad hoc networks (vanets)," in *2021 International Conference on Advanced Computer Applications (ACA)*, IEEE, 2021, pp. 156–160.

[18] R. Sedar, C. Kalalas, F. Vázquez-Gallego, L. Alonso, and J. Alonso-Zarate, "A comprehensive survey of v2x cybersecurity mechanisms and future research paths," *IEEE Open Journal of the Communications Society*, vol. 4, pp. 325–391, 2023.

[19] S. Masood, Y. Saeed, A. Ali, *et al.*, "Detecting and preventing false nodes and messages in vehicular ad-hoc networking (vanet)," *IEEE Access*, vol. 11, pp. 93 920–93 934, 2023.

[20] A. Varga, "Omnet++," in *Modeling and tools for network simulation*, Springer, 2010, pp. 35–59.

[21] S. Gyawali and Y. Qian, "Misbehavior detection using machine learning in vehicular communication networks," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, IEEE, 2019, pp. 1–6.

[22] H. Sedjelmaci, S. M. Senouci, and M. A. Abu-Rgheff, "An efficient and lightweight intrusion detection mechanism for service-oriented vehicular networks," *IEEE Internet of things journal*, vol. 1, no. 6, pp. 570–577, 2014.

[23] J. Kamel, M. Wolf, R. W. Van Der Hei, A. Kaiser, P. Urien, and F. Kargl, "Veremi extension: A dataset for comparable evaluation of misbehavior detection in vanets," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, IEEE, 2020, pp. 1–6.

[24] R. W. Van Der Heijden, T. Lukaseder, and F. Kargl, "Veremi: A dataset for comparable evaluation of misbehavior detection in vanets," in *International conference on security and privacy in communication systems*, Springer, 2018, pp. 318–337.

[25] E. A. Feukeu and S. Mbuyu, “Machine learning for link adaptation problem formulation in vanets,” in *2023 17th International Conference on Telecommunication Systems, Services, and Applications (TSSA)*, IEEE, 2023, pp. 1–5.

[26] P. Abi Singh, V. Kamboj, and R. Kaur, “Role of vanets in machine learning and deep learning,” in *2023 6th International Conference on Contemporary Computing and Informatics (IC3I)*, IEEE, vol. 6, 2023, pp. 306–312.

[27] K Suganyadevi, V Swathi, T Santhiya, S. S. Sankari, and V. Swathi, “Machine learning algorithm based vanet traffic management system,” in *2023 7th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, IEEE, 2023, pp. 747–752.

[28] N. Kadam and R. S. Krovi, “Machine learning approach of hybrid ksvn algorithm to detect ddos attack in vanet,” *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 7, 2021.

[29] A. Hameed and I. Saini, “Lightweight hybrid approach for dos attack detection in vanet,” in *2024 IEEE Middle East Conference on Communications and Networking (MECOM)*, IEEE, 2024, pp. 345–350.

[30] A. Kumar, M. A. Shahid, A. Jaekel, N. Zhang, and M. Kneppers, “Machine learning based detection of replay attacks in vanet,” in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2023, pp. 1–6.

[31] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[32] F. Ullah, S. Ullah, G. Srivastava, and J. C.-W. Lin, “Ids-int: Intrusion detection system using transformer-based transfer learning for imbalanced network traffic,” *Digital Communications and Networks*, vol. 10, no. 1, pp. 190–204, 2024.

[33] A. A. Shuvro, M. S. Khan, M. Rahman, F. Hussain, M. Moniruzzaman, and M. S. Hossen, “Transformer based traffic flow forecasting in sdn-vanet,” *IEEE Access*, vol. 11, pp. 41 816–41 826, 2023.

[34] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[35] S. Amaouche, AzidineGuezzaz, S. Benkirane, and MouradeAzrour, “Ids-xgbfs: A smart intrusion detection system using xgboostwith recent feature selection for vanet safety,” *Cluster Computing*, vol. 27, no. 3, pp. 3521–3535, 2024.

[36] J. Li, Y. Song, M. Zheng, S. Zhang, and H. Liang, “Fexgbids: Federated xgboost based intrusion detection system for in-vehicle network,” *IEEE Access*, 2025.

[37] haider094, *Veremi_dataset*, `https://www.kaggle.com/datasets/haider094/veremi-dataset`, 2023.

[38] E. Bott and C. Stinson, *Windows 10 inside out*. Microsoft Press, 2019.

[39] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[40] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[41] M. L. Waskom, "Seaborn: Statistical data visualization," *Journal of open source software*, vol. 6, no. 60, p. 3021, 2021.

[42] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 03, pp. 90–95, 2007.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[44] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, Austin, TX, vol. 445, 2010, pp. 51–56.

[45] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: `10.1038/s41586-020-2649-2`. [Online]. Available: `https://doi.org/10.1038/s41586-020-2649-2`.

[46] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, "Foundations of json schema," in *Proceedings of the 25th International Conference on World Wide Web*, International World Wide Web Conferences Steering Committee, 2016, pp. 263–273.

[47] Python Software Foundation, *`Os` —miscellaneous operating system interfaces*, Accessed: 2025-05-23, 2024. [Online]. Available: `https://docs.python.org/3/library/os.html`.

[48] O. Rainio, J. Teuho, and R. Klén, "Evaluation metrics and statistical tests for machine learning," *Scientific Reports*, vol. 14, no. 1, p. 6086, 2024.