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

# 檢測分散式阻斷服務攻擊強化基於超參數調整之機器學習模型的能力

# Strengthen the Capability of Machine Learning Models for Detecting DDoS Attack Based on Hyperparameter Tuning

研究生：陳劭睿

Researcher: Shao-Rui Chen

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 陳劭睿 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：吳和庭

馬奕葳

陳香君

指導教授：陳香君

所　　長：創新前瞻科技研究學院院長陳金聖（印） 代

中 華 民 國　一百一十四　年　七　月　九　日

# Abstract

Keyword: DDoS, Machine Learning, Hyperparameter Tuning, Cybersecurity

In recent years, the occurrence & complexity of Distributed Denial of Service (DDoS) attacks have escalated significantly, posing threats to the availability, performance, and security of networked systems. With the rapid progression of Artificial Intelligence (AI) & Machine Learning (ML) technologies, attackers can leverage intelligent tools to automate and amplify DDoS attacks with minimal human intervention. This growing sophistication underscores the urgent need for more accurate and efficient detection mechanisms.

This thesis proposes a method to strengthen the capability of ML models in detecting DDoS attacks based on hyperparameter tuning. By optimizing model parameters, the proposed approach is going to enhance the performance of ML models in identifying DDoS attacks. The CIC-DDoS2019 dataset is utilized in this thesis as it offers a comprehensive set of real-world DDoS attack scenarios across various protocols and servcies.

The proposed methodology incorporates key stages including data preprocessing, data splitting, model training, validation, and testing. Three ML Models are trained and tuned by using an adaptive grid searchCV (Cross_Validation) strategy to identify optimal parameter configurations. The results demonstrate that our method significantly improves the performance & efficacy compared with the general GridSearchCV. The SVM model attains the 99.87% testing accuracy and has less execution time than the general GridSearchCV (about 28% faster). The LR model attains the 99.6830% testing accuracy and 16.90 seconds of execution time (maintaining the same testing accuracy but decreasing the execution time by about 22.8%). And the KNN model attains 99.8395% testing accuracy and 2388.89 seconds of execution time (maintaining the same testing accuracy but decreasing the execution time by about 63%).

These results show our approach improves DDoS detection performance and efficacy, offering innovative thoughts into the practical application of enhancing ML models' performance in real-world scenarios.

# Acknowledgements

At the completion of this thesis, I would like to express my deepest gratitude to all those who have supported and accompanied me throughout this research journey.

First and foremost, I would like to sincerely thank my advisor, Professor Hsiang-Chun Chen, for her patient guidance and professional support in shaping my research direction and experimental design. During times of uncertainty and difficulty, she always provided me with clear direction and constructive feedback, serving as a strong foundation throughout the entire process.

I also wish to extend my heartfelt thanks to the oral defense committee members, Professor Ho-Ting Wu and Professor Yi-Wei Ma. Your insightful questions and valuable suggestions during the defense have greatly broadened my academic perspective and helped refine the quality of this thesis. Your feedback has been instrumental to my academic growth.

A very special thanks goes to my girlfriend, Yun Zhong. Thank you for being by my side during the most stressful and exhausting moments, offering your gentle support and understanding. Your encouragement and companionship gave me strength and confidence to keep going. Having you with me during this journey has been an irreplaceable blessing.

I am also deeply grateful to my parents. Thank you for your unconditional love and unwavering support, which allowed me to dedicate myself fully to my research without worry. Your encouragement and trust have always been the driving force that pushed me forward.

I would also like to thank my lab mates: Hsiang-Yu, Chien-Chang, Tzu-Yu, Te-Shao, Po-Hsun, Ricky, Cheng-Yen, Yu-Chun, Cheng-Cheng, and Wen-Hsuan. Thank you for walking alongside me during this journey—whether it was solving technical challenges together, cheering each other on, or sharing everyday moments. Your presence made the experience far less lonely and helped me grow both academically and personally.

The completion of this thesis embodies the collective wisdom and support of many individuals. I will always cherish this unforgettable journey and hope to transform this gratitude into continued effort and contribution as I move forward in the field I love.

Shao-Rui Chen respectfully written at

Master of Science in Information Security

National Taipei University of Technology

July 18, 2025

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1 Introduction

In recent years, the field of cybersecurity [1] has increasingly concentrated on the detection and mitigation of Distributed Denial of Service (DDoS) [2] attacks, which posed a significant threat to network availability and performance. This research stems from the growing prevalence of DDoS attacks, which can disrupt services and lead to gigantic financial losses to organizations. With the evolution of technology, the landscape of DDoS attacks has undergone significant changes. Historically, launching a DDoS attack often required manual intervention and considerable technical expertise. Attacks would need to orchestrate their efforts by coordinating mutliple compromised systems, which limited the scale and frequency of such attacks. However, with the swift advancements in Artificial Intelligence (AI) [3] and Machine Learning (ML) [4], the DDoS attacks have shifted dramatically. In the present era, AI-driven tools enable attackers to automate the process of launching DDoS attacks, making them more efficient and more challenging to detect. These sophisticated algorithms can analyze vulnerabilities in target systems and deploy large-scale attacks with minimal human involvement. As the occurrence and complexity of DDoS attacks continue to evlove, the necessity for more effective and efficient detection approaches has become more critical. The ability to execute complex attack strategies at scale has made it essential for cybersecurity professionals to develop more robust defenses against these evolving threats.

To realize the vision mentioned above, numerous scholars have made momentous contributions to the understanding and detection of DDoS attacks using ML & methods. Subhashini Peneti et al. [5] offer an approach to detect DDoS attacks using multiple kinds of models, such as Random Forest (RF), AdaBoost, XGBoost, and Multi-Layer Perceptron (MLP) to classify the network requests as normal or abnormal. By combining feature selection & ML techniques, the authors were able to achieve a high-performing Intrusion Detection System (IDS) that could promote the capability of detecting DDoS attacks. Amal M. AI-Eryani et al. [6] evaluated the performance of a variety of ML algorithms on the CIC-DDoS2019 dataset, especially the Gradient Boosting (GB) & XGBoost algorithms, and found these algorithms performed well in accuracy and with a low false positive rate. The result showed the accuracy of GB and XGBoost reached 99.99% & 99.98%. And it also highlighted the importance of continuously updating and improving detection technology to maintain the ability to reduce the likelihood of novel DDoS attacks. With this thesis, Strengthen the Capability of Machine Learning Models for Detecting DDoS Attack Based on Hyperparameter Tuning is proposed to increase the performance of ML models in detecting DDoS attacks. The research aims to improve the efficiency while maintaining the efficiency by optimizing the hyperparameter of ML models. The proposed approach will be evaluated by using the CIC-DDoS2019 dataset, which has a diverse array of network traffic data. The results obtained from this research will furnish important insights into the utility of our approach and its potential to bolster DDoS attack detection.

The remaining chapters are organized as follows: Chapter 2 provides related literature on DDoS attacks, AI , use of ML models, loss function, and hyperparameters tuning for DDoS detection. Chapter 3 indicates the methodology used in this research, including the dataset used and our approach for optimizing the hyperparameter of ML models. Chapter 4 details the results & discusses the findings. Finally, Chapter 5 analyzes the implications of our results and proposes future research directions along with improvements to DDoS defense approaches.

# Chapter 2 Background & Related Work

Faced with increasingly complicated and recurring DDoS attacks, traditional defense mechanisims often fall short in effectively addressing these threats. The combination of AI and ML techniques has emerged as a promising soultion, offering a more adaptive and strong scheme. This section introduces the present research on the AI, ML, DDoS attacks, and Hyperparameter Tuning that are related to the research topic.

## 2.1 Artificial Intelligence (AI)

The proliferation and advancement of DDoS threats have led to significant advancements in defense strategies, particularly those leveraging AI and statistical methods. Khalaf, B. A. et al. [7] delivered a review of various DDoS attack & defense methodologies, focusing on the limitations of existing approaches and emphasizing the need for dynamic analysis of attack patterns to enhance the techniques of mitigation. Their study summarized lots of DDoS defense mechanisms based on AI and Statistics, thus serving as a critical reference for researchers aiming to formulate more robust DDoS defenses.

In [8], the application of AI technology in intrusion detection is reviewed, especially in terms of how to promote system effectiveness by using feature selection and data analysis. The author deleted the Irrelevant data for filtering, applied multiple clustering methodologies for grouping data based on similarity, and selected the most representative subset of features to improve classification accuracy. Although traditional methods face limitations, advancements in artificial intelligence can lead to more robust, faster, and real-time intrusion detection systems (IDS). The research showed that AI enhances the capability of classifying the connection of the Internet, and the strategy of data reduction can shorten processing time and improve detection rate.

The increasing prevalence of DDoS attacks imposes considerable risks upon the operational continuity and accessibility of internet-based services. Suhag & Daniel [9] provided valuable insights into the application of statistical techniques and AI methods for identifying and mitigating DDoS attacks. Their research reviewed lots of leature of DDoS attacks, systematized the existing defense mechanisms based on AI and statistical methods, and also evaluated the utility of these approaches for DDoS attacks detection. In this research they discussed several statistical methods, such as Regression Analysis, statistical Modeling, Descriptive Statistics, and Comparative Analysis. And the result showed that the AI and statistical methods can effectively detect and mitigate DDoS attacks.

In the realm of cybersecurity, particularly in the detection of DDoS attacks, Antoni Jaszcz and Dawid Polap [10] introduced an innovative framework known as Artificial Intelligence Merged Methods (AIMM). This framework was designed to enhance detection capabilities through a hybrid approach that integrates multiple AI techniques. The authors proposed a three-module system comprising data preprocessing, classification, and decision-making. The first module effectively reduces the data volume while maintaining quality by aggregating information over specific time intervals. For classification, the framework employs two ML algorithms: K-Nearest Neighbors (KNN) & Artificial Neural Networks (ANN) for classifying network traffic as normal or abnormal. The third module consolidates the probabilities from these classifiers by using soft set inference and weighted averaging techniques to determine the potential attacks. And the results on the publicly available Bilkent University Netowrk (BOUN) dataset showed that the AIMM framework attained an impressive accuracy rate of 99.5% .

Nachaat Mohamed [11] provided a comprehensive review of vairous AI-based methods aimed at strengthening cybersecurity measures against threats. This study categorized AI techniques into ML algorithms and DL frameworks, such as SVM and Random Forest. The author analyzed their strengths and constraints in the real-world applications of DDoS detection and mitigation. He also discussed the integration of AI with traditional security protocols to reinforce the defensive mechanisms. And the experimental results showed that AI-driven systems significantly outperform traditional ones in both accuracy and response time. By reviewing the current state of AI in cybersecurity, this study offered valuable insights into AI techniques for applying within different professional fields.

## 2.2 Machine Learning (ML)

In [12], four classifiers, such as J48, Multi-Layer Perceptron (MLP), Random Forest (RF), and Naïve Bayes (NB), were utilized by Parvinder Singh Saini et al. With the Ubuntu 19.04 platform, the dataset which was collected by Mouhammd Alkasassbeh et al. This dataset is comprised of 27 features, including Source Address (SRC ADD), Destination Address (DES ADD), Packet Identifier (PKT ID), and so on. They used Waikato Environment for Knowledge Analysis (WEKA) to apply those four classifiers and visualized the results as five different classes: Normal, Smurf, User Datagram Protocol (UDP) Flood, HyperText Transfer Protocol (HTTP) Flood & Sustained Internet Distributed Denial of Service (SIDDOS). This study demonstrated that classifier J48 outperformed the other three algorithms, achieving an accuracy of 98.64%.The remaining classifiers yielded accuracies of 98.63%, 98.10%, and 96.93% correspondingly.

DDoS attacks pose significant challenges to the security of network systems, particularly in emerging Software-Defined Networking (SDN) environments. Numerous studies have explored techniques to identify and counter DDoS attacks within SDN environments, highlighting various strategies & algorithms to address this issue. In [13], a ML-driven approach to detecting DDoS attacks in SDN networks is introduced. The Authors conducted the experiment in a SDN environment simulated with Mininet 2.2.2 and used a Python-based OpenFlow (POX) controller and the Scalable Packet Manipulation (Scapy) tool to simulate the DDoS attacks. They utilized three different attacks: a flooding attack, a controller attack, and a bandwidth attack. By implementing four ML algorithms : Support Vector Machine (SVM), Multi-Layer Perceptron (MLP), Decision Tree & RF, they aimed to classify these attacks effectively.

Mahmood. A. AI. Shareeda et al. [14] reviewed various ML techniques, including Naive Bayes, SVM, Decision Tree, Artificial Neural Netowrks (ANN), etc. Each technique was evaluated based on its capability to classify and detect DDoS attack effectively. Futhermore, the author explored Deep Learning (DL) approaches that offer enhanced capabilities for feature extraction & classification in complex network environments. The author emphasized choosing the appropriate methodologies varies according to particular use cases, available resources, and the requirements for accuracy. And the result showed the effectiveness of different ML & DL techniques for DDoS detection.

With the growing dependence on cloud computing platforms, the challenge of recognizing DDoS attacks has attracted considerable attention within the field. This study, conducted by Mona Alduailij et al. [15], offered a novel approach to DDoS attack detection by employing ML techniques that aim to enhance detection accuracy while minimizing misclassification errors. The authors proposed a method that utilizes Mutual Information (MI) and Random Forest Feature Importance (RFFI) for feature selection, subsequently applying various ML algorithms, including RF, Gradient Boosting (GB), Weighted Voting Ensemble (WVE), KNN, and LR. The outcomes showed that these models attained an impressive accuracy rate of 99% when trained on a subset of 19 features. Remarkably, the RF model exhibited superior performance by misclassifying only one attack as normal traffic.

In [16], Murk Marvi et al. proposed a generalized model for modifying the performance of the ML models in these days. They used the CIC-DDoS2019 (Canaian Institute for Cybersecurity) dataset, spilt it into an 80:20 ratio for training. To prevent the new DDoS attacks in near real time, they tested the performance for detection with the records of unobserved DDoS attacks. For the feature engineering process, they employed Exploratory Data Analysis (EDA) to analyze the dataset and identify the most relevant features. Additionally, the Data Type Optimization (DTO) was performed to optimize the memory usage. The model was trained by a gradient boosting algorithm, and the result shows that the model achieved an accuracy of 99.9%. At the end of this study, the authors contrasted the proposed model with existing DL models by using the CICDS2017 dataset.

## 2.3 Distributed Denial of Service attack (DDoS attack)

Jai Dalvi et al. [17] proposed a framework to ascertain DDoS attacks by using Artificial Neural Networks, using the machine learning algorithm Mutual Information Classifier. They proposed a flow diagram of the proposed work for themselves. With the procedures of their work, they have tons of phases. The first phase, Input: they collected the dataset from open-source with all features. In the second phase, they did the data preprocessing to transfer data into pre-processed data which is used for the Mutual Information Classifier as input data. And the data was split as 70% training and 30% testing. Finally, the model is trained and tested on the input data, with the output data presented in the form of normal packets and anomalous packets. The reason they used a Mutual Information Classifier is that only crucial features were selected to train the model. It could make the model run adequately, and their results showed that the time complexity of the model was reduced by nearly half. In addition, the accuracy of the model slightly increased from 89.6% to 89.62%.

Latha R et al. [18] proposed a procedural framework to recognize the DDoS attack. On top of the framework, they used Naïve Bayes (NB) and Logistic Regression to perform the classification of the DDoS attack and also compare the performance of both algorithms. For the dataset, they used the Knowledge Discovery and Data Mining Cup (KDDCup) dataset which has 41 attributes such as duration, protocol_type, service, flag, and so on. They are distinguished into two different stages: the attack stage and the normal stage. They calculated the throughput and thresholds as the dataset of the Logistic regression and NB models. The total number of dataset is 20,090, and is divided into training, testing, and validation data. Finally, they performed several metrics to estimate the outcome of Logistic Regression and NB.

Ahamed Aljuhani [19] presented a comprehensive analysis of ML approaches for integrating DDoS attacks in present networking environments. It discussed various ML techniques employed in DDoS defense systems, particularly in Cloud Computing, SDN, and Internet of Things (IoT) environments. It also categorized recent research findings, assessed the effectiveness of different ML models, and identified challenges for future research in developing robust defense mechanisms against DDoS threats.

In [20], Firooz B. Saghezchi et al. proposed a detection system for Industry 4.0 Cyber-Physical Production System (CPPS). They used NetMate to extract the set of statistical network flow features in the Comma-Separated Values (CSV) format and Packet Capture (PCAP) files. The dataset of the CSV file has 40000 rows and 46 columns. After they did the pre-analysis of the training set, the authors eliminated some attributes that were irrelevant to the model. Eventually, they evaluated various kinds of algorithms, revealing that supervised methods, particularly the Decision Tree, demonstrate superior accuracy. It achieved an accuracy of 0.999 while maintaining a false positive rate of 0.01. This study fills the gaps in existing methods and provides practical insights for DDoS detection in industrial environments.

In [21], Abdullah Emir Cil et al. suggested a framework for classifying DDoS attacks utilizing the feedforward Deep Neural Networks (DNN). The authors employed the CIC-DDoS2019 dataset to train the model. And the experimental methodology is particularly noteworthy: the DNN architecture includes multiple hidden layers that facilitate both feature extraction and classification processes. This allows the model to operate effectively even with smaller sample sizes. For the training process, they used a binary cross-entropy loss function, optimizing the model's outcome through the AdaMax optimizer. Results from the experiments showed that the proposed DNN model achieved 99.97% accuracy and 99.99% precision for detection and classification.

## 2.4 Hyperparameter Tuning

Odnan Ref Sanchez et al. [22] explored the effect of traditional ML method for classifying DDoS attacks, with a focus on optimizing performance through exhaustive hyperparameter tuning using GridSearchCV. Unlike recent trends that emphasize DL approaches, this study investigated the potential of lightweight ML alogrithms such as NB, LR, Decision Trees, RF, KNN, SVM & Multi-Layer Perceptron. In order to achieve high detection accuracy with reduced computational overhead, it is appropriate for environments that lack resource like IoT. The ML models were trained and evaluated using the CIC datasets. The results demonstrated that Random Forests & Decision Trees, when optimized, can achieve accuracy levels comparable to DL methods, exceeding 98% across multiple dataset. This work highlights the value of hyperparameter optimization in enhancing the outcome of traditional ML techniques for DDoS classification.

In [23], Hijrah Nisya et al. investigated the application of a hyperparameter-tuned method for DDoS attack classification within SDN. Addressing the security vulnerabilities of SDN's centrailized control, they proposed a novel approach that optimizie the RF algorithms' performance to enhance detection accuracy. This study employed the InSDN dataset and utilized SelectFromModel (SFM) for keeping the most relevant traffic features. And hyperparameters tuning was conducted by using Random Search to find optimal parameter configurations for the RF model. The experimental results showed that hyperparameter tuning siginificantly improves the RF algorithm's accuracy, achieving a detection rate of 99.99%.

Recent studies have increasingly focused on ML-based intrusion detection systems to combat the growing threat of DDoS attacks. For instance, Sandeep Dasari and Rajesh Kaluri [24] introduced a hierarchical machine learning approach combined with hyperparameter optimization for effective DDoS attack classification in distributed networks. Addressing the critical need for data privacy in the financial sector, this study leveraged the CICIDS 2017 dataset, employing Min-Max scaling and Synthetic Minority Oversampling Technique (SMOTE) for preprocessing, and Least Absolute Shrinkage and Selection Operator (LASSO) for feature selection. The proposed method utilized a tiered ensemble of algorithms, including XGBoost, LightGBM (LGBM), CatBoost, RF, and Decision Tree, with algorithm's hyperparameters meticulously tuned to enhance performance. The results indicated that the effectiveness of the LGBM classifier, attaining a 99.77% accuracy in identifying DDoS attacks. This work contributed a novel method to intrusion detection by strategically integrating feature selection, hierarchical ML, and hyperparameter optimization, offering a robust solution for mitigating the growing threat of cyberattacks in modern networks.

In [25], Tan May May et al. investigated the optimization of ML and DL models for IDSs by focusing on hyperparameter tuning. Recognizing the high false rates & the increasing sophistication of cyberattacks posed significant challenges, the authors explored the use of RF, DNNs, and Deep Autoencoders (DAES) with a goal of improving detection accuracy. The research used the CIC-IDS2017 dataset and employed Pearson correlation for feature selection, QuantilTransformerScaler for data scaling, and grid search for hyperparameter optimization. And the results showed a significant performance improvement, achieving an average accuracy of 99.5% across all models, suggesting that the selected hyperparameters are effective for identifying normal & abnormal network traffic.

Abdussalam Ahmed Alashhab et al. [26] proposed an integrated Online Machine Learning (OML) framework to ascertain & mitigated DDoS attacks in SDN environments. To deal with the evolving nature of DDoS threats, the approach leveraged a combination of four OML classifiers: Bernoulli Naive Bayes, Passive-Aggressive, Stochastic Gradient Descent (SGD) , and MLP. This framework was designed to adapt dynamically to new attack patterns through continuous learning, enhancing its resilience against zero-day and low-rate attacks. In the end, the experimental results using datasets, including CIC-DDoS2019 and InSDN, demonstrated a detection rate of 99.2%, outperforming comparable models.

# Chapter 3 Implementation

Recognizing and moderating DDoS attacks [27] is a critical challenge in the network security field. The CIC-DDoS2019 dataset [28] provides a thorough benchmark for assessing DDoS attack classification models. To leverage the data effectively, we developed a structured framework for data processing, feature engineering, and model optimization.

## 3.1 Methodology

Here, we present the approach used to execute, evaluate, and optimize the ML-based DDoS classification models. This section can be divided into four subsections: Machine Learning Pipeline for DDoS Detection, Data Loading and Preprocessing, Data Splitting, and Model Training, Evaluation, and Mitigation. We will discuss the software & hardware stack and three significant components of the proposed framework in detail.

### 3.1.1 Machine Learning Pipeline for DDoS Detection

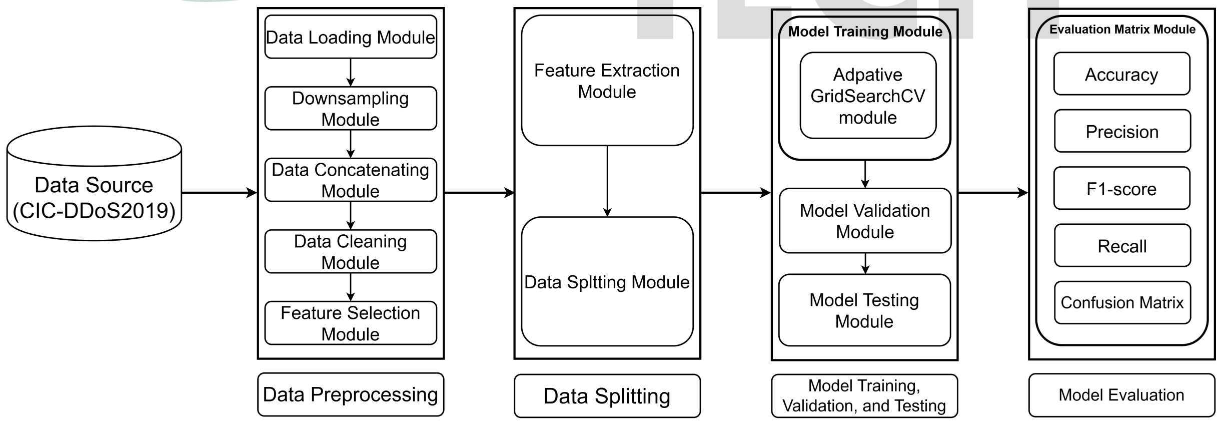


Figure 3.1: System Architecture of Enhancing ML-Based DDoS Detection.

The architecture we proposed include data preprocessing, data splitting, model training, validation & testing modules, and model evaluation (see figure 3.1) .

(A) **Data Preprocessing**

The first phase is: data preprocessing, during this phase, the data loading module is used to load the CIC-DDoS2019 dataset into chunks. Then we will label the data with flows_ok and flows_ddos, and data downsampling module will be used to balance the data. After that, the data cleaning module will replace NA/NaN value / infinty values with 0, and convert the ”flows_ok” label to 0 and the ”flows_ddos” label to 1. At the end of this phase, the feature selection module will be used to delete the irrelevant columns to keep the data clean and ready for the next step.

(B) **Data Splitting**

The second phase is: data splitting, in this stage we will separate the data into training & validation sets. First, the feature extraction module will extract the data to different labels based on the features on the data. For example the first 82 columns are the features, and the last column is the label. The data will be split into two parts: the feature set (X), containing only the first 82 columns, and the label set (y), containing only the last column. Then the data splitting module will separate the data into training & validation sets, with the ratio of 67% and 33%.

(C) **Model Training, Validation, and Testing**

The third phase is : model training, validation, and testing. In this phase, we will train three different ML models (SVM, LR, and KNN) and evaluate their performance with validation and testing sets. The model training module will take as input the feature set $(X_{\text{train}})$ and label set $(y_{\text{train}})$ to train the models, which is optimized by using the adaptive GridSearchCV method. This method iteratively explores a predefined parameter space, refining the search boundaries based on feedback to improve model performance. The model validation module and model testing module will estimate the models’ outcome using the validation set and testing sets.

(D) **Model Evaluation**

The last phase is: model evaluation. In this part, we will estimate the model outcome using five different metrics. The evaluation metrics module will calculate the accuracy, precision, recall, F1-score, and confusion matrix for each model. They help assess the models' outcome in classifying DDoS attacks and their capability to distinguish between normal & abnormal traffic.

### 3.1.2 Data Preprocessing

In this phase, we will introduce the flowchart of data preprocessing.

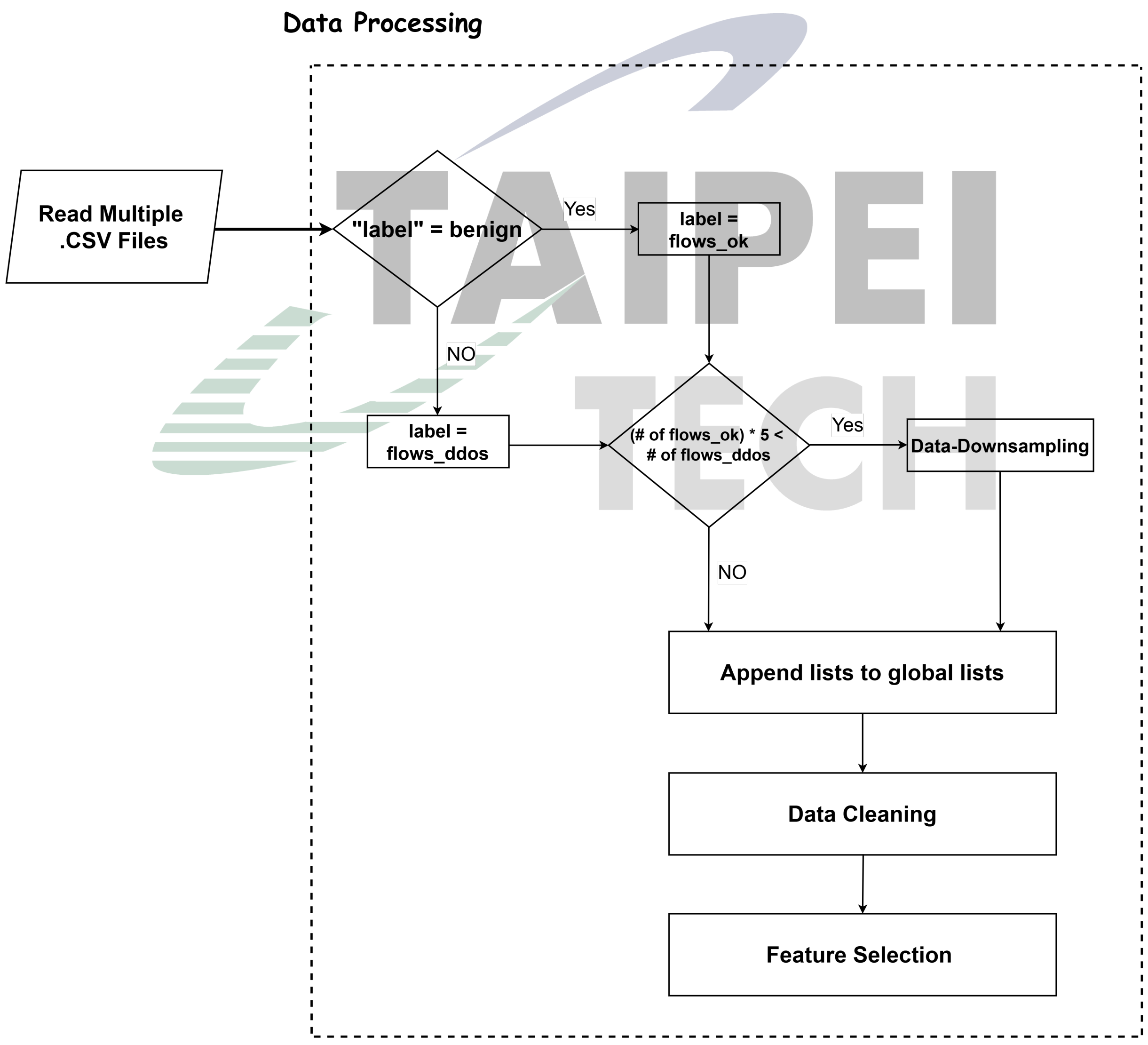


Figure 3.2: Data Preprocessing Framework for DDoS Detection.

Figure 3.2 shows the data preprocessing framework for DDoS detection. At the beginning of the procedure, the raw data from the CIC-DDoS2019 dataset is loaded. The dataset consists of two different folders: `01-12` and `03-11` (excluding UDP, UDPLag, these two files), which contain lots of `.csv` files with normal and malicious traffic.

Initially, global temporary lists are established to collect benign and attack flows from all CSV chunks, with a parameter `mult` set to ”5” (the reason is that we want to balance the number of benign and attack flows). Then the system reads multiple CSV files in chunks, examining each file for the label field to identify benign or attack flows.

When flows are identified as benign (`flows_ok`), the system checks whether the number of these flows is less than five times the number of attack flows (`flows_ddos`). If so, data downsampling is applied to extract only five times number of `flows_ok` from the `flows_ddos` ; otherwise, the original number of `flows_ddos` are retained.

Then both attack and bengin flows are appended to global lists, which are concatenated into a single dataframe. We will continue to do data cleaning (such as replacing the NA/NaN with 0 and replacing infinity with 0) and type conversion (such as converting the `"Benign"` to 0, otherwise 1, and converting the timestamp to an integer type by using UTF-8 encoding & MD5 hashing). And the last process is feature selection, we delete some features that are irrelevant to the DDoS detection, such as `"Source IP"`, `"Destination IP"`, `"Flow ID"`, `"Similar HTTP"`, and `"Unnamed: 0"` columns.

### 3.1.3 Data Splitting

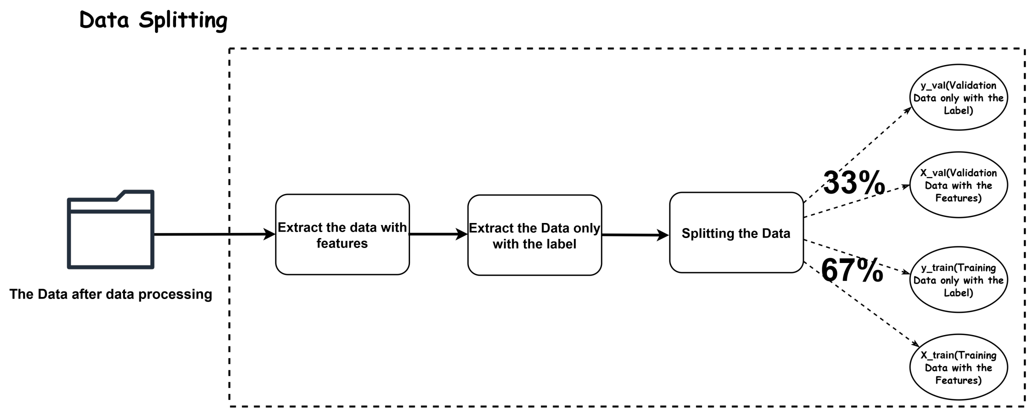

Data Splitting
The Data after data processing
Extract the data with features
Extract the Data only with the label
Splitting the Data
33%
67%
y_val(Validation Data only with the Label)
X_val(Validation Data with the Features)
y_train(Training Data only with the Label)
X_train(Training Data with the Features)

Figure 3.3: Data Splitting Framework for DDoS Detection.

Figure 3.3 illustrates the data splitting framework employed in this research. The processed data was first loaded into the system. Feature extraction was then performed on the dataset, which consists of 83 columns in total. The first 82 columns represent the input features, while the last column indicates the label (i.e., whether the flow is a DDoS attack, labeled as 1, or benign, labeled as 0). The data was subsequently separated into two parts: the feature set $X$, containing the first 82 columns, and the label set $y$, containing only the last column.

After extraction, these feature and label sets were further separated into training & validation sets, with 67% training $(X_{\text{train}}, y_{\text{train}})$ and 33% validation data $(X_{\text{val}}, y_{\text{val}})$. This division ensured that the model was trained on enough data to learning the pattern of attacks. The training set was used to fit the ML models, while the validation set was utilized to assess their validation outcome. This splitting strategy is crucial for evaluating the generalization capability of the models and preventing overfitting in the DDoS detection process.

### 3.1.4 Model Training and Mitigation

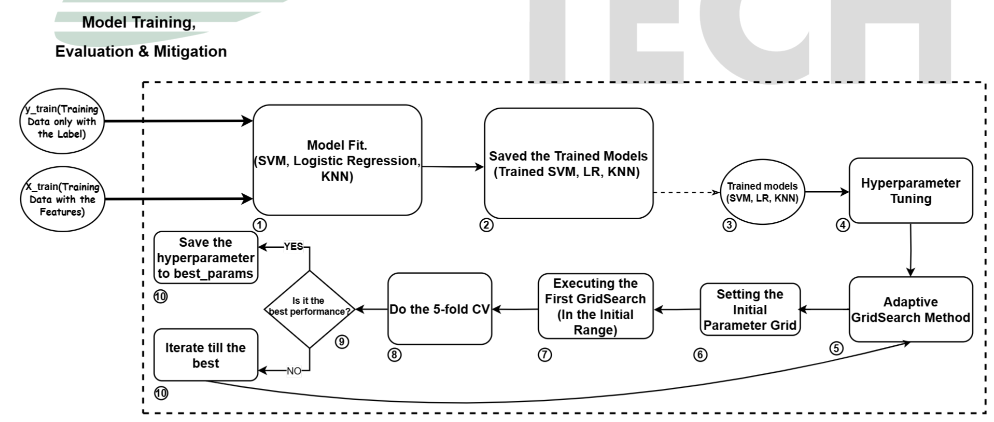

Model Training,
Evaluation & Mitigation
y_train(Training Data only with the Label)
X_train(Training Data with the Features)
Model Fit. (SVM, Logistic Regression, KNN)
Saved the Trained Models (Trained SVM, LR, KNN)
Trained models (SVM, LR, KNN)
Hyperparameter Tuning
Adaptive GridSearch Method
Setting the Initial Parameter Grid
Executing the First GridSearch (In the Initial Range)
Do the 5-fold CV
Is it the best performance?
YES
NO
Save the hyperparameter to best_params
Iterate till the best
①
②
③
④
⑤
⑥
⑦
⑧
⑨
⑩
⑩

Figure 3.4: Iterative Hyperparameter Tuning Methodology for ML Models Optimization.

Figure 3.4 indicates the systematic process of ML model development, from the training phase to the evaluation and mitigation phase. The pipeline begins with the separation of training and testing datasets (X_train, y_train, X_test, y_test), followed by fitting using multiple ML algorithms (SVM, Logistic Regression (LR), KNN).

A critical component is the hyperparameter tuning cycle, implemented through GridSearch methodology with 5-fold cross-validation (with 4 folds used for training and 1 fold for validation). After the model is trained, the performance metrics are calculated to estimate the model's outcome. If the hyperparameter combination is not optimal, the process is repeated until the best-performing model is identified. Once the optimal hyperparameters are determined, the combination will be saved to the best_params variable. The final model is then estimated by using the testing dataset, and the metrics are calculated. This iterative process ensures that those three models have the best hyperparameters and performance metrics for more effective DDoS attack detection.

### 3.1.5 Model Validation and Evaluation

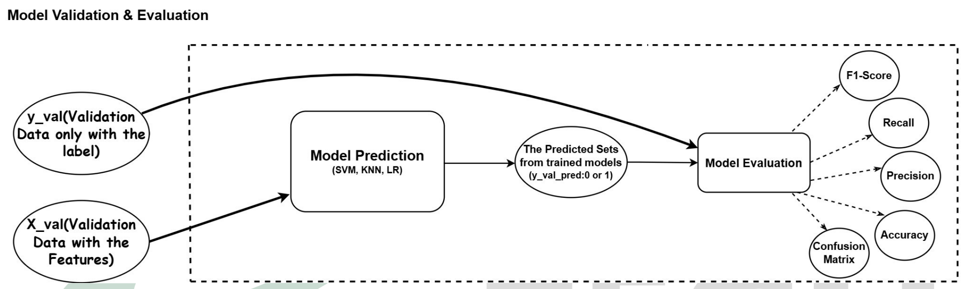

Model Validation & Evaluation
y_val(Validation Data only with the label)
X_val(Validation Data with the Features)
Model Prediction
(SVM, KNN, LR)
The Predicted Sets from trained models (y_val_pred:0 or 1)
Model Evaluation
F1-Score
Recall
Precision
Accuracy
Confusion Matrix

Figure 3.5: Framework for Model Validation and Evaluation.

Figure 3.5 shows the process of model validation. After we separate the data, we use validation data as the input for the models, which is mitigated by the Adaptive GridSearch method. These models will be used to predict the set of the validation data (y_val_pred), and the predicted labels will be compared with the actual labels (y_val) to assess the model's outcome. The outcome indicators, which are estimated to assess the model's effectiveness in detecting DDoS attacks.

### 3.1.6 Model Testing and Evaluation

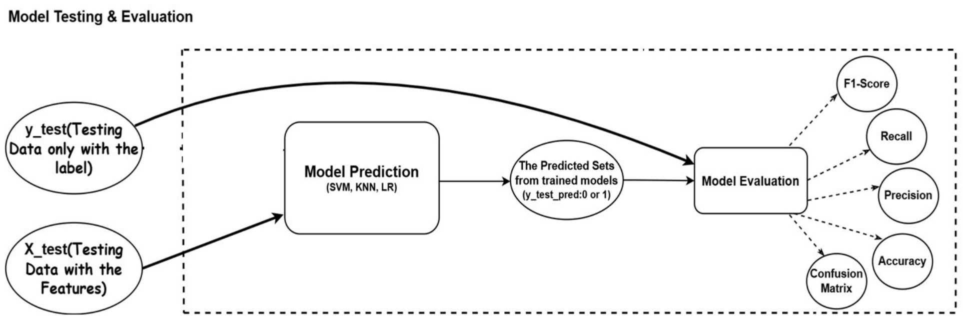

Model Testing & Evaluation
y_test(Testing Data only with the label)
X_test(Testing Data with the Features)
Model Prediction (SVM, KNN, LR)
The Predicted Sets from trained models (y_test_pred:0 or 1)
Model Evaluation
F1-Score
Recall
Precision
Accuracy
Confusion Matrix

Figure 3.6: Framework for Model Testing and Evaluation.

Figure 3.6 shows the process of model testing and evaluation. After we validate the model, we will use the testing set as the input for the models, which is mitigated by the Adaptive GridSearchCV method. These models will be utilized to generate the set of the testing data (y_test_pred), and the predicted labels will be compared with the actual labels (y_test) to estimate the model's performance. The outcome indicators are shown as a figure, which help us to assess the model's effectiveness.

## 3.2 Environment Setup

In this section, it outlines requirements for implementing the proposed framework, including hardware & software requirements, third-party software tools, and custom-developed scripts. For the rest of this section, we discuss the establishment of the experimental environment for the DDoS detection models and the practical steps for implementing the proposed framework.

### 3.2.1 Software and Hardware Requirements

Table 3.1 illustrates the hardware requirements utilized in this research. The system is powered by a 13th Gen Intel(R) Core(TM) i7-13700 processor, which provides the computational capabilities for data processing and model training. To handle the extensive memory require-

Table 3.1: Hardware Requirements

| Name | Description |
|---|---|
| CPU | 13th Gen Intel(R) Core(TM) i7-13700 |
| Memory | 32GB |
| Disk | SAMSUNG MZVL2512HCJQ-00B00 |
| GPU | NVIDIA GeForce RTX 4060 |

ments of machine learning operation and large-scale datasets, the system is equipped with 32GB of RAM. The storage system is managed by a SAMSUNG MZVL2512HCJQ-00B00 solid-state drive, ensuring efficient data access and retrieval. Additionally, the NVIDIA GeForce RTX 4060 GPU is employed to accelerate model training, optimization, and inference processes, particularly beneficial for handling large-scale DDoS detection workloads. These hardware requirements are carefully selected to ensure optimal performance during the training and evaluation of our proposed detection models.

Table 3.2: Third-Party Software Requirements

| Name | Description | Version | License |
|---|---|---|---|
| Visual Studio Code [29] | Integrated Development Environment | 1.97.2 | MIT License |
| Jupyter Notebook [30] | Interactive Computing Environment | 7.4.2 | BSD License |
| Python [31] | Programming Language | 3.12.0 | PSF License |
| NumPy [32] | Scientific Computing Library | 2.0.2 | BSD License |
| Pandas [33] | Data Manipulation Library | 2.2.3 | BSD License |
| Keras [34] | Deep Learning Framework | 3.7.0 | MIT License |
| Psutil [35] | Used to retrieve information on running processes & system utilization | 6.1.1 | BSD License |
| Scikit-Learn [36] | Machine Learning Library | 1.6.1 | BSD License |
| Tqdm [37] | Progress Bar Library | 4.67.1 | MIT License |

Table 3.2 presents the third-party software requirements utilized in our experimental implementation. The development environment was built upon VSCode, which served as the primary IDE, complemented by Jupyter Notebook for interactive code development and visualization. For the code data processing and analysis capabilities, our implementation leveraged NumPy for scientific computing operations and Pandas for efficient data manipulation and preprocessing tasks.

The ML models were implemented utilizing multiple libraries from the scikit-learn ecosystem, including specialized modules for Support Vector Machine (sklearn.svm), Logistic Regression (sklearn.linear_model), and K-Nearest Neighbors (sklearn.neighbors) algorithms. Additional scikit-learn modules were employed for data preprocessing (sklearn.preprocessing), model selection (sklearn.model_selection), and performance evalution (sklearn.metrics).

And the system resource monitoring was facilitated by Psutil, while Tqdm was utilized to track processing progress during computationally intensive operations. This comprehensive third-party software configuration enabled efficient implementation of the proposed framework, providing the necessary tools for data processing, model development, performance evaluation, and mitigation of models' performance.

Table 3.3: Software Requirements

| **Name** | **Description** |
|---|---|
| Data Preprocessing and Cleaning.ipynb | Loading & Merging Dataset, Data cleaning, resampling, converting, normalizing, and Splitting |
| Model Training, Evaluation, and Mitigation.ipynb | Executing model training, evaluating the performance on the vaidation set & testing set, and reforcing the capability of ML models |

Table 3.3 lists the sofwares which are developed for the implementation of the proposed framework. From the beginning of the data preprocessing and cleaning phase to the model training, evaluation, and mitigation phase, two Jupyter Notebooks are created to facilitate the implementation of the DDoS detection models.

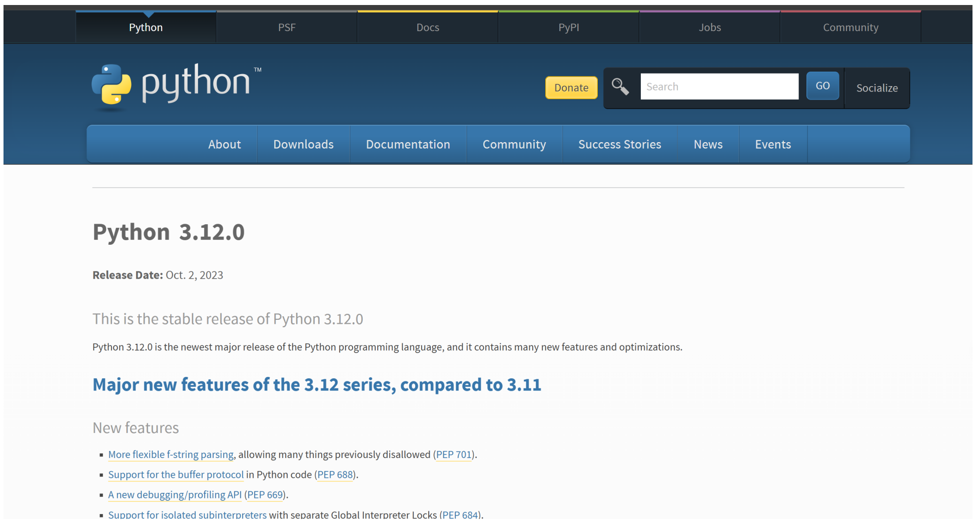
The Data Preprocessing and Cleaning.ipynb focuses on the crucial initial step of data preprocessing and cleaning for the CIC-DDoS2019 dataset. Utilizing Pandas, Numpy, and Scikit-learn libraries, this noteboook addresses the challenges posed by the dataset's scale and class imbalance. There are lots of core functions such like : `load_file`, `load_huge_file`, `resample_data`, `string2numeric_hash`, `normalize_data`, and `train_test`. For the `load_file`, it loads the data from the CIC-DDoS2019 dataset and merges the files into a single dataframe. The `load_huge_file` function is used to load large-scale data files, while the `resample_data` function is utilized to balance the class distribution of the dataset. The `string2numeric_hash` converts infinity & NaN to '0', 'Flow Packets/s' & 'Flow Bytes/s' from string to numeric, and convert 'Timestamp' to UTF-8 & hash value. And it also replace the 'Benign' to '0' and 'DDoS' to '1' for the label. The `normalize_data` function normalizes the data to the range of -1 to +1. And the `train_test` function splits the data into training and testing sets in a ratio of 67% and 33%.

For the Model, Training, and mitigation.ipynb, it concentrated on the model training, evaluation, and mitigation phases. This notebook leverages the Scikit-learn library to implement the SVM, LR, and KNN models for DDoS detection. The key functions include: `train_model`, `evaluate_model`, and `optimize_model`. The `train_model` function trains the ML models using the training dataset by fitting the model to the training data. The `evaluate_model` function estimates the models' outcome using the testing dataset to calculate. The `optimize_model` function performs hyperparameter tuning using the adaptive GridSearch methodology, aiming to identify the optimal hyperparameters for each model. These two .ipynb files are crucial components of the proposed framework for DDoS detection, integrating advanced ML techniques to improve the outcome and efficacy.

### 3.2.2 The Establishment of the Environments

(A) **Install Python**

1. Go to the official Python website : [figure 3.7]



Python 3.12.0

**Release Date:** Oct. 2, 2023

This is the stable release of Python 3.12.0

Python 3.12.0 is the newest major release of the Python programming language, and it contains many new features and optimizations.

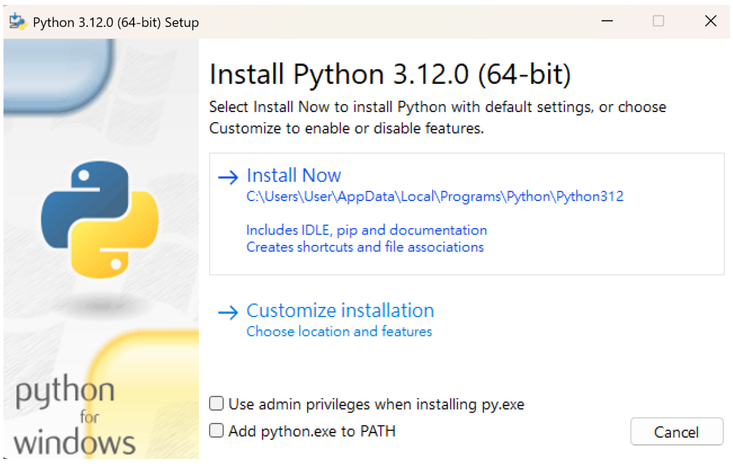
**Major new features of the 3.12 series, compared to 3.11**

New features

- More flexible f-string parsing, allowing many things previously disallowed (PEP 701).
- Support for the buffer protocol in Python code (PEP 688).
- A new debugging/profiling API (PEP 669).
- Support for isolated subinterpreters with separate Global Interpreter Locks (PEP 684).

Figure 3.7: Python Environment Setup for DDoS Detection.

2. Download the Python 3.12.0 : [figure 3.8]


Python 3.12.0 (64-bit) Setup

Install Python 3.12.0 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\User\AppData\Local\Programs\Python\Python312

Includes IDLE, pip and documentation
Creates shortcuts and file associations

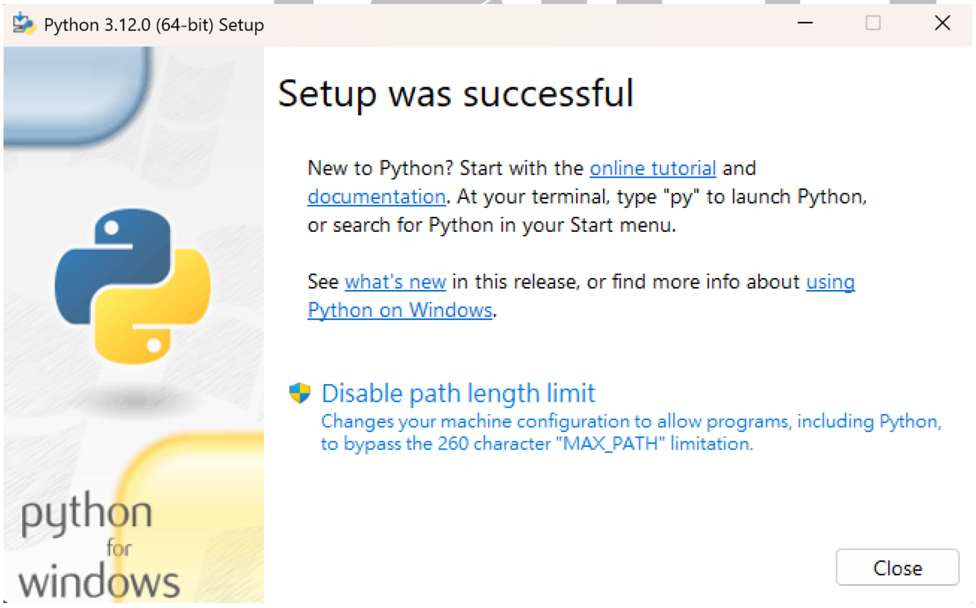
→ Customize installation
Choose location and features

☐ Use admin privileges when installing py.exe
☐ Add python.exe to PATH

Figure 3.8: Python Installer Initial Screen.

Figure 3.8 Python intaller initial screen. Note: It's recommended to check the "Add python.exe to PATH" option to enable using Python from the command line.

3. Execute the installer and follow the steps to install Python. [Figure 3.9 and 3.10]


Python 3.12.0 (64-bit) Setup

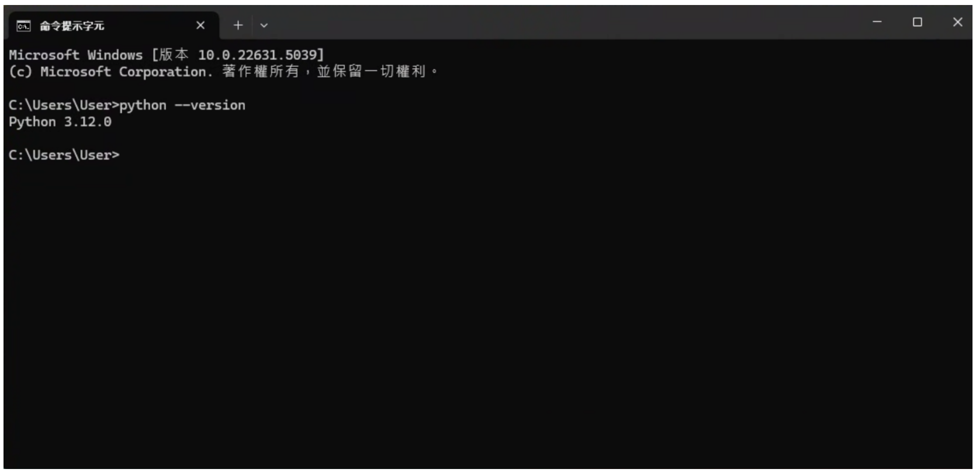
Setup was successful

New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

See what's new in this release, or find more info about using Python on Windows.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

Figure 3.9: Python installation successful completion screen.



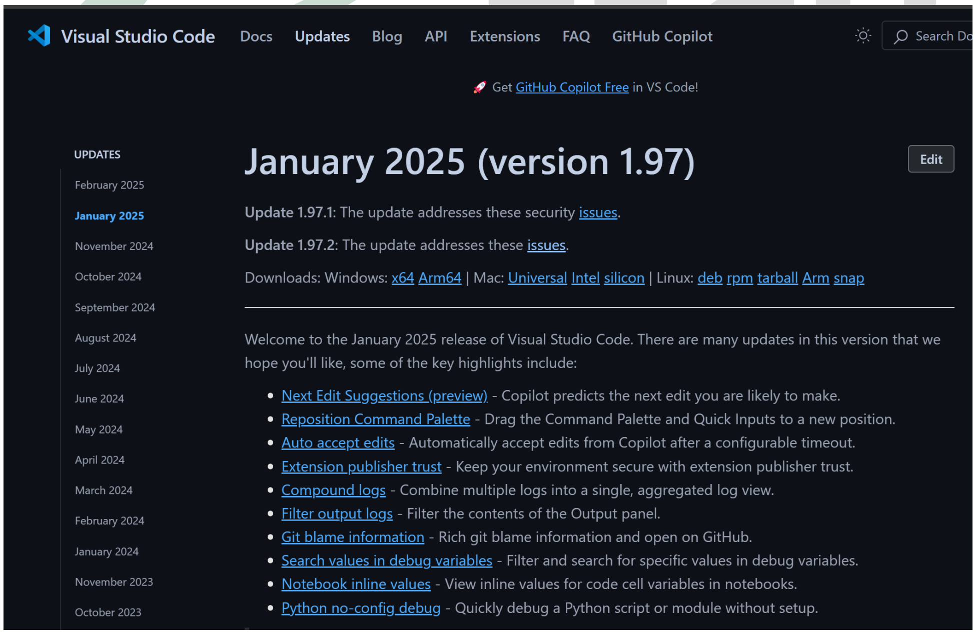
```
Microsoft Windows [版本 10.0.22631.5039]
(c) Microsoft Corporation. 著作權所有，並保留一切權利。

C:\Users\User>python --version
Python 3.12.0

C:\Users\User>
```

Figure 3.10: Verifying Python installation.

Figure: 3.10 Verifying Python installation in the command prompt using the ”python –version” command.

(B) **Install Visual Studio Code**

1. Go to the official Visusl Studio Code website: [figure 3.11]



Get GitHub Copilot Free in VS Code!

UPDATES
February 2025
January 2025
November 2024
October 2024
September 2024
August 2024
July 2024
June 2024
May 2024
April 2024
March 2024
February 2024
January 2024
November 2023
October 2023

January 2025 (version 1.97)

**Update 1.97.1**: The update addresses these security issues.

**Update 1.97.2**: The update addresses these issues.

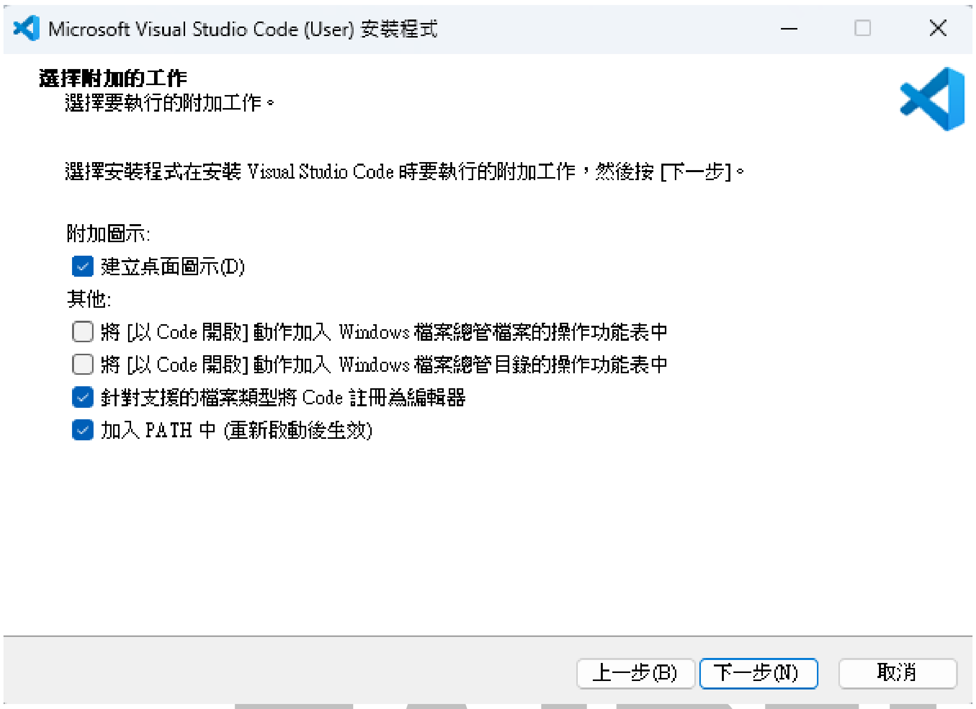
Downloads: Windows: x64 Arm64 | Mac: Universal Intel silicon | Linux: deb rpm tarball Arm snap

Welcome to the January 2025 release of Visual Studio Code. There are many updates in this version that we hope you'll like, some of the key highlights include:

- Next Edit Suggestions (preview) - Copilot predicts the next edit you are likely to make.
- Reposition Command Palette - Drag the Command Palette and Quick Inputs to a new position.
- Auto accept edits - Automatically accept edits from Copilot after a configurable timeout.
- Extension publisher trust - Keep your environment secure with extension publisher trust.
- Compound logs - Combine multiple logs into a single, aggregated log view.
- Filter output logs - Filter the contents of the Output panel.
- Git blame information - Rich git blame information and open on GitHub.
- Search values in debug variables - Filter and search for specific values in debug variables.
- Notebook inline values - View inline values for code cell variables in notebooks.
- Python no-config debug - Quickly debug a Python script or module without setup.

Figure 3.11: The VSCode Version Setup for DDoS Detection.

2. Download the Visual Studio Code 1.97.2: [Figure 3.12]


Microsoft Visual Studio Code (User) 安裝程式

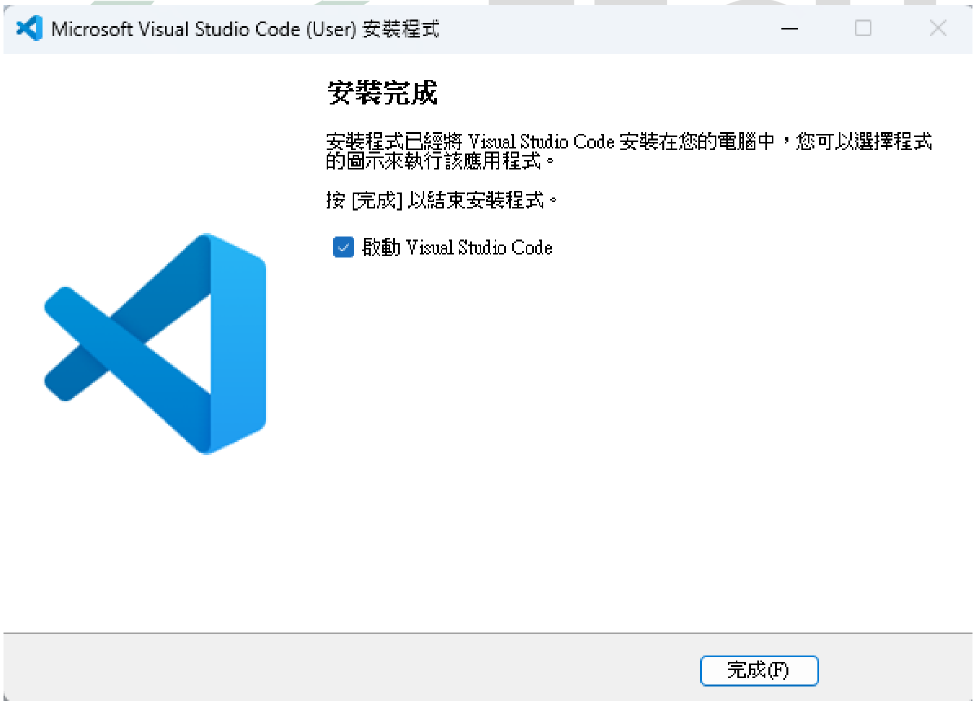
**選擇附加的工作**
選擇要執行的附加工作。

選擇安裝程式在安裝 Visual Studio Code 時要執行的附加工作，然後按 [下一步]。

附加圖示:
[✓ 建立桌面圖示(D)]
其他:
將 [以 Code 開啟] 動作加入 Windows 檔案總管檔案的操作功能表中
將 [以 Code 開啟] 動作加入 Windows 檔案總管目錄的操作功能表中
[✓ 針對支援的檔案類型將 Code 註冊為編輯器]
[✓ 加入 PATH 中 (重新啟動後生效)]

Figure 3.12: The VSCode Installer Initial Screen.

3. Execute the installer and follow the steps to install VSCode. [Figure 3.13]


Microsoft Visual Studio Code (User) 安裝程式

**安裝完成**

安裝程式已經將 Visual Studio Code 安裝在您的電腦中，您可以選擇程式的圖示來執行該應用程式。

按 [完成] 以結束安裝程式。

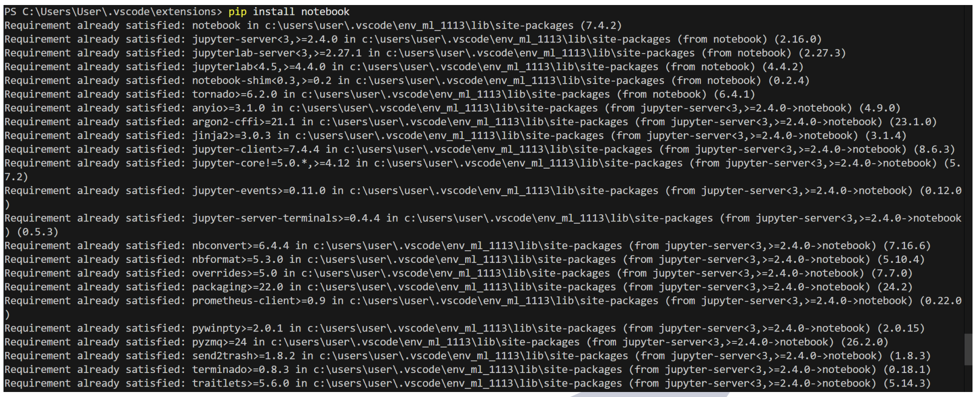
[✓ 啟動 Visual Studio Code]

Figure 3.13: The VSCode Installation Finished Screen.

Figure 3.13 shows the VSCode installation finished screen. The installation process is complete.

(C) **Extensions and Packages**

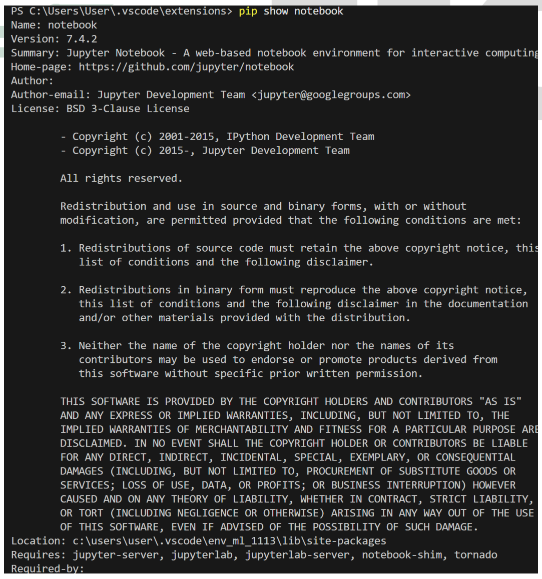
- **Python Extension**



```
PS C:\Users\User\.vscode\extensions> pip install notebook
Requirement already satisfied: notebook in c:\users\user\.vscode\env_ml_1113\lib\site-packages (7.4.2)
Requirement already satisfied: jupyter-server<3,>=2.4.0 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from notebook) (2.16.0)
Requirement already satisfied: jupyterlab-server<3,>=2.27.1 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from notebook) (2.27.3)
Requirement already satisfied: jupyterlab<4.5,>=4.4.0 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from notebook) (4.4.2)
Requirement already satisfied: notebook-shim<0.3,>=0.2 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from notebook) (0.2.4)
Requirement already satisfied: tornado>=6.2.0 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from notebook) (6.4.1)
Requirement already satisfied: anyio>=3.1.0 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (4.9.0)
Requirement already satisfied: argon2-cffi>=21.1 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (23.1.0)
Requirement already satisfied: jinja2>=3.0.3 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (3.1.4)
Requirement already satisfied: jupyter-client>=7.4.4 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (8.6.3)
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (5.
7.2)
Requirement already satisfied: jupyter-events>=0.11.0 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (0.12.0
)
Requirement already satisfied: jupyter-server-terminals>=0.4.4 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook
) (0.5.3)
Requirement already satisfied: nbconvert>=6.4.4 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (7.16.6)
Requirement already satisfied: nbformat>=5.3.0 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (5.10.4)
Requirement already satisfied: overrides>=5.0 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (7.7.0)
Requirement already satisfied: packaging>=22.0 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (24.2)
Requirement already satisfied: prometheus-client>=0.9 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (0.22.0
)
Requirement already satisfied: pywinpty>=2.0.1 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (2.0.15)
Requirement already satisfied: pyzmq>=24 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (26.2.0)
Requirement already satisfied: send2trash>=1.8.2 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (1.8.3)
Requirement already satisfied: terminado>=0.8.3 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (0.18.1)
Requirement already satisfied: traitlets>=5.6.0 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from jupyter-server<3,>=2.4.0->notebook) (5.14.3)
```

Figure 3.14: The Instruction for Installing Jupyter Notebook.

Figure 3.14 shows the instruction for installing Jupyter Notebook. We can install the Jupyter Notebook by using the command line. The command is : `pip install notbook`. After completing the installation, we can use the Jupyter Notebook in the VSCode.

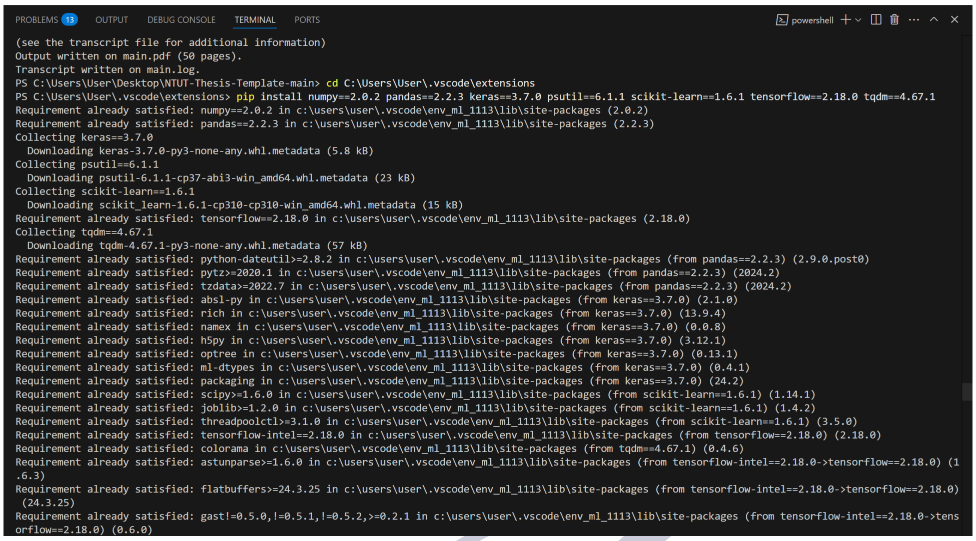


```
PS C:\Users\User\.vscode\extensions> pip show notebook
Name: notebook
Version: 7.4.2
Summary: Jupyter Notebook - A web-based notebook environment for interactive computing
Home-page: https://github.com/jupyter/notebook
Author:
Author-email: Jupyter Development Team <jupyter@googlegroups.com>
License: BSD 3-Clause License

        - Copyright (c) 2001-2015, IPython Development Team
        - Copyright (c) 2015-, Jupyter Development Team

        All rights reserved.

        Redistribution and use in source and binary forms, with or without
        modification, are permitted provided that the following conditions are met:

        1. Redistributions of source code must retain the above copyright notice, this
           list of conditions and the following disclaimer.

        2. Redistributions in binary form must reproduce the above copyright notice,
           this list of conditions and the following disclaimer in the documentation
           and/or other materials provided with the distribution.

        3. Neither the name of the copyright holder nor the names of its
           contributors may be used to endorse or promote products derived from
           this software without specific prior written permission.

        THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
        AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
        IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
        DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
        FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
        DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
        SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
        CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
        OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
        OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Location: c:\users\user\.vscode\env_ml_1113\lib\site-packages
Requires: jupyter-server, jupyterlab, jupyterlab-server, notebook-shim, tornado
Required-by:
```

Figure 3.15: The Instruction for Verifying the Version of Jupyter Notebook.

Figure 3.15 shows the instructions for verifying the version of Jupyter Notebook. We can verify the version by using the command line. The command is: `pip show notebook`. After we verify the version, we can use the Jupyter Notebook in the VSCode.

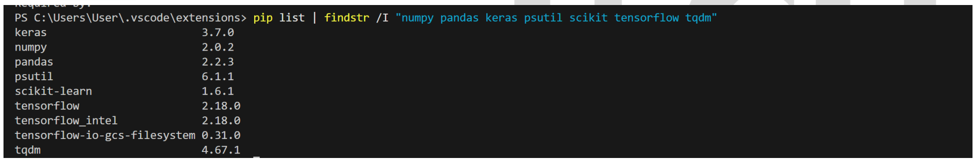
- **Python Packages**



```
(see the transcript file for additional information)
Output written on main.pdf (50 pages).
Transcript written on main.log.
PS C:\Users\User\Desktop\NTUT-Thesis-Template-main> cd C:\Users\User\.vscode\extensions
PS C:\Users\User\.vscode\extensions> pip install numpy==2.0.2 pandas==2.2.3 keras==3.7.0 psutil==6.1.1 scikit-learn==1.6.1 tensorflow==2.18.0 tqdm==4.67.1
Requirement already satisfied: numpy==2.0.2 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (2.0.2)
Requirement already satisfied: pandas==2.2.3 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (2.2.3)
Collecting keras==3.7.0
  Downloading keras-3.7.0-py3-none-any.whl.metadata (5.8 kB)
Collecting psutil==6.1.1
  Downloading psutil-6.1.1-cp37-abi3-win_amd64.whl.metadata (23 kB)
Collecting scikit-learn==1.6.1
  Downloading scikit_learn-1.6.1-cp310-cp310-win_amd64.whl.metadata (15 kB)
Requirement already satisfied: tensorflow==2.18.0 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (2.18.0)
Collecting tqdm==4.67.1
  Downloading tqdm-4.67.1-py3-none-any.whl.metadata (57 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from pandas==2.2.3) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from pandas==2.2.3) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from pandas==2.2.3) (2024.2)
Requirement already satisfied: absl-py in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from keras==3.7.0) (2.1.0)
Requirement already satisfied: rich in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from keras==3.7.0) (13.9.4)
Requirement already satisfied: namex in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from keras==3.7.0) (0.0.8)
Requirement already satisfied: h5py in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from keras==3.7.0) (3.12.1)
Requirement already satisfied: optree in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from keras==3.7.0) (0.13.1)
Requirement already satisfied: ml-dtypes in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from keras==3.7.0) (0.4.1)
Requirement already satisfied: packaging in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from keras==3.7.0) (24.2)
Requirement already satisfied: scipy>=1.6.0 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from scikit-learn==1.6.1) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from scikit-learn==1.6.1) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from scikit-learn==1.6.1) (3.5.0)
Requirement already satisfied: tensorflow-intel==2.18.0 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from tensorflow==2.18.0) (2.18.0)
Requirement already satisfied: colorama in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from tqdm==4.67.1) (0.4.6)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow==2.18.0) (1
.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow==2.18.0)
 (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in c:\users\user\.vscode\env_ml_1113\lib\site-packages (from tensorflow-intel==2.18.0->tens
orflow==2.18.0) (0.6.0)
```

Figure 3.16: The Instruction for Installing the Python Packages.

Figure 3.16 shows the instructions for installing the Python packages. We can install these Python packages by using the command line. The command is: `pip list | findstr /I "numpy pandas keras psutil scikit tensorflow tqdm"`.



```
PS C:\Users\User\.vscode\extensions> pip list | findstr /I "numpy pandas keras psutil scikit tensorflow tqdm"
keras                        3.7.0
numpy                        2.0.2
pandas                       2.2.3
psutil                       6.1.1
scikit-learn                 1.6.1
tensorflow                   2.18.0
tensorflow_intel             2.18.0
tensorflow-io-gcs-filesystem 0.31.0
tqdm                         4.67.1
```

Figure 3.17: The Instruction for verifying the version of Python Packages.

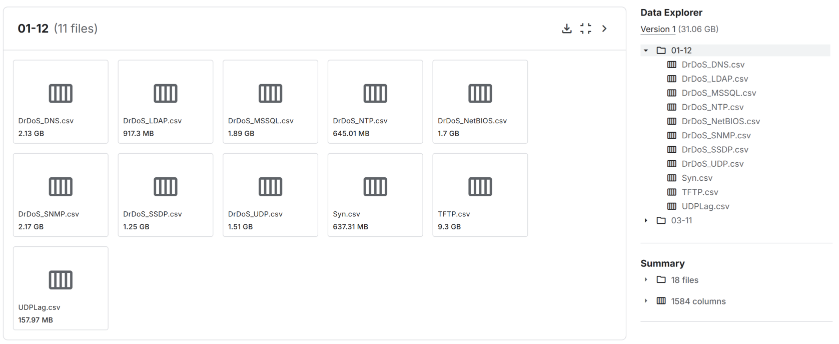
Figure 3.17 shows the instructions for verifying the version of Python packages. We can verify the version by using the command line. The command is: `pip show numpy pandas scikit-learn tensorflow keras tqdm`. After we verify the version, we can use these packages in the VSCode.

### 3.2.3 Implementation Steps

In this part, we will show the implementation steps of our framework. We will separate into two parts: the CIC-DDoS2019 dataset and Adaptive GridSearchCV.

(A) **CIC-DDoS2019 Dataset [38]**

In this part, we will discuss the CIC-DDoS2019 dataset file structure. And we will show the command to download the dataset from the Kaggle website.


**01-12** (11 files)

| | | | | |
|---|---|---|---|---|
| DrDoS_DNS.csv 2.13 GB | DrDoS_LDAP.csv 917.3 MB | DrDoS_MSSQL.csv 1.89 GB | DrDoS_NTP.csv 645.01 MB | DrDoS_NetBIOS.csv 1.7 GB |
| DrDoS_SNMP.csv 2.17 GB | DrDoS_SSDP.csv 1.25 GB | DrDoS_UDP.csv 1.51 GB | Syn.csv 637.31 MB | TFTP.csv 9.3 GB |
| UDPLag.csv 157.97 MB | | | | |

**Data Explorer**

Version 1 (31.06 GB)

- 01-12
  - DrDoS_DNS.csv
  - DrDoS_LDAP.csv
  - DrDoS_MSSQL.csv
  - DrDoS_NTP.csv
  - DrDoS_NetBIOS.csv
  - DrDoS_SNMP.csv
  - DrDoS_SSDP.csv
  - DrDoS_UDP.csv
  - Syn.csv
  - TFTP.csv
  - UDPLag.csv
- 03-11

**Summary**

- 18 files
- 1584 columns

Figure 3.18: CIC-DDoS2019 Dataset File Structure.

Figure 3.18 indicates the file structure of the CIC-DDoS2019 dataset on Kaggle. The dataset is divided into two folders: `01-12` and `03-11`. And we use the `01-12` folder as data for splitting for training and validation sets, and the `03-11` folder as data for the testing set. Each folder contains multiple `.csv` files, which represent the different types of DDoS attacks. And each `.csv` file contains lots of features and one label column. The label column indicates whether the flow is a DDoS attack or benign.

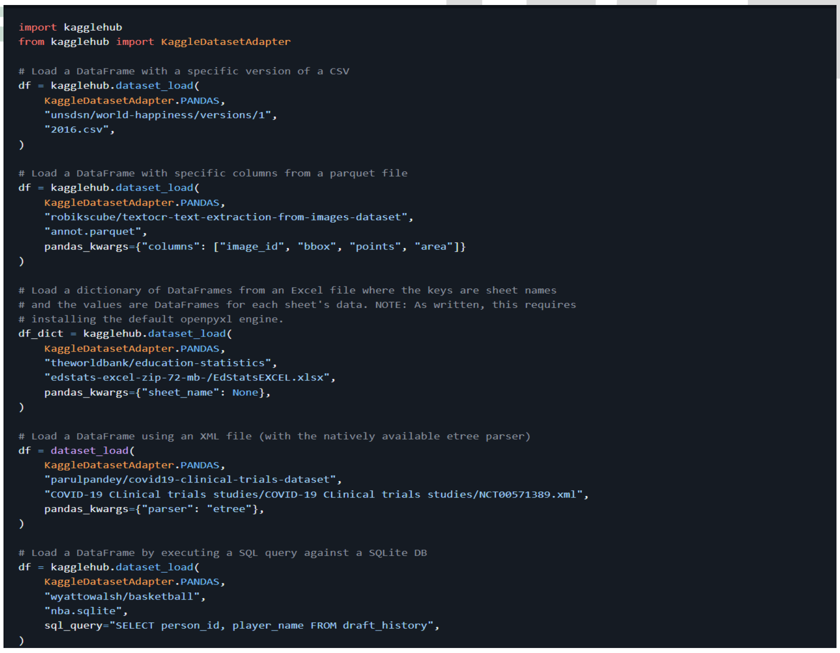

```python
import kagglehub
from kagglehub import KaggleDatasetAdapter

# Load a DataFrame with a specific version of a CSV
df = kagglehub.dataset_load(
    KaggleDatasetAdapter.PANDAS,
    "unsdsn/world-happiness/versions/1",
    "2016.csv",
)

# Load a DataFrame with specific columns from a parquet file
df = kagglehub.dataset_load(
    KaggleDatasetAdapter.PANDAS,
    "robikscube/textocr-text-extraction-from-images-dataset",
    "annot.parquet",
    pandas_kwargs={"columns": ["image_id", "bbox", "points", "area"]}
)

# Load a dictionary of DataFrames from an Excel file where the keys are sheet names
# and the values are DataFrames for each sheet's data. NOTE: As written, this requires
# installing the default openpyxl engine.
df_dict = kagglehub.dataset_load(
    KaggleDatasetAdapter.PANDAS,
    "theworldbank/education-statistics",
    "edstats-excel-zip-72-mb-/EdStatsEXCEL.xlsx",
    pandas_kwargs={"sheet_name": None},
)

# Load a DataFrame using an XML file (with the natively available etree parser)
df = dataset_load(
    KaggleDatasetAdapter.PANDAS,
    "parulpandey/covid19-clinical-trials-dataset",
    "COVID-19 CLinical trials studies/COVID-19 CLinical trials studies/NCT00571389.xml",
    pandas_kwargs={"parser": "etree"},
)

# Load a DataFrame by executing a SQL query against a SQLite DB
df = kagglehub.dataset_load(
    KaggleDatasetAdapter.PANDAS,
    "wyattowalsh/basketball",
    "nba.sqlite",
    sql_query="SELECT person_id, player_name FROM draft_history",
)
```

Figure 3.19: CIC-DDoS2019 Dataset Download from Kaggle.

Figure 3.19 shows the command to download the CIC-DDoS2019 dataset from Kaggle. After we download the dataset, we will get the whole dataset on our local machine. And are ready for the next step of implementation.

(B) **Adaptive GridSearchCV**

Based on the architecture and processes of the proposed framework, we will introduce the main method, which is used for the hyperparameter tuning. This method, called Adaptive GridSearchCV, is one of the core contributions of this study. It is a novel enhancement over traditional grid search strategies, designed to improve both efficiency and accuracy in hyperparameter optimization.

In this subsection, we will discuss the pseudocode of the Adaptive GridSearchCV [see details in Algorithm 1].

In Algorithm 1, it shows the main algorithm of the Adaptive Parameter GridSearchCV. We employ this algorithm to progressively narrow the hyperparameter search space during each iteration of model tuning. The purpose is to increase both efficiency and accuracy in identifying optimal hyperparameter configurations.

At the beginning of each iteration, a shrink factor is defined as $\frac{0.25}{t+1}$, where t is the current iteration number, This factor dynamically controls the construction of the parameter space, allowing broader exploration in early stages and finer resolution in later stages.

During each iteration, the algorithm updates the parameter grid based on the current best parameters. For each hyperparameter, the algorithm distinguishes between categorical (or strin-type) and numerical parameters. If a parameter is categorical or string-type, only the current best value is retained for the next search, ensuring that irrelevant categories are excluded from subsequent iterations.

For numerical parameters, the algorithm first examines the current candidate values. If there is only one value left, it is directly used in the next search. Otherwise, the algorithm locates the position of the current best value within the candidate set. It then calculates the full range of the parameter and detetmines a new interval centered around the best value, with the width of this interval controlled by the shrink factor. This interval is futher discretized into a small set of candidate values (either integer or floating-pint, depending on the parameter type), ensuring that the search remains focused yet sufficiently explorative.

**Algorithm 1:** Adaptive Parameter GridSearch

```
Input: Current best parameters best_params, original parameter grid param_grid,
       current iteration number $t$
Output: Refined parameter grid new_param_grid
$shrink\_factor \leftarrow \frac{0.25}{t+1}$;
$new\_param\_grid \leftarrow \{\}$;
foreach $(param\_name, param\_value) \in best\_params$ do
    if $param\_name$ is categorical or string then
        $new\_param\_grid[param\_name] \leftarrow [param\_value]$;
        continue;
    $current\_values \leftarrow param\_grid[param\_name]$;
    if $|current\_values| \leq 1$ then
        $new\_param\_grid[param\_name] \leftarrow current\_values$;
        continue;
    if $param\_value \in current\_values$ then
        $idx \leftarrow$ index of $param\_value$;
    else
        $idx \leftarrow \arg\min_i |current\_values[i] - param\_value|$;
    $param\_range \leftarrow \max(current\_values) - \min(current\_values)$;
    $delta \leftarrow param\_range \cdot shrink\_factor$;
    if $param\_range < 10^{-10}$ then
        $new\_param\_grid[param\_name] \leftarrow current\_values$;
        continue;
    if $param\_value$ is integer then
        $lower \leftarrow \max(param\_value - \delta, \min(current\_values))$;
        $upper \leftarrow \min(param\_value + \delta, \max(current\_values))$;
        if $lower = upper$ then
            $new\_values \leftarrow [lower]$;
        else
            $step \leftarrow \max(1, \lfloor (upper - lower)/3 \rfloor)$;
            $new\_values \leftarrow$ range from $lower$ to $upper$ with $step$;
    else
        $lower \leftarrow \max(param\_value - \delta, \min(current\_values))$;
        $upper \leftarrow \min(param\_value + \delta, \max(current\_values))$;
        $step \leftarrow (upper - lower)/3$;
        $new\_values \leftarrow \{round(lower + i \cdot step, 6) \mid i = 0, 1, 2, 3\}$;
    $new\_param\_grid[param\_name] \leftarrow sorted(set(new\_values))$;
return $new\_param\_grid$
```

By iteratively updating the parameter grid in this manner, the algorithm adaptively narrows the search space around promising regions. This approach not only reduces computational cost by avoiding redundant evaluations in less promising regions but also increases the likelihood of identifying the true optimal hyperparameter configuration. This method ensures that the search begins broadly to explore the parameter space and progressively becomes more focused around the optimal region.

In summary, the proposed Adaptive GridSearchCV algorithm represents a key innovation in this research. It provides an efficient and systematic way to balance exploration and exploitation during hyperparameter optimization. Its adaptability and progressive refinement make it particularly suitable for high-dimensional search spaces or limited computational budgets. This novel strategy demonstrates our contribution to advancing hyperparameter tuning techniques in machine learning.

# Chapter 4 Result and Discussion

For this chapter, we will offer the results of our experimental study and discuss the performance of the Machine Learning (ML) models. This chapter unfolds in the sequence described below: First, we will introduce the Evaluation indicators used to estimate the outcome of the models. The we will present the outcomes of the research, including the outcome of the models on the training and validation sets, as well as on the testing set (with two different optimized methods). In the end, we will contrast the outcome and efficacy of the models with different optimized methods on the testing set.

## 4.1 Evaluation Metrics [39]

Here, we will indicate the evaluation indicators used to assess the outcome of the ML models. To provide a comprehensive evaluation module, we introduce five fundamental metrics that capture different aspects of model performance: Accuracy, Precision, Recall, F1-score, and confusion metrics. Each metric offers perspective insights into the model's behavior, and together they provide a holistic view of the classification performance. We will first define each metric mathematically, then explain their practical implications and relevance to our specific classification task.

- **Accuracy** is the most straightforward metric, representing the overall correctness of the model's predictions. It expresses the rate of successful predictions compared to the total sample size.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

- **Precision**: This indicator reflects the model's proficiency in locating every authentic case available. It represents the fraction of detected events relative to the complete set of existing phenomena.

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

- **Recall**: This indicator quantifies the detection coverage achieved across relevant data samples. The formula computes actual discoveries divided by the aggregate of genuine occurrences and overlooked cases.

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

- **F1-score**: This parameter consolidates both identification quality and capture efficiency into one representative score.

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4.4}$$

- **Macro Average**: It calculates the unweighted mean of the above indicators across all classes in a multi-class classification problem. Each class contributes equally to the final average, regardless of the class distribution or sample size. This indicator is particularly important when all classes are equally important and you want to estimate the model's outcome on minority classes.

$$\text{Macro Average} = \frac{M_0 + M_1}{2} \tag{4.5}$$

where $M_0$ and $M_1$ are the metric values for class 0 and class 1, respectively.

- **Weighted Average**: It average takes the simple mean of each inidicatorcalculated per class, providing equal weight to all classes independent of their support size. Classes with more samples have greater influence on the final average. This metric reflects the overall classification performance and is closer to the overall accuracy when dealing with imbalanced datasets.

$$\text{Weighted Average} = \frac{M_0 \times S_0 + M_1 \times S_1}{S_0 + S_1} \tag{4.6}$$

where $M_i$ is the metric value for class $i$, and $S_i$ is the support (sample size) for class

$i$.

- **Confusion Matrix**: It is a structured display of classification outcomes that breaks down prediction results by showing the frequency of each type of classification decision.

$$\text{Confusion Matrix} = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix} \tag{4.7}$$

The top-right cell ($FP$) and bottom-left cell ($FN$) of the confusion matrix are particularly critical in our binary classification task. A **False Positive (FP)** indicates that a benign instance is misclassified as an attack, which could lead to unnecessary disruptions such as false alerts or blocking legitimate traffic. A **False Negative (FN)**, on the other hand, represents an attack misclassified as benign. This is especially dangerous in DDoS detection tasks, as it allows malicious traffic to go unnoticed, potentially compromising the system. Therefore, reducing both FP and FN rates is essential, with a particular emphasis on minimizing FN to enhance system security and reliability.

## 4.2 Result and Analysis

In this part, we will present the outcomes of our experiments and analyze the performance of three ML models: SVM, LR, and KNN. For the first step, we will show the performance of the models which optimized by using Adaptive GridSearchCV with training and validation sets, and then we will show the outcome of the models on the testing set. The end of this part will contrast the outcome and efficiency of the models with different optimized methods on the testing set. The results will be presented in figures and tables, and we will provide a detailed analysis of the findings.

### 4.2.1 The Performance on the Training and Validation Sets

Figure 4.1 shows the training and validation accuracy of the SVM model which optimized by using Adaptive GridSearchCV. As illustrated in the training output, the SVM model attains

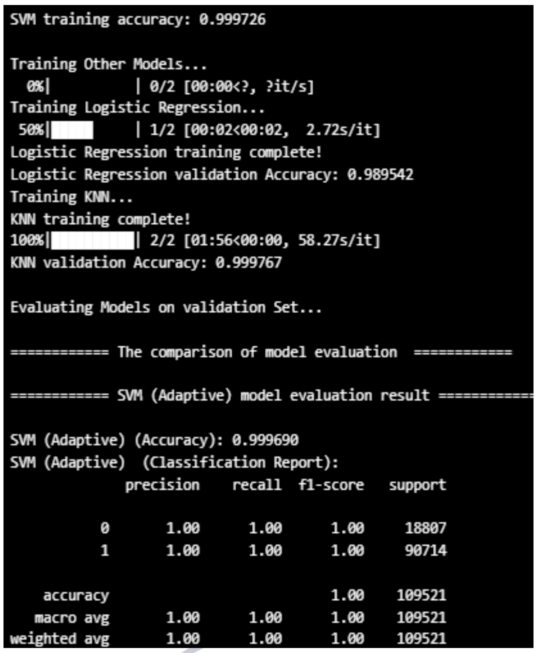

```
SVM training accuracy: 0.999726

Training Other Models...
  0%|          | 0/2 [00:00<?, ?it/s]
Training Logistic Regression...
 50%|█████     | 1/2 [00:02<00:02,  2.72s/it]
Logistic Regression training complete!
Logistic Regression validation Accuracy: 0.989542
Training KNN...
KNN training complete!
100%|██████████| 2/2 [01:56<00:00, 58.27s/it]
KNN validation Accuracy: 0.999767

Evaluating Models on validation Set...

=========== The comparison of model evaluation  ===========

=========== SVM (Adaptive) model evaluation result ===========

SVM (Adaptive) (Accuracy): 0.999690
SVM (Adaptive)  (Classification Report):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     18807
           1       1.00      1.00      1.00     90714

    accuracy                           1.00    109521
   macro avg       1.00      1.00      1.00    109521
weighted avg       1.00      1.00      1.00    109521
```

Figure 4.1: Training and Validation sets on the optimized SVM model.

0.999726 training accuracy while maintaining 0.999767 validation accuracy. The close alignment between these two metrics, with validation accuracy even slightly exceeding training accuracy, is a positive indicator that the model generalizes well to unseen data rather than merely memorizing training examples. This indicates that the it has successfully learned the underlying patterns in the training data without overfitting to noise or specific instances.

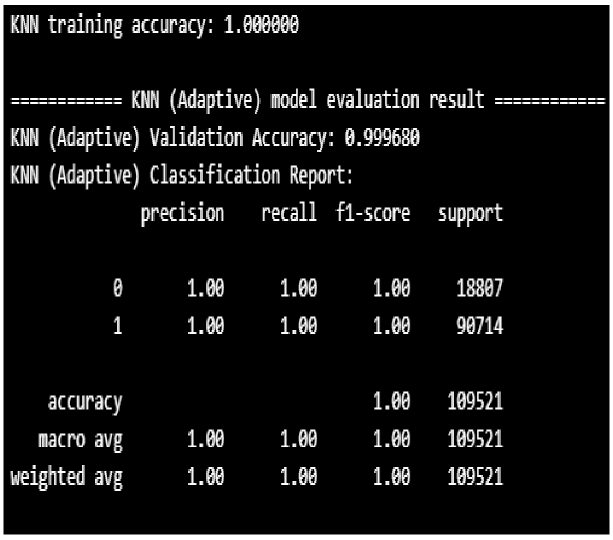
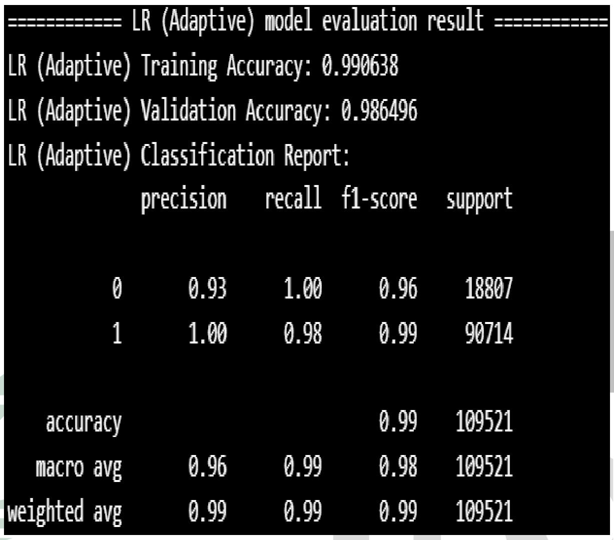

```
KNN training accuracy: 1.000000

=========== KNN (Adaptive) model evaluation result ===========
KNN (Adaptive) Validation Accuracy: 0.999680
KNN (Adaptive) Classification Report:
              precision    recall  f1-score   support


           0       1.00      1.00      1.00     18807
           1       1.00      1.00      1.00     90714


    accuracy                           1.00    109521
   macro avg       1.00      1.00      1.00    109521
weighted avg       1.00      1.00      1.00    109521
```

Figure 4.2: Training and Validation sets on the optimized KNN model.

Figure 4.2 demonstrates the training and validation accuracy of the KNN model which was optimized by using Adaptive GridSearchCV. The optimized KNN model attains 1.0000 training accuracy & 0.999680 validation accuracy, indicating that the model performs exceptionally well on the training data, achieving perfect accuracy. However, the validation accuracy is slightly lower, showing that while the model has learned the pattern very well, it still generalizes effectively to unseen data. The minimal difference between training & validation accuracy (only 0.000320 gap) demonstrates that the model is not overfitting to the training set. The consistency high performance across both training and validation sets confirms the model's robustness and its ability to maintain predictive accuracy in real-world scenarios.



```
=========== LR (Adaptive) model evaluation result ===========
LR (Adaptive) Training Accuracy: 0.990638
LR (Adaptive) Validation Accuracy: 0.986496
LR (Adaptive) Classification Report:
              precision    recall  f1-score   support

           0       0.93      1.00      0.96     18807
           1       1.00      0.98      0.99     90714

    accuracy                           0.99    109521
   macro avg       0.96      0.99      0.98    109521
weighted avg       0.99      0.99      0.99    109521
```

Figure 4.3: Training and Validation sets on the optimized LR model.

As shown in Figure 4.3, the training and validation accuracy of the LR model which was optimized by using Adaptive GridSearchCV. The optimized LR model reaches a 0.990638 training accuracy and 0.986496 validation accuracy. It also demonstrates that the model performs well on the training data, and still generalizes effectively to unseen data like the validation set. The difference between training and validation accuracy is 0.004142, which is relatively small, suggesting that the model is not overfitting to the training data. This shows that the LR model has learned the underlying patterns in the training data and can maintain a good level of predictive accuracy when applied to new, unseen data (such as a testing set).

### 4.2.2 The Performance on the Testing Set

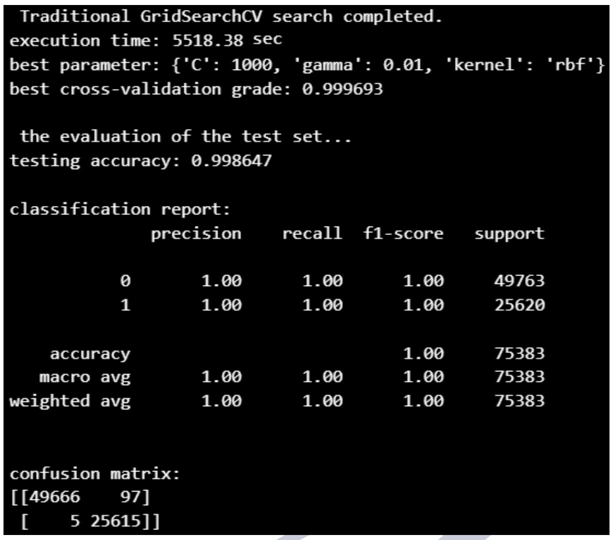
In this subsection, we will split into three parts to show the performance of the models on the testing set. The first part will show the performance of the models which optimized by using Standard GridSearchCV, the second part will show the performance of the models which optimized by using Adaptive GridSearchCV with testing set, and the last part will contrast the outcome & efficiency of the models with different optimized methods on the testing set.

Figure 4.4 illustrates the outcome of the SVM model on the testing set with general GridSearchCV. With the optimized hyperparameters, the classification report shows that the SVM model achieves perfect precision, recall, and F1-score of 1.00 for both classes (0 and 1), the model correctly classifies 49763 instances of class 0 and 25620 instances of class 1. Overall accuracy achieves 0.998647, with marco & weighted averages also reaching perfect scores of 1.00 across all metrics. And the confusion matrix shows [49666,97] and [5,25615], indicating minimal misclassifications, with only 97 false positives & 5 false negatives out of 75383 total instances.

Figure 4.5 shows the outcome of the LR model on the testing set with the general GridSearchCV method. The classification report indicates that the LR model achieves perfect precision (1.00), recall (1.00), and F1-score (1.00) for class 0 with 49763 instances. Class 1 also shows outstanding results with precision of 0.99, recall of 1.00, and F1-score of 1.00 across 25620 instances. Overall accuracy of 0.0996830, with marco & weighted averages also reaching 1.00 across all metrics. The confusion matrix shows [49526,237] and [2,25618], indicating minimal classification errors, with only 237 false positives and 2 false negatives out of 75383 total instances.

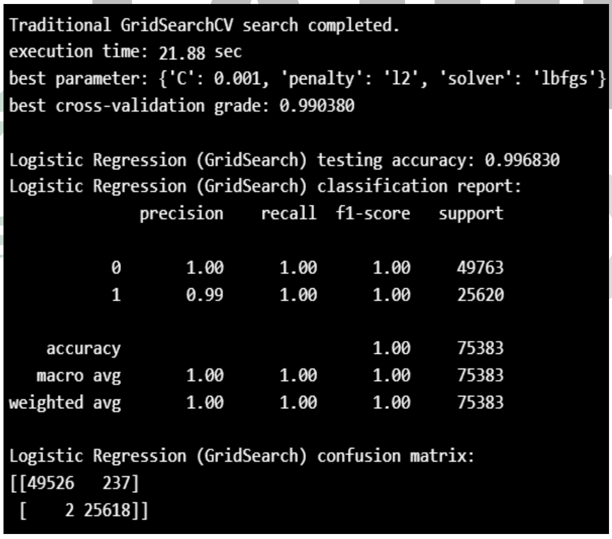
Figure 4.6 presents the outcome of the KNN model on the testing set with general GridSearchCV. The classification report demonstrates exceptional performance across both classes. Both class 0 and class 1 achieve perfect precision (1.00), recall (1.00), and F1-score (1.00), with class 0 having 49763 instances and class 1 having 25620 instances. The overall accuracy of the KNN model is 0.998395, with marco and weighted averages also reaching perfect scores of 1.00 across all metrics. And the confusion matrix shows minimal misclassification with only

119 false positives and 2 false negatives out of 75383 total instances.



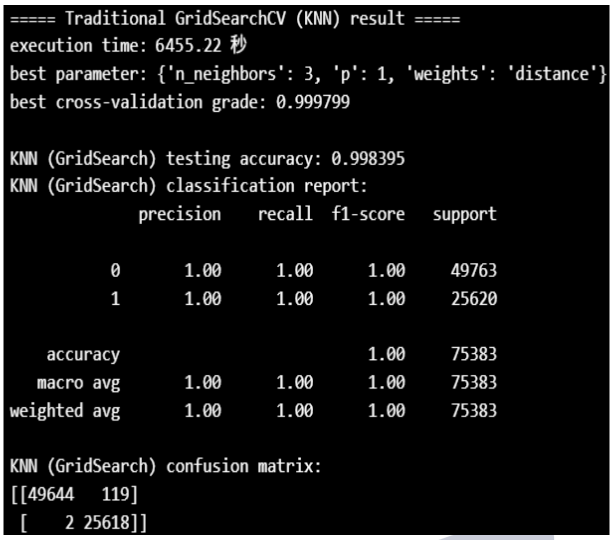
```
 Traditional GridSearchCV search completed.
execution time: 5518.38 sec
best parameter: {'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}
best cross-validation grade: 0.999693

 the evaluation of the test set...
testing accuracy: 0.998647

classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     49763
           1       1.00      1.00      1.00     25620

    accuracy                           1.00     75383
   macro avg       1.00      1.00      1.00     75383
weighted avg       1.00      1.00      1.00     75383


confusion matrix:
[[49666    97]
 [    5 25615]]
```

Figure 4.4: The Performance of SVM model with Standard GridSearchCV on the Testing Set.



```
Traditional GridSearchCV search completed.
execution time: 21.88 sec
best parameter: {'C': 0.001, 'penalty': 'l2', 'solver': 'lbfgs'}
best cross-validation grade: 0.990380

Logistic Regression (GridSearch) testing accuracy: 0.996830
Logistic Regression (GridSearch) classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     49763
           1       0.99      1.00      1.00     25620

    accuracy                           1.00     75383
   macro avg       1.00      1.00      1.00     75383
weighted avg       1.00      1.00      1.00     75383

Logistic Regression (GridSearch) confusion matrix:
[[49526   237]
 [    2 25618]]
```

Figure 4.5: The Performance of LR model with Standard GridSearchCV on the Testing Set.



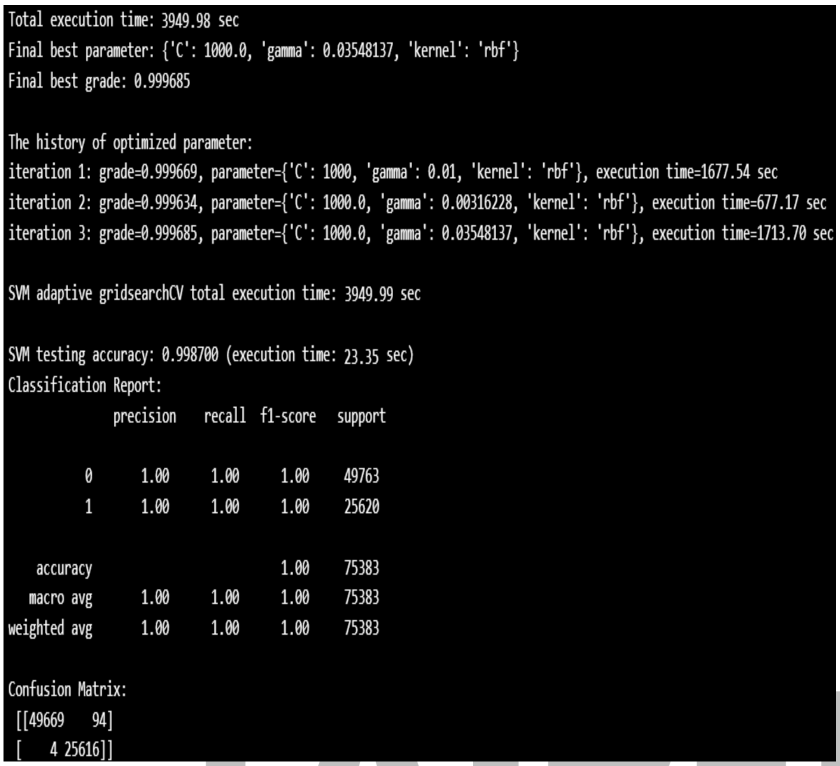
```
===== Traditional GridSearchCV (KNN) result =====
execution time: 6455.22 秒
best parameter: {'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
best cross-validation grade: 0.999799

KNN (GridSearch) testing accuracy: 0.998395
KNN (GridSearch) classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     49763
           1       1.00      1.00      1.00     25620

    accuracy                           1.00     75383
   macro avg       1.00      1.00      1.00     75383
weighted avg       1.00      1.00      1.00     75383

KNN (GridSearch) confusion matrix:
[[49644   119]
 [    2 25618]]
```

Figure 4.6: The Performance of KNN model with Standard GridSearchCV on the Testing Set.

Figure 4.7 indicates the outcome of the SVM model on the testing set with Adaptive GridSearchCV. With the classification report, the SVM model achieves a precision, recall and F1-score of 1.00 for both classes (0 and 1). And the macro & weighted averages also reach scores of 1.00 across all metrics. The overall accuracy of the SVM model is 0.998700, with the confusion matrix showing [49669,94] and [4,25616], demonstrating with only 94 false positives and 4 false negatives out of 49763 class 0 instances and 25620 class 1 instances (total 75383 instances).

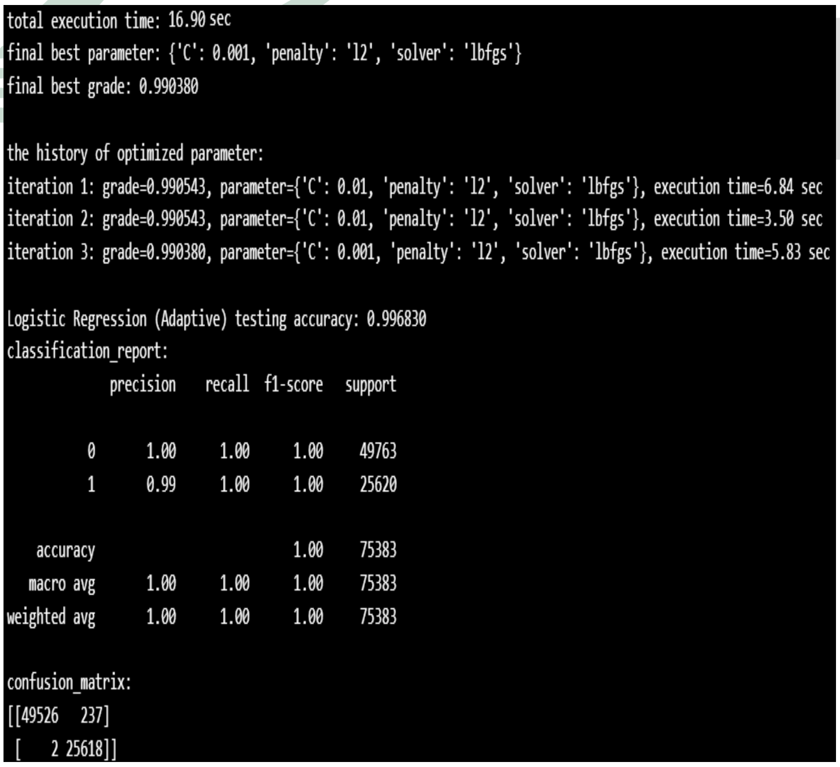
Figure 4.8 shows the outcome of the LR model on the testing set with Adaptive GridSearchCV. The classification report indicates that the LR model attains a precision, recall, and F1-score of 1.00 for class 0 , a precision of 0.99, recall od 1.00, and F1-score of 1.00 for class 1. And the overall testing accuracy is 0.996830, with macro and weighted averages also reaching scores of 1.00 across all metrics. The confusion matrix shows [49526, 237] and [2,25618], indicating with only 237 false positives and 2 false negatives out of 75383 total instances.

Figure 4.9 demonstrates the outcome of the KNN model on the testing set with Adaptive GridSearchCV. THE classification report shows that the KNN model achieves a precision, recall, and F1-score of 1.00 for both classes (0 and 1). The overall testing accuracy is 0.998395, with macro and weighted averages also attaining scores of 1.00 across all metrics. The confusion matrix shows [49644, 119] and [2, 25618], indicating with only 119 false positives and 2 false negatives out of 75383 total instances.



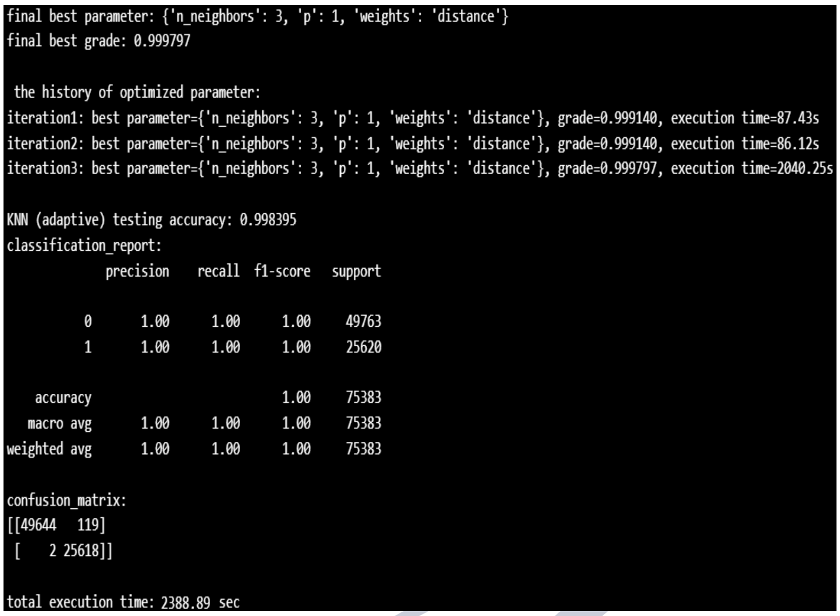
```
Total execution time: 3949.98 sec
Final best parameter: {'C': 1000.0, 'gamma': 0.03548137, 'kernel': 'rbf'}
Final best grade: 0.999685

The history of optimized parameter:
iteration 1: grade=0.999669, parameter={'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}, execution time=1677.54 sec
iteration 2: grade=0.999634, parameter={'C': 1000.0, 'gamma': 0.00316228, 'kernel': 'rbf'}, execution time=677.17 sec
iteration 3: grade=0.999685, parameter={'C': 1000.0, 'gamma': 0.03548137, 'kernel': 'rbf'}, execution time=1713.70 sec

SVM adaptive gridsearchCV total execution time: 3949.99 sec

SVM testing accuracy: 0.998700 (execution time: 23.35 sec)
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     49763
           1       1.00      1.00      1.00     25620

    accuracy                           1.00     75383
   macro avg       1.00      1.00      1.00     75383
weighted avg       1.00      1.00      1.00     75383

Confusion Matrix:
 [[49669    94]
 [    4 25616]]
```

Figure 4.7: The Performance of SVM model with Adaptive GridSearchCV on the Testing Set.



```
total execution time: 16.90 sec
final best parameter: {'C': 0.001, 'penalty': 'l2', 'solver': 'lbfgs'}
final best grade: 0.990380

the history of optimized parameter:
iteration 1: grade=0.990543, parameter={'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}, execution time=6.84 sec
iteration 2: grade=0.990543, parameter={'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}, execution time=3.50 sec
iteration 3: grade=0.990380, parameter={'C': 0.001, 'penalty': 'l2', 'solver': 'lbfgs'}, execution time=5.83 sec

Logistic Regression (Adaptive) testing accuracy: 0.996830
classification_report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     49763
           1       0.99      1.00      1.00     25620

    accuracy                           1.00     75383
   macro avg       1.00      1.00      1.00     75383
weighted avg       1.00      1.00      1.00     75383

confusion_matrix:
[[49526   237]
 [    2 25618]]
```

Figure 4.8: The Performance of LR model with Adaptive GridSearchCV on the Testing Set.



```
final best parameter: {'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
final best grade: 0.999797

 the history of optimized parameter:
iteration1: best parameter={'n_neighbors': 3, 'p': 1, 'weights': 'distance'}, grade=0.999140, execution time=87.43s
iteration2: best parameter={'n_neighbors': 3, 'p': 1, 'weights': 'distance'}, grade=0.999140, execution time=86.12s
iteration3: best parameter={'n_neighbors': 3, 'p': 1, 'weights': 'distance'}, grade=0.999797, execution time=2040.25s

KNN (adaptive) testing accuracy: 0.998395
classification_report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     49763
           1       1.00      1.00      1.00     25620

    accuracy                           1.00     75383
   macro avg       1.00      1.00      1.00     75383
weighted avg       1.00      1.00      1.00     75383

confusion_matrix:
[[49644   119]
 [    2 25618]]

total execution time: 2388.89 sec
```

Figure 4.9: The Performance of KNN model with Adaptive GridSearchCV on the Testing Set.

In Figure 4.10, 4.11, and 4.12, they show the performance comparison of SVM, LR, and KNN models with different optimized methods on the testing set. The outcome is estimated using execution time and testing accuracy. The execution time is measured in seconds, and the testing accuracy is represented as a decimal value between 0 and 1, where 1 means perfect accuracy. And the models are optimized by using two different methods: one is the Adaptive GridSearchCV, and the other is the Standard GridSearhcCV. Adaptive GridSearchCV is the method we proposed, which is designed to adaptively select the best hyperparameters for the models by using a more efficient search strategy. Standard GridSearchCV is the traditional method that exhaustively searches through a predefined set of hyperparameters to find the best combination. We compare those two methods (the third one is the GridSearchCV from other paper) to estimate the performance of the models on the testing set. The results demonstrate that: another with the LR and KNN models, the method we proposed (Adaptive GridSearchCV) achieves the same testing accuracy as the Standard GridSearchCV, but with apparently less execution time. This indicates that it is more efficient than the traditional one. However, with the SVM model, it further improves the testing accuracy while decreasing lots of execution time. At the same time, we also compare the performance of the models with the GridSearchCV from another paper (but without execution time) [40], which is a method that is used in another paper to optimize the models. The result shows that our method attains a apparently higher testing accuracy than the

GridSearchCV from another paper, which indicaates that our method is more effective than the traditional one.

From Tables 4.1, 4.2, and 4.3, they show the comparison of performance of SVM, LR, and KNN models optimized by General GridSearchCV and Adaptive GridSearchCV. The results show three different models with two different optimized methods. The first one is the Adaptive GridSearchCV, which is the method we proposed, the second one is the General GridSearchCV, which is the traditional method that exhaustively searches through a predefined set of hyperparameters to find the best combination, and the third one is the General GridsearchCV from another paper (but without execution time).

In table 4.1, it illustrates that the SVM model optimized by our method achieves a testing accuracy of 0.998700 with an execution time of 3949.99 seconds. The General GridSearchCV attains 0.998647 testing accuracy with an execution time of 5518.38 seconds. The General GridSearchCV from other paper attains a testing accuracy of 0.95 (without execution time). This indicates that our method decreases around 1568.39 seconds of execution time (about 28% time decrease), and it further improves the testing accuracy by 0.000053 (extremely close to testing accuracy of 1).

Table 4.2 indicates the camparision of performance of LR model optimized by general GridSearchCV and Adaptive GridSearchCV. The result shows that the LR model optimized by our method attains 0.996830 testing accuracy with an execution time of 16.90 seconds, while the General GridSearchCV achieves the same testing accuracy of 0.996830 with an execution time of 21.88 secoonds, and the General GridSearchCV from other paper (without execution time) reaches a testing accuracy of 0.94. From the result, we can see that our method we apparently reduces the execution time by 4.98 seconds (about 22.8% time reduction) while maintaining the same testing accuracy as the general method. And compared to the general method from another paper, our method evidently has higher testing accuracy by 0.056830 (about 6% improvement). It suggests that our method keeps the same testing accuracy and is even better than another paper while reducing the execution time.

In table 4.3, it shows that the KNN model optimized by our method achieves a testing accuracy of 0.998395 with an execution time of 2388.89 seconds, while the general GridSearchCV achieves a testing accuracy of 0.998395 with an execution time of 6455.22 seconds, and the general one from other papers (without execution time) attains a testing accuracy of 0.9461. Based on the result, we can see that our method strongly reduces the execution time by 4066.33 seconds (about 63% time reduction) while maintaining the same testing accuracy. And compared to the general method from other paper, our method reaches a higher testing accuracy by 0.052295 (about 5.5% improvement). This indicates that our method is more efficient than the general method while maintaining a good level of testing accuracy.

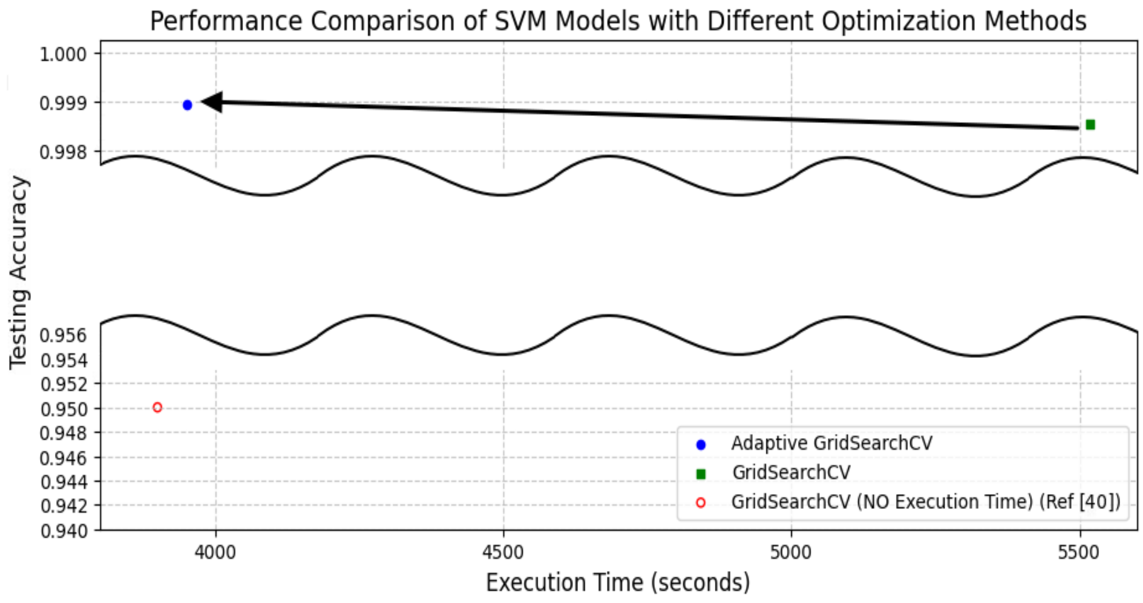


Figure 4.10: The Performance comparison of SVM models with Different Optimized Methods on the Testing Set.

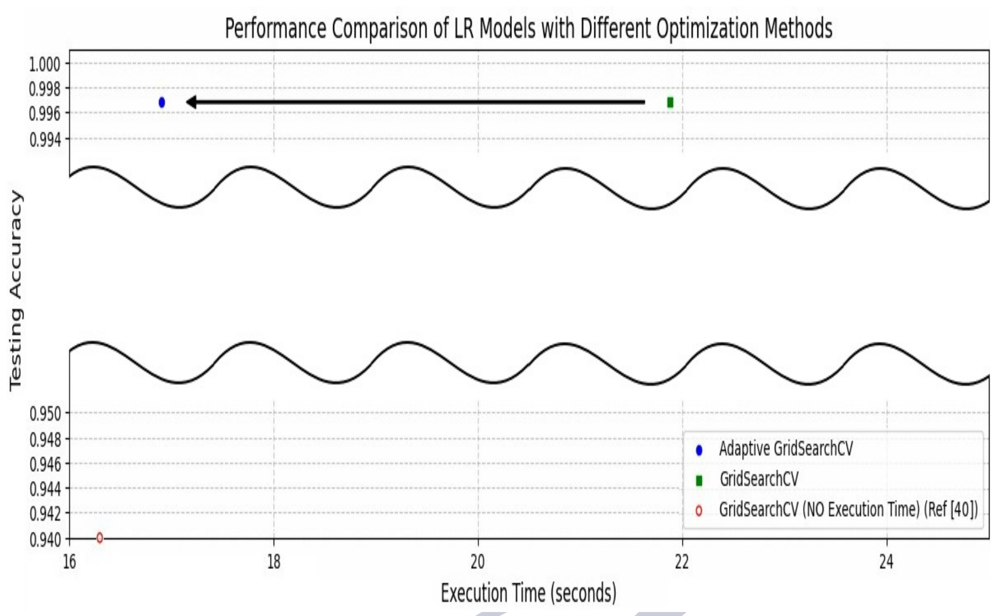


Figure 4.11: The Performance comparison of LR models with Different Optimized Methods on the Testing Set.

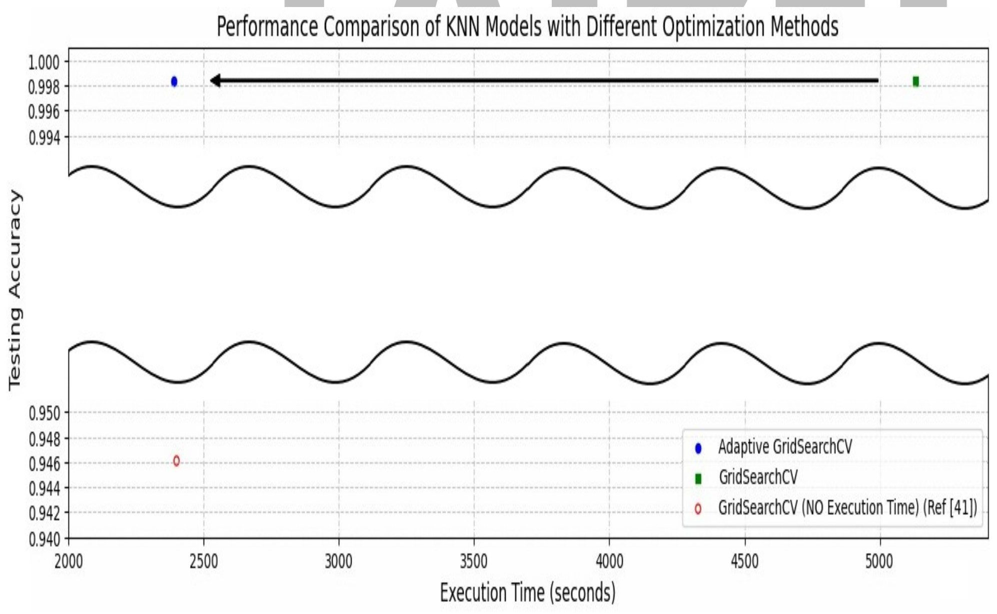


Figure 4.12: The Performance comparison of KNN models with Different Optimized Methods on the Testing Set.

Table 4.1: The Comparision of Performance of SVM Model Optimized by General GridSearchCV and Adaptive GridSearchCV

| **Optimized method** | **Execution Time** | **Testing Accuracy** |
|---|---|---|
| SVM(Adaptive GridSearchCV) | 3949.99sec | 0.998700 |
| SVM (GridSearchCV) | 5518.38sec | 0.998647 |
| SVM(GridSearchCV from other paper) [40] | X | 0.95 |

Table 4.2: The Comparision of Performance of LR Model Optimized by General GridSearchCV and Adaptive GridSearchCV

| **Optimized method** | **Execution Time** | **Testing Accuracy** |
|---|---|---|
| LR (Adaptive GridSearchCV) | 16.90sec | 0.996830 |
| LR (GridSearchCV) | 21.88sec | 0.996830 |
| LR (GridSearchCV from other paper) [40] | X | 0.94 |

Table 4.3: The Comparision of Performance of KNN Model Optimized by General GridSearchCV and Adaptive GridSearchCV

| **Optimized method** | **Execution Time** | **Testing Accuracy** |
|---|---|---|
| KNN (Adaptive GridSearchCV) | 2388.89sec | 0.998395 |
| KNN (GridSearchCV) | 6455.22sec | 0.998395 |
| KNN (GridSearchCV from other paper) [41] | X | 0.9461 |

# Chapter 5 Conclusion and Future Work

For this chapter we will summarize the main findings of this thesis and discuss the potential future work that can be fulfilled to further enhance the detection & optimization of ML models performance & DDoS attacks.

## 5.1 Conclusion

In response to the growing sophistication of DDoS attacks and the urgent need for enhance detection mechanisms, this research demonstrated a method to strengthen the capability of ML models in detecting DDoS attacks through hyperparameter tuning. While conventional grid searchCV method requires exhaustive exploration of all parameters combinations, this study introduces an adaptive grid searchCV that strategically narrows the search space by first identifying potential regions and then refining the parameter range iteratively. This approach not only improves or maintains accuracy levels but also with substantial time savings.

The empirical findings using CIC-DDoS2019 dataset validate the effectiveness of the proposed adaptive grid searchCV. The SVM model attained 99.87% testing accuracy with approximately 28% reduction in execution time, while the LR model maintained 99.68% accuracy with a 22.8% time improvement. Most notably, the KNN model demonstrated the same high accuracy (99.8395%) while achieving a remarkable 63% reduction in computational time, clearly demonstrating that the adaptive strategy can enhance both performance and efficiency simultaneously.

## 5.2 Future work

An important avenue for future work involves expanding the evaluation scope to encompass a broader range of DDoS attack datasets with varying characteristics. Testing the adaptive grid searchCV on datasets that capture different attack vectors, network topologies, and temporal patterns would strenthgen confidence in its universal applicability. This would also reveal whether the adaptive search parameters require adjustment based on specific dataset properties, potentially leading to dataset-ware optimization strategies.

Building upon the cross-dataset validation, another critical resarch direction involves developing model-specific adaptive search strategies that account for the unique characteristics of different ML models. Rather than applying a uniform adaptive approach, future work could investigate customized search mechanisms for the specific parameter sensitivity patterns and optimization landscapes of individual models such as neural networks or ensemble methods.

# References

[1] I. H. Sarker, A. S. M. Kayes, S. Badsha, H. Alqahtani, P. Watters, and A. Ng, "Cybersecurity data science: An overview from machine learning perspective," *Journal of Big Data*, vol. 7, pp. 1–29, 2020.

[2] A. Abhishta, W. van Heeswijk, M. Junger, L. J. M. Nieuwenhuis, and R. Joosten, "Why would we get attacked? an analysis of attacker's aims behind ddos attacks," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 11, no. 2, pp. 3–22, 2020.

[3] M. Abdullahi et al., "Detecting cybersecurity attacks in internet of things using artificial intelligence methods: A systematic literature review," *Electronics*, vol. 11, no. 2, p. 198, 2022.

[4] M. M. Mijwil, I. E. Salem, and M. M. Ismaeel, "The significance of machine learning and deep learning techniques in cybersecurity: A comprehensive review," *Iraqi Journal for Computer Science and Mathematics*, vol. 4, no. 1, p. 10, 2023.

[5] S. Peneti and E. Hemalatha, "Ddos attack identification using machine learning techniques," in *2021 International Conference on Computer Communication and Informatics (ICCCI)*, 2021, pp. 1–5.

[6] A. M. Al-Eryani, E. Hossny, and F. A. Omara, "Efficient machine learning algorithms for ddos attack detection," in *2024 6th International Conference on Computing and Informatics (ICCI)*, 2024, pp. 174–181.

[7] B. A. Khalaf, S. A. Mostafa, A. Mustapha, M. A. Mohammed, and W. M. Abduallah, "Comprehensive review of artificial intelligence and statistical approaches in distributed denial of service attack and defense methods," *IEEE Access*, vol. 7, pp. 51 691–51 713, 2019.

[8] J. Frank, "Artificial intelligence and intrusion detection: Current and future directions," in *Proceedings of the 17th national computer security conference*, sBaltimore, MD, vol. 10, 1994, pp. 1–12.

[9] A. Suhag and A Daniel, "Study of statistical techniques and artificial intelligence methods in distributed denial of service (ddos) assault and defense," *Journal of Cyber Security Technology*, vol. 7, no. 1, pp. 21–51, 2023.

[10] A. Jaszcz and D. Połap, "Aimm: Artificial intelligence merged methods for flood ddos attacks detection," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 10, pp. 8090–8101, 2022.

[11] N. Mohamed, “Ddos attacks mitigation: A review of ai-based strategies and techniques,” in *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, IEEE, 2024, pp. 1–6.

[12] P. S. Saini, S. Behal, and S. Bhatia, “Detection of ddos attacks using machine learning algorithms,” in *2020 7th International Conference on Computing for Sustainable Global Development (INDIACom)*, IEEE, 2020, pp. 16–21.

[13] R. Santos, D. Souza, W. Santo, A. Ribeiro, and E. Moreno, “Machine learning algorithms to detect ddos attacks in sdn,” *Concurrency and Computation: Practice and Experience*, vol. 32, no. 16, e5402, 2020.

[14] M. A. Al-Shareeda, S. Manickam, and M. A. Saare, “Ddos attacks detection using machine learning and deep learning techniques: Analysis and comparison,” *Bulletin of Electrical Engineering and Informatics*, vol. 12, no. 2, pp. 930–939, 2023.

[15] M. Alduailij, Q. W. Khan, M. Tahir, M. Sardaraz, M. Alduailij, and F. Malik, “Machine-learning-based ddos attack detection using mutual information and random forest feature importance method,” *Symmetry*, vol. 14, no. 6, p. 1095, 2022.

[16] M. Marvi, A. Arfeen, and R. Uddin, “A generalized machine learning-based model for the detection of ddos attacks,” *International Journal of Network Management*, vol. 31, no. 6, e2152, 2021.

[17] J. Dalvi, V. Sharma, R. Shetty, and S. Kulkarni, “Ddos attack detection using artificial neural network,” in *2021 International Conference on Industrial Electronics Research and Applications (ICIERA)*, IEEE, 2021, pp. 1–5.

[18] R Latha and S. J. J. Thangaraj, “Machine learning approaches for ddos attack detection: Naive bayes vs logistic regression,” in *2023 Second International Conference On Smart Technologies For Smart Nation (SmartTechCon)*, IEEE, 2023, pp. 1043–1048.

[19] A. Aljuhani, “Machine learning approaches for combating distributed denial of service attacks in modern networking environments,” *IEEE Access*, vol. 9, pp. 42 236–42 264, 2021.

[20] F. B. Saghezchi, G. Mantas, M. A. Violas, A. M. de Oliveira Duarte, and J. Rodriguez, “Machine learning for ddos attack detection in industry 4.0 cppss,” *Electronics*, vol. 11, no. 4, p. 602, 2022.

[21] A. E. Cil, K. Yildiz, and A. Buldu, “Detection of ddos attacks with feed forward based deep neural network model,” *Expert Systems with Applications*, vol. 169, p. 114 520, 2021.

[22] O. R. Sanchez, M. Repetto, A. Carrega, and R. Bolla, “Evaluating ml-based ddos detection with grid search hyperparameter optimization,” in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, IEEE, 2021, pp. 402–408.

[23] H. Nisya, S. N. Hertiana, and Y. Purwanto, “Implementation of hyperparameter tuning random forest algorithm in machine learning for sdn security: An innovative exploration of ddos attack detection,” in *2024 International Conference on Artificial Intelligence, Blockchain, Cloud Computing, and Data Analytics (ICoABCD)*, IEEE, 2024, pp. 321–326.

[24] S. Dasari and R. Kaluri, “An effective classification of ddos attacks in a distributed network by adopting hierarchical machine learning and hyperparameters optimization techniques,” *IEEE Access*, vol. 12, pp. 10 834–10 845, 2024.

[25] T. M. May, Z. Zainudin, N. Muslim, N. S. Jamil, N. A. M. Jan, N. Ibrahim, et al., “Intrusion detection system (ids) classifications using hyperparameter tuning for machine learning and deep learning,” in *2024 5th International Conference on Artificial Intelligence and Data Sciences (AiDAS)*, IEEE, 2024, pp. 344–349.

[26] A. A. Alashhab et al., “Enhancing ddos attack detection and mitigation in sdn using an ensemble online machine learning model,” *IEEE Access*, 2024.

[27] M. Gniewkowski, “An overview of dos and ddos attack detection techniques,” in *International Conference on Dependability and Complex Systems*, Springer, 2020, pp. 233–241.

[28] University of New Brunswick Cyber Defence Lab. “CIC-DDoS2019 Dataset,” Accessed: Dec. 14, 2024. [Online]. Available: `https://www.unb.ca/cic/datasets/ddos-2019.html`.

[29] Microsoft. “Visual studio code.” Accessed: 2025-05-13. [Online]. Available: `https://code.visualstudio.com/`.

[30] Project Jupyter. “Project jupyter.” Accessed: 2025-05-13. [Online]. Available: `https://jupyter.org/`.

[31] Python Software Foundation. “Python 3.12.0.” Accessed: 2025-05-19. [Online]. Available: `https://www.python.org/downloads/release/python-3120/`.

[32] NumPy Developers. “Numpy.” Accessed: 2025-05-19. [Online]. Available: `https://numpy.org/`.

[33] pandas development team. “Pandas - python data analysis library.” Accessed: 2025-05-19. [Online]. Available: `https://pandas.pydata.org/`.

[34] Keras Team. “Keras: The python deep learning api.” Accessed: 2025-05-19. [Online]. Available: `https://keras.io/getting_started/`.

[35] Giampaolo Rodola. “Psutil: Cross-platform lib for process and system monitoring in python.” Accessed: 2025-05-19. [Online]. Available: `https://pypi.org/project/psutil/`.

[36] scikit-learn developers. “Scikit-learn: Machine learning in python.” Accessed: 2025-05-19. [Online]. Available: https://scikit-learn.org/stable/.

[37] tqdm developers. “Tqdm: A fast, extensible progress bar for python and cli.” Accessed: 2025-05-19. [Online]. Available: https://tqdm.github.io/.

[38] R. R. Silva. “Cic-ddos2019 30gb (full dataset csv files),” Accessed: Apr. 13, 2025. [Online]. Available: https://www.kaggle.com/datasets/rodrigorosasilva/cic-ddos2019-30gb-full-dataset-csv-files.

[39] R. Dey. “A guide to key evaluation metrics for machine learning models,” Accessed: May 13, 2025. [Online]. Available: https://medium.com/@roshmitadey/a-guide-to-key-evaluation-metrics-for-machine-learning-models-188d589a347b.

[40] M. A. I. Sabbir, “Enhanced detection and classification of DDoS attacks using optimized hybrid machine learning models,” 2025, pp. 1–10.

[41] T. T. Khoei, S. Ismail, and N. Kaabouch, “Boosting-based models with tree-structured parzen estimator optimization to detect intrusion attacks on smart grid,” in *2021 IEEE 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, IEEE, 2021, pp. 0165–0170.