SERBIAN JOURNAL OF ELECTRICAL ENGINEERING
Vol. 22, No. 2, June 2025, 183-199
UDC: 004.85:656.01 **DOI: https://doi.org/10.2298/SJEE2502183S**
Original scientific paper

Developing Tunable Machine Learning Workflow for Traffic Analysis in SDN

Sama Salam Samaan[1], Hassan Awheed Jeiad[1]

Abstract: Traffic monitoring is a critical issue in networking in general, especially in SDN due to its layered architecture in which the control plane represents a single point of failure. Therefore, this paper is tailored to mitigate the control and mitigate the effect od the DDoS attacks in SDN networks. It presents a complete machine learning (ML) workflow that begins with data ingestion and end with a trained model that is capable of analyzing packets in a production network. Three ML pipelines are part of this workflow, where the training process is carried out on a distributed framework, i.e., Spark, to accomplish a near real time analysis for each flow of packets. To evaluate the performance of the suggested workflow, the LR-HR DDoS 2024 dataset is employed. The decision tree model outperforms the remaining models with 99% of accuracy and 4 min 33 s of training time.

Keywords: Machine learning pipeline, Machine learning workflow, SDN, Spark, Traffic analysis.

1 Introduction

The SDN architecture offers a unique opportunity to leverage big data analytics and tools like Spark for network traffic analysis and management, enabled by its advanced features, including the separation of the control plane from the data plane, a global network view, and network programmability [1, 2]. The recent advancements in high-speed data processing have opened up countless opportunities for the scientific and industrial sectors to delve into new domains. Big data analytics and SDN are two pivotal technologies that have played a major role in these remarkable accomplishments.

While SDN offers a suitable control platform for traffic analysis, several technical challenges need to be addressed before it can be fully implemented [3]. To ensure efficient service performance and optimal resource utilization, the

[1]Computer Engineering Department, University of Technology, Iraq,
sama.s.samaan@uotechnology.edu.iq, https://orcid.org/0000-0002-6805-1483;
hassan.a.jeiad@uotechnology.edu.iq, https://orcid.org/0000-0001-5560-3157

SDN controller must meticulously analyze the vast volumes of data gathered on various network states and traffic loads [4]. In dynamic, large-scale SDN environments characterized by diverse network states and fluctuating traffic patterns, advanced data analysis and decision-making capabilities are essential–capabilities that conventional network traffic analysis methods struggle to provide. Additionally, the centralized design of SDN networks raises critical security concerns, as the SDN controller can become a single point of failure. Moreover, the current SDN standards do not enforce security protections due to implementation complexities, which could hinder SDN's widespread adoption over time. Consequently, these vulnerabilities are attracting increased attention from malicious attackers, making security threats a significant concern.

This work is motivated by the desire to advance the field of SDN by incorporating big data analytics and tools to enhance network analysis capabilities. The motivation stems from optimizing SDN networks and improving their performance, scalability, and efficiency. Scalability and flexibility are crucial factors in addressing the dynamic nature of SDN networks. Leveraging Spark to build ML pipelines enables efficient handling of large-scale networks and adaptation to evolving network conditions.

This work introduces an architecture for ML pipelines tailored for traffic analysis in SDN. Three ML pipelines are designed and persisted to support ML workflow operations, representing a novel approach to addressing traffic analysis challenges in SDN networks. Furthermore, a configurable ML workflow is proposed, comprising seven stages: Data Ingestion, Exploratory Data Analysis (EDA), Data Cleaning, Data Segregation, Machine Learning Pipeline Training, Model Evaluation, and Model Deployment.

The workflow utilizes several Spark modules, including Spark SQL, Spark Streaming, and Spark ML. Spark SQL supports the EDA and Data Cleaning stages, Spark Streaming facilitates stream processing, and Spark ML is instrumental in constructing the proposed ML pipelines.

The key contributions of this research are threefold. First, proposing a new ML pipeline architecture consisting of feature engineering and model training stages. Second, the architected pipelines are utilized within a tunable ML workflow to construct efficient and reliable models to detect DDoS attacks in SDN networks. Third, the ML workflow is evaluated using the "LR-HR DDoS 2024" dataset.

2 Related Work

This part of the work outlines the relevant studies that utilized ML and deep learning in traffic analysis in SDN, specifically, DDoS detection.

C. Li et al. [5] utilized deep learning technique to build a DDoS detection model and defence system in an SDN environment. The model is able to identify

patterns in network traffic and detect historical network attack activities. The proposed system efficiently reduces DDoS attack traffic in SDN by minimizing dependency on the environment, streamlining real-time updates to the detection system, and facilitating easier upgrades or modifications to the detection strategy. Experimental validation using the ISCX dataset demonstrates the system's effectiveness, achieving high detection accuracy. The study in [6] uses XGBoost for DDoS detection in SDN-based clouds, utilizing POX controller and Mininet to simulate attacks. It presented XGBoost dominance in accuracy, speed, and scalability compared to other classifiers, addressing vital cloud security considerations by efficiently identifying DDoS patterns. M. Oo et al. [7] introduced an SDN-based approach for detecting DDoS attacks that minimizes disruptions to legitimate user activities. They also enhanced the Support Vector Machine (SVM) algorithm for improved DDoS attack detection. Their Advanced SVM (ASVM) technique employs a three-class classification method to effectively identify two types of flooding-based DDoS attacks. The method's performance was assessed using metrics such as false alarm rate, detection rate, and accuracy. Results revealed a detection accuracy of approximately 97%, with minimal training and testing times.

R. Swami et al. [8] explored the impact of spoofed and non-spoofed TCP-SYN flooding attacks on SDN controller resources. They created an intrusion detection system employing machine learning and evaluated five classification models from different algorithm families for traffic classification, using cross-validation for validation. This approach enabled the extraction of more effective features and accurate traffic classification. Experimental results demonstrated that all the evaluated classification models performed satisfactorily.

A. Singh et al. [9] presents SDN as a transformative approach for network analysis, highlighting its vulnerability to DDoS attacks. The paper’s major contributions include the creation of a large DDoS dataset, achieving 99.1% accuracy in DDoS detection using both Snort and ML with various algorithms. Two methods were employed for mitigation, dropping suspicious traffic and redirecting it to a different path. The paper also emphasizes the need for redundancy in network design, the unavailability of public SDN DDoS datasets, their plan to share their dataset for future research, and the significance of classifying normal and DDoS traffic in mitigation.

Researchers continuously explore methods to detect DDoS attacks with improved efficiency and speed. The previous studies provide valuable knowledge about creating intelligent networks. However, based on our search, none of the existing studies utilized Spark for SDN traffic analysis, specifically DDoS detection. Spark is recognized as a robust big data processing framework that can construct ML models and train them on vast amounts of data faster than currently available tools.

3 Machine Learning Pipeline

A ML pipeline is a series of interconnected stages to automate developing and deploying ML models. It streamlines designing, fine-tuning, and assessing the performance of a ML workflow [10]. It comprises a series of interconnected stages designed to automate the workflow [11]. The pipeline stages are executed sequentially, transforming the input DataFrame at each stage. In a ML pipeline, the output of one stage is used as the input to the next stage, where each stage can be categorized as either an estimator or a transformer. An estimator is an algorithm applied to a DataFrame to generate a transformer, while a transformer is an algorithm that modifies a DataFrame by removing, adding, or updating existing features.

The ML pipeline provides several benefits, including parallel processing, code simplification, error reduction, time savings, and enhanced model performance [12]. Apache Spark is a popular big data processing framework with ML libraries such as MLlib [13]. It offers an efficient way to build ML pipelines at scale.

3.1 Machine learning pipelines construction and persistence

The proposed ML pipeline consists of five sequential stages: imputer, vector assembler, feature selection, standard scaler, and ML algorithm. The initial four stages encompass the data preprocessing and feature engineering stages. In the final stage, a specific ML algorithm is used to construct a specific ML pipeline. Therefore, three ML pipelines are constructed: decision tree (DT) pipeline, random forest (RF) pipeline, and logistic regression (LR) pipeline. These pipelines are persisted and loaded whenever training a ML model is needed. Fig. 1 shows the proposed ML pipeline.

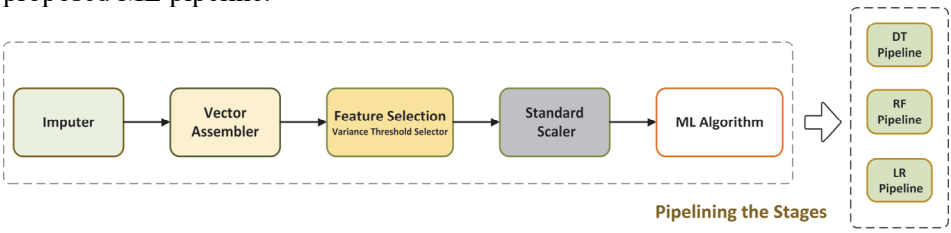

Fig. 1 – *The proposed ML pipeline.*

The pipeline stages are explained as follows:

A Imputer

Dealing with missing values is a crucial step in machine learning, as many algorithms prohibit the presence of such values [14]. The imputer is a transformer used to impute missing values [15] using the numerical feature's mean, median, or mode of the numerical feature [16]. In this work, the mean of each numerical feature is computed and used to replace the null values within that feature. The

input to this stage is a DataFrame with missing values, and the output is a DataFrame with no null values.

B Vector assembler

At this stage, the features are combined into a single vector to prepare them for the next stage, which is the feature selection stage. To do this, a transformer known as the vector assembler assembles multiple numerical columns into a single vector [17]. Vectors come in two forms: dense, which contains many distinct values, and sparse, with mostly zero values. A dense vector is created as an array containing all the values. A sparse vector is created by specifying the vector length, the indices of the non-zero elements, and their values. Sparse vectors are preferred when most values are zero because they are more compact. The input to this stage is the completed DataFrame from the Imputer stage, which consists of multiple separated columns. The output is a modified DataFrame with an additional column that contains the assembled features as a single vector.

C Feature selection

In this stage, the low-variance features are discarded using the variance threshold selector [18]. The variance of each feature is computed according to equation (1), in which σ^2 represents the feature variance, x_i is the feature value, $\bar{x}$ is the feature mean, and n is the number of instances.

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 \ . \tag{1}$$

The input to this stage is the output of the vector assembler stage, and the output is a modified DataFrame that contains the most informative features for the model training.

D Standard scaler

Some features in the input DataFrame showcases a wide range of values, ranging from small to large numbers. In this stage, a transformation is applied to each feature within the vector row, normalizing them to have a zero mean and a standard deviation of one [19]. Although this stage is not mandatory, it can significantly reduce the convergence time, making it an advantageous process to include. The input to this stage is the DataFrame resulting from the Feature Selection stage, and the output is a DataFrame containing each feature's scaled values.

E Machine learning algorithm

It represents the final stage of the pipeline, where the ML model is built utilizing the outputs from the preceding stages. Three ML algorithms are used interchangeably in this process:

1. **Decision Tree:** Decision Tree (DT) is a supervised learning algorithm that handles continuous and discrete data [20]. It uses frequency tables to

make predictions. Data in DT is split continuously according to a specific parameter. It is used to represent decisions and decision-making explicitly [21]. DT is a tree-based model characterized by its clarity in understanding decisions and the ability to select the most favourable features [22]. In addition, it can classify data without extensive computations [23].

2. **Random Forest:** Random Forest (RF) is a supervised learning algorithm used in classification problems. Random Forests comprise many trees, specifically, many decision trees. One of RF's strengths is its efficiency in handling massive training datasets [24].
3. **Logistic Regression:** Logistic Regression (LR) is a predictive supervised learning algorithm for classifying categorical variables. It is built utilizing the concept of probability.

The input to this stage is the preprocessed DataFrame prepared for training, and the output is a trained model capable of learning the underlying patterns in the training data and making predictions on new, unseen data.

4 The End-to-End Machine Learning Workflow

The ML workflow is a sequence of stages to build a predictive model from data [25]. The sequence of these stages and their functionality may vary depending on the problem domain and the available historical data. The suggested ML workflow, Fig. 2, consists of seven stages: Data Ingestion, Exploratory Data Analysis, Data Cleaning, Data Segregation, ML Pipeline Training, Model Evaluation, and Model Deployment. Next, each stage in the workflow is clarified according to Fig. 2**,** which illustrates the suggested ML workflow stages.

A Data ingestion

In the suggested ML workflow, the first six stages represent the offline phase in which a static data source is required. The final stage, i.e., the model deployment stage, represents the online phase in which the data is generated in real-time; hence, it is considered a streaming data source.

There are two types of data ingestion, batching and real-time. In batching data ingestion, the data moves from source to destination as batches at scheduled intervals. While in real-time ingestion, also known as streaming, data is collected and processed from diverse sources. This approach is used when the collected data is time-sensitive, such as traffic statistics in SDN networks.

B Exploratory data analysis

Exploratory Data Analysis (EDA) is an essential stage for initial explorations of the dataset to discover patterns, find anomalies, and check hypotheses with statistical measures and graphical representations. Spark SQL, part of the Spark

ecosystem, is used in the EDA stage as a distributed SQL query engine. The EDA stage consists of the following steps:

1. **DataFrame exploration**: in this step, the DataFrame schema is discovered, and the data type of each column is identified.
2. **DataFrame statistical description**: This step provides a statistical summary of the dataset. The count, minimum, maximum, mean, and standard deviation are found and computed for each feature. These statistics are utilized in the following stages of the ML workflow.

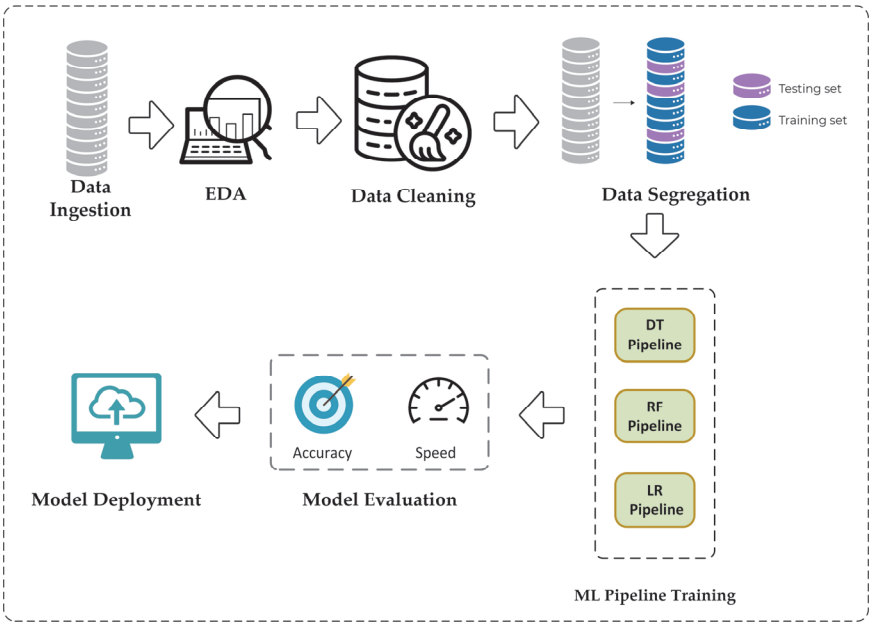

Fig. 2 – *The End-to-End ML Workflow.*

C Data cleaning

Data cleaning involves preparing the DataFrame by removing unnecessary features and duplicated rows to ensure the suitability of data for analysis and modeling. This stage includes the following processes:

– **Removing nominal features:** In this step, the columns with nominal data types are removed from the DataFrame. This is due to the restriction of the vector assembler, which is the second stage in the proposed ML pipeline, to accept only numeric, Boolean, and vector data types.

– **Removing duplicated records:** Duplicated records are removed in this step since they can introduce non-random sampling and may bias the fitted model [26]. However, it is better to bypass this step when the dataset becomes unbalanced after removing the duplicated records. With such

dataset, a high accuracy will be acquired just by predicting the majority class, but the model will fail in capturing the minority class, which is the aim of creating the model.

D Data segregation

After the EDA process, the refined DataFrame is divided randomly into 70% allocated for training and the remaining 30% designated for testing. Two subsets are created, the first subset (training DataFrame) is used to fit the ML pipeline and produce the ML model, and the second subset (testing DataFrame) is used to assess the performance of the produced ML model.

E Machine learning pipeline training

During the training stage, the training DataFrame passes through the five stages of the ML pipeline (imputer, vector assembler, feature selection, standard scaler, and ML algorithm) to produce the ML model. As a result, three ML models are created, DT, RF, and LR.

F Model evaluation

In this stage, the testing DataFrame is utilized to assess the performance of the ML model by comparing the predicted results with the actual outcomes. Three ML models are evaluated: DT, RF, and LR models. The evaluation process includes accuracy measures [27] and ML pipeline training time measures. A block diagram illustrating the Data Segregation stage, the ML Pipeline Training stage, and the Model Evaluation stage is shown Fig. 3.

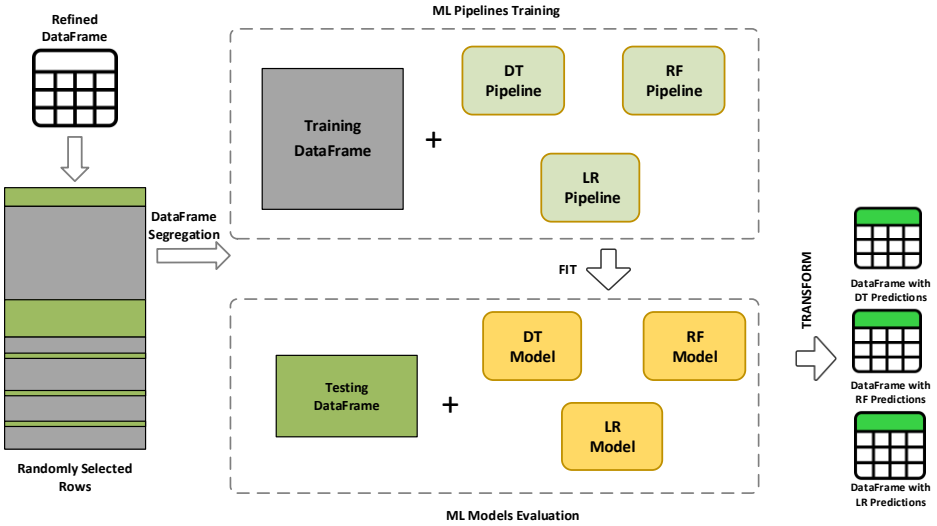

Fig. 3 – *DataFrame segregation and ML pipeline construction and training.*

G Model deployment

At this stage, the trained model is deployed within the production network to generate predictions on incoming data. As depicted in Fig. 4**,** the suggested ML model deployment involves a messaging system, Spark Streaming, and the trained ML model. The messaging system facilitates traffic data transmission from the SDN controller to the Spark Streaming for necessary analysis and processing. The selected messaging system should be scalable, fault-tolerant, resilient, and able to handle vast amounts of real-time data with minimal delay.

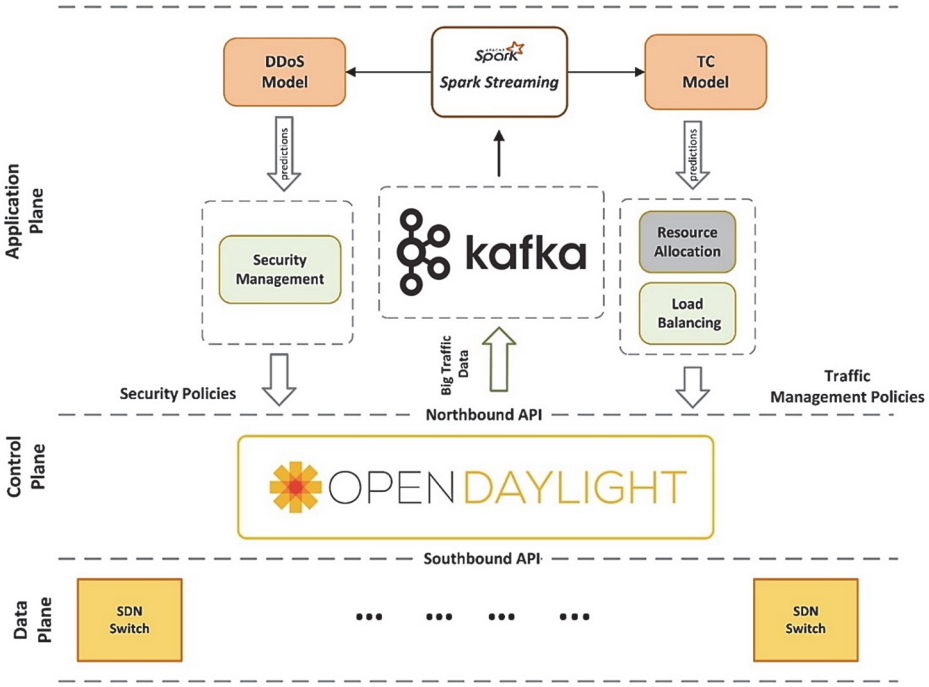

Fig. 4 – *Model Deployment.*

Apache Kafka fulfils all these requirements [28]. Moreover, it can be integrated seamlessly with the SDN employed in this work, i.e., the OpenDayLight (ODL) controller, since it offers a northbound plugin enabling real-time event streaming into Kafka. The ODL controller publishes traffic flow data as messages on Kafka using a common topic, and the Spark Streaming subscribed to receive the message streams. Since the data stream originates from the SDN controller, it must undergo several preprocessing steps before entering the ML model.

The data streams may contain irrelevant or redundant information that could negatively impact the ML model's performance. Preprocessing involves feature selection, data cleaning, and normalization to prepare the data for analysis. Once

the data has been processed, it can be fed into the ML model for processing and analysis. Spark Streaming conducts real-time data cleaning and preprocessing to generate the necessary information for the ML model [29]. The model with the best accuracy and training time performance is selected for the deployment stage. Once the model has been deployed, it is essential to monitor its performance and make adjustments as needed.

5 Results and Discussion

To evaluate the performance of the ML workflow, the "LR-HR DDoS 2024" dataset is processed through the multiple stages that constitute the suggested workflow. The specifics and results of each stage in the workflow are illustrated as follows.

A Data Ingestion

The "LR-HR DDoS 2024" dataset from Kaggle platform [30] is utilized to assess the performance and effectiveness of the ML workflow. The dataset consists of 24 features with 113,407 packets collected throughout the simulation, including 42,899 normal packets and 70,508 malicious packets. Normal packets are labelled as 0 while malicious packets are labelled as 1.

B EDA

The EDA process is performed in two steps:

1. **Dataset Exploration:** the dataset exploration process reveals that all features are numerical. According to the dataset schema, all features are nullable, meaning they can contain missing values.
2. **Dataset Statistical Description:** This process gives a comprehensive overview of each feature in the dataset, including the count, mean, standard deviation, and minimum and maximum values.

As part of the EDA process, the class distribution is demonstrated in Fig. 5, for a total of 113,407 records.

C Data cleaning

This step is skipped in this case, since removing duplicated records will reduce the malicious records to 987 compared to 42,890 normal records, leading to unbalanced dataset.

D Data segregation

After completing the EDA process, the processed dataset is divided randomly into two subsets, 70% for training and 30% for testing. The two subsets have different purposes. The training dataset is further split into five folds for cross validation, while the testing dataset is used to evaluate the model's

performance on unseen data. The class distribution for the training and testing datasets is demonstrated in Fig. 6.

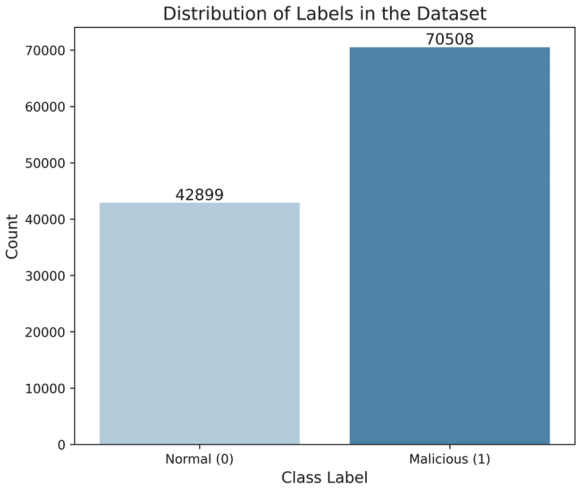

Fig. 5 – *Class Distribution for the LR-HR DDoS 2024 dataset.*

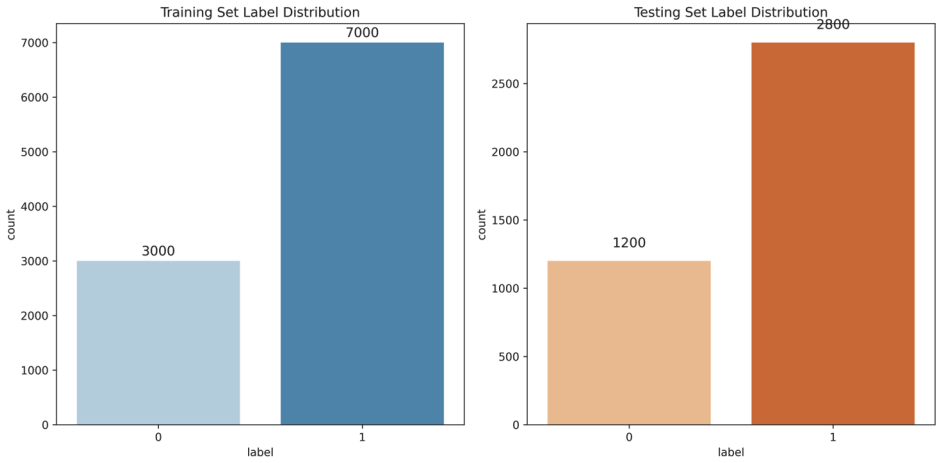

Fig. 6 – *Class Distribution for the training and testing datasets.*

E Machine learning pipeline training

Each ML pipeline (DT, RF, LR) is trained using the training data to produce a ML model. Therefore, three ML models are produced: DT, RF, and LR. A brief description of each stage of the ML pipeline with the adopted dataset is as follows:

1. **Imputer:** At this stage of the ML pipeline, missing values in each feature are filled with the mean value of that particular feature. Even though the dataset used has no missing values, incorporating an imputer in the preprocessing step enhances the robustness of the ML pipeline, ensuring it can effectively handle missing data in future datasets.
2. **Vector Assembler:** At this stage, features are merged into a single vector column, serving as input for the feature selection process. The vector is dense, as its features are primarily populated with non-zero values. The vector length is 24, corresponding to the number of features included in the vector.
3. **Feature Selection:** As mentioned earlier, the variance threshold selector is applied for feature selection with a threshold set to 0.6. This means features with a variance less than or equal to 0.6 are removed. These features exhibit low variability or a limited range of values, providing minimal information for analysis or predictive modeling. As a result, the vector length is reduced from 24 to 16, with the following features removed, *Fwd PSH Flags*, *Bwd PSH Flags*, *Fwd URG Flags*, *Bwd URG Flags*, *FIN Flag Cnt*, *SYN Flag Cnt*, *RST Flag Cnt*, and *Init Fwd Win Byts*.
4. **Standard Scaler:** This stage normalizes each selected feature with a unit standard deviation and zero mean.
5. **Machine Learning Algorithm:** As mentioned earlier, three machine learning models are employed in this step: Decision Tree (DT), Random Forest (RF), and Logistic Regression (LR). The DT algorithm is chosen for its interpretability and its ability to handle non-linear relationships effectively. The RF algorithm is included to enhance accuracy and mitigate overfitting. Finally, the LR algorithm is selected due to its simplicity and its effectiveness in handling linear problems. For cross-validation, the dataset is split into five folds. The hyperparameters for the DT, RF and LR are set according to **Table 1**.

Table 1
Hyperparameter tuning for the DT, RF, and LR algorithms.

ML Algorithm	Parameters
DT	maxDepth: 5, 8, 10 maxBins: 16, 32
RF	numTrees: 100, 200 maxDepth: 5, 8
LR	regParam: 0.01, 0.1, 0.5 elasticNetParam: 0.0, 0.5, 1.0

F Machine learning models evaluation

After building the ML models, they are evaluated to select the best candidate in terms of four metrics: accuracy, recall, precision and F1 score. Accuracy is calculated according to the following equation [31]:

$$\text{Accuracy} = \frac{\textit{True Positives} + \textit{True Negatives}}{\textit{All Samples}}.$$

The recall ratio is computed as follows:

$$\text{Recall} = \frac{\textit{True Positives}}{\textit{True Positives} + \textit{False Negatives}}.$$

The precision ratio condenses the model performance in predicting the positive classes. It is calculated as follows:

$$\text{Precision} = \frac{\textit{True Positives}}{\textit{True Positives} + \textit{False Positives}}.$$

F1-score is computed according to the following equation:

$$\text{F1-score} = 2 \times \frac{\textit{Precision} \cdot \textit{Recall}}{\textit{Precision} + \textit{Recall}}.$$

The best model in terms of the aforementioned metrices is used in the deployment stage. In this work, Databricks on AWS is used as the experimental environment. With this platform, it becomes possible to handle all data storage and management needs on a single, accessible platform that integrates data warehouses and data lakes. This unified lakehouse approach streamlines analytics and AI workloads. The cluster consists of three nodes: one driver and two executors. Each node is an Amazon EC2 i3.4xlarge machine, as seen in Fig. 7.

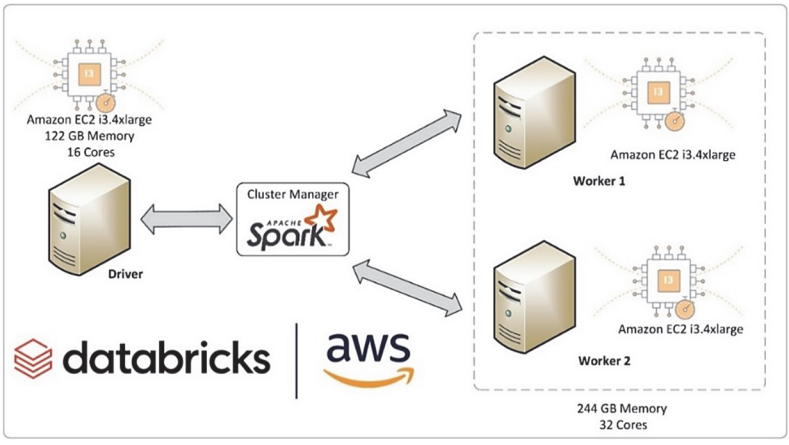

Fig. 7 – *Experimental environment using Databricks on AWS.*

Table 2 shows the overall statistics and training time for the three ML pipelines. DT pipeline demonstrates superior performance in terms of accuracy, F1 score, recall, precision and training time compared to the other candidates. Therefore, the DT model is the best candidate to be used in the deployment stage.

Table 2
Evaluation metrics for the DT, RF, and LR ML pipelines within the ML workflow.

ML Pipeline	Accuracy	F1	Recall	Precision	Training time
DT	0.9970397	0.996867	0.99703974	0.99702276	4 min 33s
RF	0.996686	0.996188	0.9966862	0.99634872	5 min 41s
LR	0.9803287	0.974007	0.9803287	0.97081757	7 min 47s

A part of the evaluation process is to compare the proposed work with the state-of-the-art as shown in **Table 3**.

Table 3
Comparison among the proposed system and the state of the art work.

Ref.	DL/ML Model	Dataset	Distributed Framework
[5]	RNN, LSTM, CNN	ISCX 2012 dataset	-
[6]	XGBoost	KDD Cup 1999	-
[7]	Advanced SVM	Custom dataset	-
[8]	DT, RF, LR, MLP	Custom dataset	-
[9]	CNN, LSTM	ROAD dataset	-
The proposed model	DT, RF, LR	LR-HR DDoS 2024	Spark

7 Conclusion

This work demonstrates the practicality and effectiveness of incorporating Spark into SDN environments, providing a foundation for enhanced traffic analysis and real-time decision-making processes. Using big data techniques, such as Spark, in SDN networks has exhibited significant advantages, including the architectural design of ML pipelines and the construction of ML models. This research has successfully designed and implemented three ML pipelines to be deployed in the suggested ML workflow. Each pipeline is used for building a specific ML model. As a result, the time dedicated to planning and setting up the ML workflow is eliminated, fostering efficiency in ML model development.

The constructed pipelines have been employed to build three ML models for detecting DDoS attacks in SDN networks. By leveraging Spark and its modules,

the time required for model construction has been significantly reduced. The distributed processing capabilities of Spark, running across a cluster, enable the parallelization of the complete workflow.

Future work will focus on enhancing the ML pipelines' architecture, capabilities, and workflow. This could involve incorporating more advanced feature engineering techniques, exploring diverse ML algorithms or ensembles, and optimizing the pipeline's performance for specific network scenarios, such as anomaly detection, predictive maintenance, network optimization, or other relevant applications. In addition, future work will focus on simulating the method within an actual SDN environment.

8 Acknowledgments

The authors would like to thank the Editor-in-Chief and anonymous referees for their suggestions and helpful comments that have improved the paper's quality and clarity.

9 References

[1] M. Hussain, N. Shah, R. Amin, S. S. Alshamrani, A. Alotaibi, S. M. Raza: Software-Defined Networking: Categories, Analysis, and Future Directions, Sensors, Vol. 22, No. 15, August 2022, p. 5551.

[2] S. C. Narayanan, V. Varghese: A Proactive Controller Failure Recovery Mechanism in SD-WAN with Multiple Controllers, Serbian Journal of Electrical Engineering, Vol. 21, No. 2, June 2024, pp. 235 – 250.

[3] C. Urrea, D. Benítez: Software-Defined Networking Solutions, Architecture and Controllers for the Industrial Internet of Things: A Review, Sensors, Vol. 21, No. 19, October 2021, p. 6585.

[4] J. Wang, L. Wang: SDN-Defend: A Lightweight Online Attack Detection and Mitigation System for DDoS Attacks in SDN, Sensors, Vol. 22, No. 21, November 2022, p. 8287.

[5] C. Li, Y. Wu, X. Yuan, Z. Sun, W. Wang, X. Li, L. Gong: Detection and Defense of DDoS Attack-Based on Deep Learning in OpenFlow-Based SDN, International Journal of Communication Systems, Vol. 31, No. 5, March 2018, p. e3497.

[6] Z. Chen, F. Jiang, Y. Cheng, X. Gu, W. Liu, J. Peng: XGBoost Classifier for DDoS Attack Detection and Analysis in SDN-Based Cloud, Proceedings of the IEEE International Conference on Big Data and Smart Computing (BigComp), Shanghai, China, January 2018, pp. 251 – 256.

[7] M. Myint Oo, S. Kamolphiwong, T. Kamolphiwong, S. Vasupongayya: Advanced Support Vector Machine- (ASVM-) Based Detection for Distributed Denial of Service (DDoS) Attack on Software Defined Networking (SDN), Journal of Computer Networks and Communications, Vol. 2019, January 2019, p. 8012568.

[8] R. Swami, M. Dave, V. Ranga: Detection and Analysis of TCP-SYN DDoS Attack in Software-Defined Networking, Wireless Personal Communications, Vol. 118, No. 4, June 2021, pp. 2295 – 2317.

[9] J. Singh, S. Behal: Detection and Mitigation of DDoS Attacks in SDN: A Comprehensive Review, Research Challenges, and Future Directions, Computer Science Review, Vol. 37, August 2020, p. 100279.

[10] A. Posoldova: Machine Learning Pipelines: From Research to Production, IEEE Potentials, Vol. 39, No. 6, November 2020, pp. 38 – 42.

[11] E. Haihong, K. Zhou, M. Song: Spark-Based Machine Learning Pipeline Construction Method, Proceedings of the International Conference on Machine Learning and Data Engineering (iCMLDE), Taipei, Taiwan, December 2019, pp. 1 – 6.

[12] M. Mirkov, A. Gavrovska: Tumor Detection Using Brain MRI and Low-Dimension Co-Occurrence Feature Approach, Serbian Journal of Electrical Engineering, Vol. 19, No. 3, October 2022, pp. 273 – 289.

[13] O. Azeroual, A. Nikiforova: Apache Spark and MLlib-Based Intrusion Detection System or How the Big Data Technologies Can Secure the Data, Information, Vol. 13, No. 2, February 2022, p. 58.

[14] S. S. Samaan, H. A. Jeiad: Architecting a Machine Learning Pipeline for Online Traffic Classification in Software Defined Networking Using Spark, IAES International Journal of Artificial Intelligence (IJ-AI), Vol. 12, No. 2, June 2023, pp. 861 – 873.

[15] A. I. Kawoosa, D. Prashar, M. Faheem, N. Jha, A. A. Khan: Using Machine Learning Ensemble Method for Detection of Energy Theft in Smart Meters, IET Generation, Transmission & Distribution, Vol. 17, No. 21, November 2023, pp. 4794 – 4809.

[16] M. Gupta, B. Gupta: A New Scalable Approach for Missing Value Imputation in High-Throughput Microarray Data on Apache Spark, International Journal of Data Mining and Bioinformatics, Vol. 23, No. 1, February 2020, pp. 79 – 100.

[17] S. Bagui, M. Walauskis, R. DeRush, H. Praviset, S. Boucugnani: Spark Configurations to Optimize Decision Tree Classification on UNSW-NB15, Big Data and Cognitive Computing, Vol. 6, No. 2, June 2022, p. 38.

[18] R. Dhanya, I. R. Paul, S. S. Akula, M. Sivakumar, J. J. Nair: F-Test Feature Selection in Stacking Ensemble Model for Breast Cancer Prediction, Procedia Computer Science, Vol. 171, 2020, pp. 1561 – 1570.

[19] H. Ahmed, E. M. G. Younis, A. A. Ali: Predicting Diabetes Using Distributed Machine Learning based on Apache Spark, Proceedings of the International Conference on Innovative Trends in Communication and Computer Engineering (ITCE), Aswan, Egypt, February 2020, pp. 44 – 49.

[20] G. Kumar, H. Alqahtani: Machine Learning Techniques for Intrusion Detection Systems in SDN-Recent Advances, Challenges and Future Directions, CMES - Computer Modeling in Engineering and Sciences, Vol. 134, No. 1, August 2022, pp. 89 – 119.

[21] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, Y. Liu: A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges, IEEE Communications Surveys & Tutorials, Vol. 21, No. 1, Firstquarter 2019, pp. 393 – 430.

[22] N. F. Idris, M. A. Ismail, M. S. Mohamad, S. Kasim, Z. Zakaria, T. Sutikno: Breast Cancer Disease Classification Using Fuzzy-ID3 Algorithm based on Association Function, IAES International Journal of Artificial Intelligence, Vol. 11, No. 2, June 2022, pp. 448 – 461.

[23] R. M. Aziz, A. Hussain, P. Sharma, P. Kumar: Machine Learning-Based Soft Computing Regression Analysis Approach for Crime Data Prediction, Karbala International Journal of Modern Science, Vol. 8, No. 1, January 2022, p. 1.

[24] Z. Ali Mohammed, M. N. Abdullah, I. H. Al-Hussaini: Predicting Incident Duration Based on Machine Learning Methods, Iraqi Journal of Computer, Communication, Control and Systems Engineering, Vol. 21, No. 1, March 2021, pp. 1 – 15.

[25] M. E. Maros, D. Capper, D. T. W. Jones, V. Hovestadt, A. von Deimling, S. M. Pfister, A. Benner, M. Zucknick, M. Sill: Machine Learning Workflows to Estimate Class Probabilities for Precision Cancer Diagnostics on DNA Methylation Microarray Data, Nature Protocols, Vol. 15, No. 2, February 2020, pp. 479 – 512.

[26] H. Kamel, M. Z. Abdullah: Distributed Denial of Service Attacks Detection for Software Defined Networks Based on Evolutionary Decision Tree Model, Bulletin of Electrical Engineering and Informatics, Vol. 11, No. 4, August 2022, pp. 2322 – 2330.

[27] H. Alqahtani, G. Kumar: Deep Learning-Based Intrusion Detection System for In-Vehicle Networks with Knowledge Graph and Statistical Methods, International Journal of Machine Learning and Cybernetics, Vol. 16, No. 5-6, June 2025, pp. 3539 – 3555.

[28] B. Zhou, J. Li, X. Wang, Y. Gu, L. Xu, Y. Hu, L. Zhu: Online Internet Traffic Monitoring System Using Spark Streaming, Big Data Mining and Analytics, Vol. 1, No. 1, March 2018, pp. 47 – 56.

[29] B. Zhou, J. Li, J. Wu, S. Guo, Y. Gu, Z. Li: Machine-Learning-Based Online Distributed Denial-of-Service Attack Detection Using Spark Streaming, Proceedings of the IEEE International Conference on Communications (ICC), Kansas City, USA, May 2018, pp. 1 – 6.

[30] A. Ahmed: LR-HR DDoS 2024 Dataset for SDN-Based Networks, Available at: https://www.kaggle.com/datasets/abdussalamahmed/lr-hr-ddos-2024-dataset-for-sdn-based-networks, [Accessed: 21-Dec-2024].

[31] S. S. Samaan, H. A. Jeiad: Feature-Based Real-Time Distributed Denial of Service Detection in SDN Using Machine Learning and Spark, Bulletin of Electrical Engineering and Informatics, Vol. 12, No. 4, August 2023, pp. 2302 – 2312.