Sarat Mahavratayajula Veeravenkata Maruthi,[1] Lakshmi Ganesh Nerella[2]

AI Enhanced DevSecOps Pipeline for Automated Database Deployment & Security

Journal of Electrical Systems

Abstract: - With rising security concerns in software systems, DevSecOps has emerged to embed security into DevOps workflows; however, this integration often challenges agility and slows delivery speed. To address this challenge, this study presents an AI-enhanced DevSecOps pipeline designed for secure and automated database deployment, leveraging machine learning for real-time intrusion detection without compromising speed. The methodology incorporates rigorous preprocessing, including feature selection, class balancing with SMOTE, and feature scaling, followed by training and evaluation of ML models such as K-Nearest Neighbor (KNN), Multi-Layer Perceptron (MLP), and Logistic Regression (LR). Experimental results demonstrate that KNN outperforms other models with an accuracy of 99.57%, precision of 99.80%, recall of 99.55%, and F1-score of 99.67. In addition, the suggested models demonstrate better performance and scalability when dealing with the complicated TII-SSRC-23 dataset when compared to current methods such as Decision Tree (DT), J48, NaïveBayes (NB), and LSTM. The best-performing model was integrated into a CI/CD pipeline. This enabled intelligent threat detection during deployment. The work shows that security can be embedded proactively into DevOps. AI helps preserve agility while strengthening cybersecurity. The framework is scalable and high-performing.

Keywords: DevSecOps, Software Development, Automated Deployment, Cyber Threat Detection, security Pipeline, AI-Driven Machine Learning

I. INTRODUCTION

The fundamental features that secure software must have when developed, deployed, configured, and maintained include the ability to either continue running during computer assaults or to minimize damage and recover as fast as feasible [1]. Achieving this requires adopting a proactive security mindset that integrates defense mechanisms from the earliest phases of the SDLC [2]. However, in reality, software security is generally reactive, with vulnerabilities corrected after release [3][4]. The ever-changing software development techniques highlight the shortcomings of reactive security. Innovation has been expedited and release times have been decreased because to continuous delivery pipelines, microservices, cloud-native architectures, and agile approaches [5]. At the same time, these advancements expand the attack surface and introduce new security vulnerabilities [6][7]. Traditional manual and fragmented security practices are proving inadequate in countering sophisticated and dynamic cyber threats [8][9][10]. As a result, a paradigm change is required to include security comprehensively and constantly into the SDLC [11][12]. A critical aspect of this shift is deployment optimization. Modern enterprises require fast, reliable, and resource-efficient releases, yet traditional methods often face long cycles, poor resource allocation, and failures from weak microservice coordination [13].

Optimizing deployments involves reducing memory usage, communication overhead, and execution costs, an NP-complete problem where exact solutions are impractical [14][15]. Thus, intelligent near-optimal strategies are essential to balance efficiency with scalability. This is where DevSecOps plays a transformative role. In this context, DevSecOps has emerged as a promising paradigm that embeds security practices into CI/CD pipelines, ensuring rapid and secure software delivery [16]. However, with the rise of heterogeneous infrastructures such as edge computing, new challenges arise in orchestrating distributed nodes, deploying AI models for real-time decision-making, and securing decentralized environments. Addressing these issues requires extending DevSecOps with advanced intelligence and automation. This research proposes an AI-Enhanced DevSecOps Pipeline for Automated Database Deployment and Security. The framework integrates artificial intelligence into CI/CD pipelines for anomaly detection, predictive threat modeling, and adaptive resource allocation. By combining intelligent automation with DevSecOps principles, the proposed approach enables secure, efficient, and resilient database deployments while mitigating risks earlier in the lifecycle and reducing reliance on manual interventions.

[1] Independent Researcher, USA
[2]Sr. Database Administrator, Summerfield, NC, USA

A. Motivation and Contributions of the Study

Modern software development faces the dual challenge of accelerating deployment while ensuring robust security, yet current practices remain largely reactive, addressing vulnerabilities only after deployment, which increases costs, delays remediation, and leaves systems exposed to sophisticated cyber threats. The complexity intensifies in database deployment scenarios, where traditional methods struggle with inefficient resource allocation, prolonged cycles, and failures in coordinating microservices. Since deployment optimization is an NP-complete problem, exact solutions are infeasible, demanding intelligent, near-optimal approaches. At the same time, emerging infrastructures such as cloud-native and edge environments expand the attack surface and require real-time, secure decision-making. While DevSecOps offers a paradigm for embedding security into CI/CD pipelines, its effectiveness is limited without automation and intelligence. This motivates the need for an AI-enhanced DevSecOps pipeline that integrates predictive threat detection, anomaly monitoring, and adaptive resource allocation to enable secure, efficient, and resilient database deployments in dynamic environments.

- Utilized the publicly available TII-SSRC-23 dataset from Kaggle, which provides a large-scale, realistic representation of network traffic.
- A robust preprocessing pipeline that integrates feature selection, class balancing, and normalization to improve model readiness and accuracy.
- Comprehensive visualization and feature analysis to better understand traffic patterns and key attributes influencing intrusion detection.
- Evaluation of multiple ML models (KNN, LR, MLP) with identification and deployment of the best-performing model.
- Developed a novel DevSecOps framework that embeds machine learning models directly into the CI/CD pipeline for proactive and automated threat detection during database deployment.
- Conducted an extensive comparison of KNN, MLP, and LR models using accuracy, precision, recall, and F1score to identify a most suitable classifier for pipeline integration.

B. Justification and Novelty

The justification for the proposed AI-enhanced DevSecOps pipeline lies in its ability to address the limitations of existing security approaches, which often suffer from manual intervention, delayed detection, high false positives, and lack of adaptability in dynamic deployment environments. Unlike traditional rule-based or single-model methods, the proposed pipeline integrates advanced machine learning models, particularly KNN and MLP, achieving higher accuracy and robustness in detecting threats with minimal errors. Its novelty stems from embedding these models within a fully automated CI/CD-enabled DevSecOps framework that not only preprocesses and analyzes data with high precision but also proactively blocks risky deployments, logs threats for auditing, and continuously adapts through monitoring and retraining. This closed-loop, AI-driven approach ensures real-time protection, scalability, and resilience, setting it apart from prior works that either focused solely on model performance or DevSecOps practices without tightly coupling automation, adaptability, and security enforcement.

C. Organisation of the Paper

The paper is organized as follows: Security-related tasks for the DevSecOps Pipeline is covered in Section II. The methods and data pre-processing are described in Section III. Model comparisons and experimental findings are shown in Section IV. The article is concluded and future work is outlined in Section V.

II. LITERATURE REVIEW

Table I presents a comparative summary of studies using AI, ML, and DL in DevSecOps and cybersecurity, highlighting improved threat detection, automation, and model efficiency. Existing research focuses on applying advanced data-driven techniques to intrusion detection and cyber threat intelligence, addressing challenges like class imbalance and feature selection to enhance detection accuracy.

Herzalla, Lunardi and Andreoni (2023) address these issues by introducing TII-SSRC-23, a new and extensive dataset. Their dataset covers a big diversity of traffic sorts and subtypes, therefore it could be useful for researchers.

It also does a feature importance analysis, which sheds light on the most important characteristics for intrusion detection tasks. In the constantly evolving field of network security, their dataset also helps to create and adjust IDS models by providing solid foundations for supervised and unsupervised intrusion detection techniques via rigorous testing.

Ramaj et al. (2022) provide a synopsis of the compliance-related components of DevSecOps and go into the methods used to guarantee compliance. In addition, the study finds possible avenues for future research in this area and discloses compliance patterns based on the existing literature. Consequently, they carried out a comprehensive review of the literature on the incorporation of compliance elements into DevSecOps, closely according to Kitchenham and Charters' guidelines. By searching 5 bibliographic databases (163) and Google Scholar (771), it found 934 papers on the topic. After a thorough screening procedure, they chose 15 publications as main studies. Next, they divided the DevSecOps compliance components into 3 main groups: compliance minutiae, compliance administration, and compliance initiation. Their observation of a lack of research leads them to recommend more investigation into compliance aspects, their automated integration, and the development of metrics to evaluate this process within the framework of DevSecOps.

Bahaa et al. (2021) discovered a total of 49 original papers, and seven distinct ML approaches were used to build the detection models. Public datasets like NSL-KDD and UNSW-NB15 were utilized by some of the main research, but the majority relied on IoT device testbed datasets. Numerical and graphical metrics are often utilized to assess an efficacy of models. The research indicates that the majority of IoT assaults take place at the network layer. Detection models that included DevSecOps pipelines into the creation of IoT devices were more secure. The findings of this research indicate that IoT assaults may be detected by machine learning approaches; nevertheless, there are certain problems with the detection model design. Additionally, it suggests keeping hybrid frameworks in place to better identify IoT assaults, configuring monitoring infrastructure with sophisticated methodologies based on software pipelines, and using ML techniques for enhanced supervision and monitoring.

Based on the reviewed studies, a clear research gap emerges in the integration of AI-, ML-, and DL-based security approaches within practical DevSecOps pipelines. While existing research demonstrates improvements in threat detection, automation, and dataset availability, challenges remain in addressing automation gaps, domain-specific scalability, and compliance integration. Most studies emphasize model accuracy or dataset creation but fall short in operational deployment, real-time monitoring, and balancing performance overhead with security effectiveness. Furthermore, compliance aspects, container security, and IoT-focused DevSecOps practices are still underexplored, leaving room for research on unified frameworks that holistically combine automation, compliance, scalability, and adaptive threat intelligence within DevSecOps environments.

TABLE I. COMPARATIVE SUMMARY OF AI, ML, AND DL APPLICATIONS IN DEVSECOPS AND CYBERSECURITY

Reference	Methods	Results	Advantages	Limitations	Recommendation
Herzalla, Lunardi & Andreoni (2023)	Development of TII-SSRC-23 dataset; feature importance analysis; IDS experiments with supervised & unsupervised learning	Dataset supports diverse traffic analysis; provides baselines for IDS models	Comprehensive dataset; enables reproducibility and benchmarking; identifies critical IDS features	Limited to dataset-centric contributions; real-world applicability not tested at scale	Extend validation in operational networks; expand dataset with emerging attack types
Ramaj et al. (2022)	DevSecOps compliance (934 articles 15 primary studies)	Identified compliance aspects: initiation, management, technicalities	First structured review on DevSecOps compliance; highlights research gap	Few studies available; lack of automated compliance approaches	Develop automated compliance metrics; expand research into compliance integration
Bahaa et al. (2021)	Literature review of 49 primary studies	ML models detect IoT attacks	Demonstrates ML's potential for IoT	Detection model design	Use hybrid frameworks;

	using ML for IoT attack detection; datasets: NSL-KDD, UNSW-NB15, IoT testbeds	effectively; hybrid frameworks recommended; most attacks at network layer	security; hybrid models show promise	challenges; limited pipeline integration	strengthen IoT security with DevSecOps pipelines; develop advanced monitoring

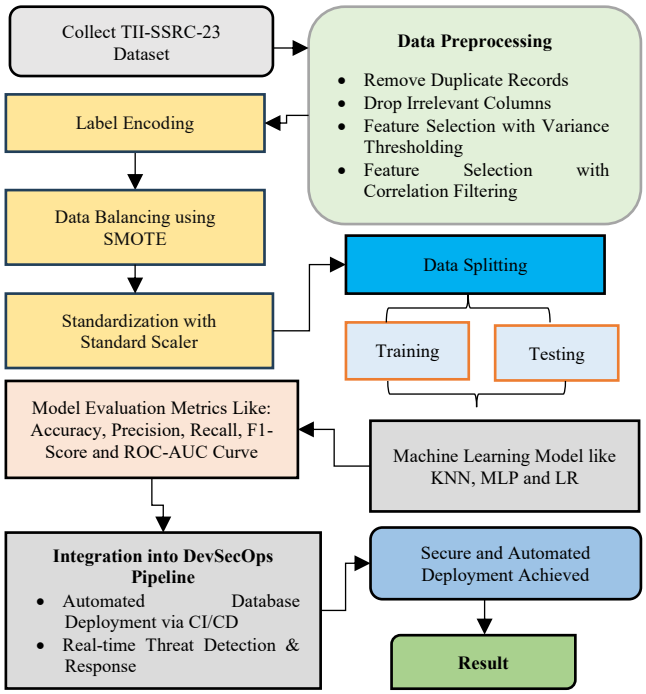


Fig. 1. Flowchart of the Proposed AI-Enhanced DevSecOps Pipeline

III. METHODOLOGY

The proposed methodology embeds machine learning into a secure DevSecOps pipeline to enable intelligent, real-time threat detection during automated database deployment. The process starts with data collection from Kaggle, followed by data splitting into 80% for training and 20% for testing. A comprehensive preprocessing stage is performed, involving removal of duplicate records and irrelevant columns, label encoding, feature selection through variance thresholding and correlation filtering, data balancing using SMOTE and standard scaler. Several machine learning models such as KNN, MLP, and LR are trained. Several measures are used to assess these models, including recall, accuracy, precision, F1score, and ROC-AUC curve. The pipeline monitors performance, updates models when needed, and embeds security scans, intrusion detection, and compliance checks to ensure real-time protection and secure, efficient deployment. The entire methodology is illustrated in Figure 1, presenting a clear flow from dataset acquisition to achieving secure and automated deployment.

Each step and phase of the proposed flowchart are detailed and explained in below:

A. Data Collection and Analysis

The TII-SSRC-23 dataset[2], publicly available on Kaggle, offers a realistic and extensive collection of network traffic for intrusion detection in SDN environments. It includes both tabular CSV data with 86 extracted features and raw PCAP files for in-depth traffic analysis. The dataset covers 6,925,413 instances across eight traffic types and 32 subcategories, six benign and 26 malicious capturing diverse attack scenarios such as DoS, DDoS, reconnaissance, exploitation, and Mirai botnet activity. A strong benchmark for assessing AI-based security solutions in DevSecOps pipelines, it is generated employing real-world SDN traffic and controller logs. Data

[2] https://www.kaggle.com/datasets/danielherzalla/tii-ssrc-23

visualization is the process of visually portraying data or information, often using maps, graphs, charts, and other visual components as described below:

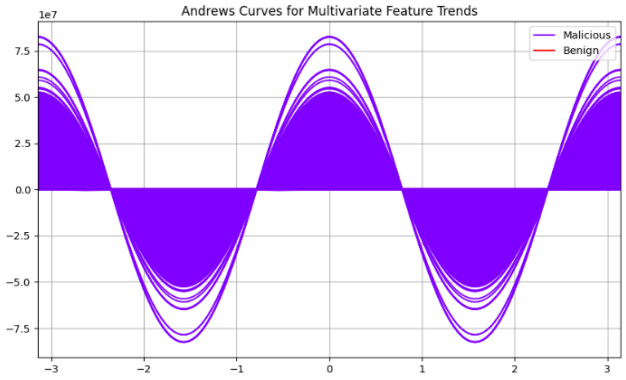


Fig. 2. Andrews Curves Multivariate feature trends

Figure 2 shows the Andrews Curves used to visualize multivariate feature trends. Most curves, in purple, represent malicious data, while a barely visible red line indicates sparse or overlapping benign data. The curves follow a wave-like pattern, with x-values from -3 to 3 and y-values from -8×10^7 to 8×10^7. This helps reveal the structure and distribution of malicious patterns in high-dimensional space.

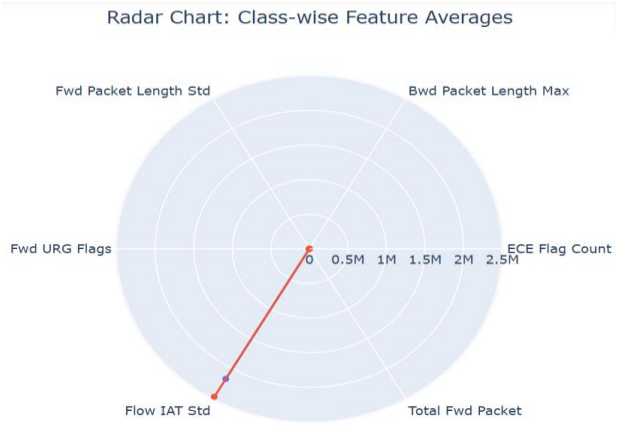


Fig. 3. Radar Chart (Spider Plot)

Figure 3 shows a radar chart of average values for selected features of a specific class. Axes represent features like Fwd Packet Length Std, Bwd Packet Length Max, ECE FlagCount, Total Fwd Packet, FlowIAT Std, and FwdURG Flags. Values range from 0 to 2.5 million. The red line indicates that Flow IAT Std has a much higher average than the other features, which remain near zero.

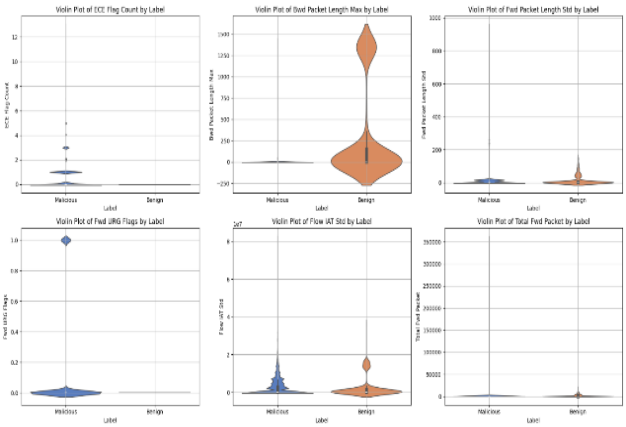


Fig. 4. Violin Plots for Top 6 selected features

Figure 4 displays a 2x3 grid of violin plots representing the distributions of the top six selected network traffic features across malicious and benign categories. Features such as Bwd Packet Length Max and Flow IAT Std show noticeable differences in value ranges between the two classes, while ECE Flag Count and Fwd URG Flags are predominantly present in malicious traffic. These visual patterns support the relevance of the selected features in distinguishing different types of network behavior.

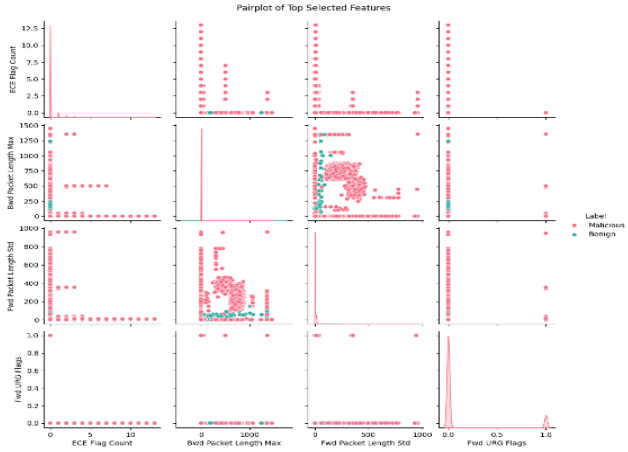


Fig. 5. Pair plot for top 4 selected Features

Figure 5 shows scatter plots for all pairwise combinations of four selected features: ECE FlagCount, BwdPacketLength Max, FwdPacketLength Std, and Fwd URG Flags. The diagonal plots display the univariate distributions (e.g., histograms or kernel density estimates) for each feature, separated by malicious (red) and benign (teal) labels. Off-diagonal plots illustrate the relationships between feature pairs, also color-coded by class.

B. Data Preprocessing

The data preprocessing phase ensures that the TII-SSRC-23 dataset is transformed into a clean, balanced, and model-ready format for accurate intrusion detection. The data preprocessing involved several key steps removing duplicates, irrelevant identifiers, and missing values, followed by variance thresholding and correlation pruning to retain informative features. The target labels were encoded, class imbalance was addressed using undersampling with SMOTE, and features were standardised with Standard Scaler to prepare balanced, normalised data for model training. These pre-processing steps are described in below:

- **Handling Duplicate Records in the Dataset:** As a basic data cleaning step, the dataset was checked for duplicates and 769 repeated rows were found. These were removed to ensure data accuracy, leaving 6,924,644 unique records for analysis.
- **Dropping Irrelevant Columns:** As a basic preprocessing step, the following columns were removed: 'FlowID', 'SrcIP', 'SrcPort', 'Dst IP', 'DstPort', 'Timestamp', 'Traffic Type', and 'Traffic Subtype'. These fields are identifiers that do not support threat prediction and may introduce noise, so they were excluded.

C. Feature Selection Using Variance Thresholding

Feature selection was performed using Variance Thresholding to eliminate low-variance features and retain only informative variables. Specifically, numerical features of types int64 and float64 were selected, and a variance threshold of 0.01 was applied. This procedure eliminates variables with little to no variance between samples, since it contribute little to predictive modelling. As a result, Variance Thresholding effectively reduced the dimensionality of the dataset, retaining 72 features with sufficient variability for further analysis and modeling.

D. Feature Selection with Correlation Filtering

After applying variance thresholding, correlation filtering was performed to eliminate redundant features. The absolute correlation matrix was computed for the selected numerical features, and highly correlated pairs (correlation > 0.9) were identified using the upper triangle method. One feature from each correlated pair was

removed to reduce multicollinearity. As a result, the final dataset was reduced to 45 non-redundant features for subsequent analysis and modelling as shows in Figure 6.

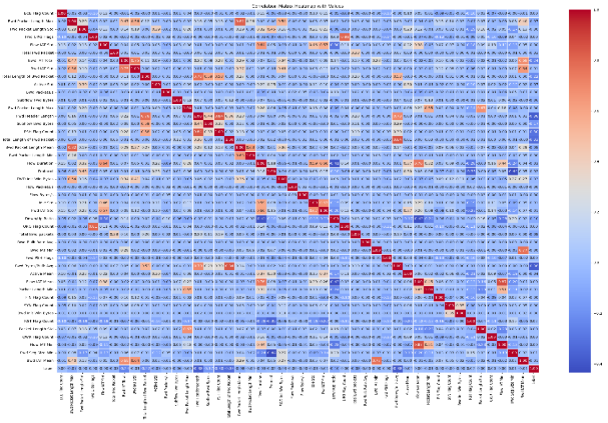


Fig. 6. Correlation heatmap of the 45 numerical features

Figure 6 presents a Pearson correlation heatmap of the 45 numerical features retained after correlation filtering. An x- and y-axis shows the same set of variables, and each cell indicates the correlation coefficient between a pair of features. Deep blue indicates a severe negative correlation (e.g., -0.84), white indicates neutrality (~0), and deep red indicates a high positive correlation (e.g., 1.00 on the diagonal). The consistent red diagonal reflects perfect self-correlation (1.00), while the absence of strong off-diagonal correlations (e.g., values > 0.9) confirms the effectiveness of the filtering in reducing multicollinearity.

E. Label Encoding

Label encoding is one of the most well-known DL methods for converting numerical input to a categorical format. Facilitating the algorithmic processing of data entails numerically valuing each separate category variable. Label encoding is one of the most adopted encoding schemes owing to the simplicity of the conversion mechanism. The data values are converted to numbers from the list of enumerations of the different values represented by the feature. Considering that feature F in each sample *i* is represented by one of the *M* categorical values such that $f_i \epsilon\ C$ where C = {c_1,c_2,…, c_M}, label encoding simply assigns the numerical index value of the category to which the sample feature value belongs, in Equation (1).

$$f_{i_{label}} = k \; \forall \; f_i = c_k \; where \; i = 1,2, \dots, N \qquad (1)$$

k = 1, 2,…, M, Although label encoding is easy to implement, it produces implicit ordinality among the converted values even though none exists. The assigned values can also misrepresent the meaning, thereby impacting the model performance.

F. Data balancing with SMOTE

A broader region is covered by minority classes thanks to SMOTE, which produces synthetic instances near existing instances. Classifiers are now better able to uncover previously unseen occurrences of minority groups. SMOTE generates a large area for minority cases using an oversampling approach. Thus, SMOTE will be a major player in the feature market. A larger feature space area and synthetic minority occurrences are introduced by SMOTE. It does, however, raise class overlaps and add more noise.

G. Standardization using Standard Scaler

Assuming that the data inside each component follows a normal distribution, the Standard Scaler adjusts the scale of each component so that its distribution is centred at 0 and its standard deviation is 1. Equation (2) is used to scale the component after the determination of its mean and standard deviation:

$$Z_{scaled} = \frac{(X-\mu)}{\sigma} \qquad (2)$$

Where

- $\mu =$ Mean
- $\sigma =$ Standard Deviation.

H. Machine Learning Models in AI-Enhanced DevSecOps

This study implement KNN, MLP, and LR classifiers for cyber threat detection using the TII-SSRC-23 dataset, aiming to enhance DevSecOps pipeline security.

1) K-Nearest Neighbor (KNN)

An extremely popular and well-established ML technique, the KNN algorithm finds use in a broad range of real-world scenarios. The KNN algorithm is a supervised classification method that uses distance calculations to find the nearest neighbor (K) in order to establish a new class according to predetermined rules. It takes a test sample (X) as its starting point and grows in the area until it encompasses all of the training samples (K).Equation (3) presented in the following manner:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \quad (3)$$

The points in Cartesian coordinates are denoted by x_i and y_i in the above Equation (3), with n standing for the Euclidean space.

2) Multi-layer perceptron (MLP)

The MLP is a feed-forward neural network classifier that is completely connected. The output $h^{(3)}$ of a neural network is determined by adding up the outputs of each layer in accordance with Equations (4), (5), and (6), where z x is the input vector containing the features that were picked, $W^{(i)}$ is the Weight Matrix, and $b^{(i)}$ is the Bias Vector for layer i.

$$h^{(1)} = g_1(W^{(1)T}.x + b^{(1)}) \quad (4)$$

$$h^{(2)} = g_1(W^{(2)T}.h^{(1)} + b^{(2)}) \quad (5)$$

$$\mathrm{h}^{(3)} = \mathrm{g}_2(\mathrm{W}^{(3)\mathrm{T}}.\mathrm{h}^{(2)} + \mathrm{b}^{(2)}) \quad (6)$$

There is non-linearity in the neural network because of the activation functions g_1 and g_2. Hidden layers with $\lambda = 1.0507$ and $\alpha = 1.6733$ use the scaled exponential linear unit from Equation (7) as their activation function since MLP does not automatically normalize its outputs. Every output may be understood as the likelihood of predicting a certain class thanks to the output layer's usage of the softmax activation function g2, which is specified in Equation (8). The forecasted label yˆ is given by yˆ = argmax $h^{(3)}$

$$\mathrm{g}_1(z) = \lambda.\begin{cases}\alpha.(\mathrm{e}^{\mathrm{z}} - 1) & \text{for } \mathrm{z} < 0\\ \mathrm{z} & \text{for } \mathrm{z} \geq 0\end{cases} \quad (7)$$

$$\mathrm{g}_2(z)_j = \frac{e^{z_j}}{\sum_{K=1}^{N} e^{z_j}} \text{for } j \in [1;15] \quad (8)$$

An architecture summary of the proposed MLP model consists of four dense (fully connected) layers as shows in Figure 7. This structure consists of a 128-unit dense layer at the top, a 32-unit and 64-unit dense layer below it, and finally a dropout layer. The final dense layer has 1 unit for binary classification. The dropout layers do not contribute to the parameter count and are used solely for regularization. Each layer is listed with its corresponding output shape and the number of parameters involved in training. This tabular representation helps visualize the layer-wise transformation of data as it moves through the network.

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	5,888
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2,080
dense_3 (Dense)	(None, 1)	33

Total params: 16,257 (63.50 KB)
Trainable params: 16,257 (63.50 KB)
Non-trainable params: 0 (0.00 B)

Fig. 7. MLP Model Architecture Summary

The model comprises a total of 16,257 trainable parameters, with no non-trainable components in Figure 7. These parameters are distributed across the four dense layers, with the majority concentrated in the first two layers (5,888 and 8,256 parameters, respectively), reflecting the higher dimensionality at those stages. The number of layers and neurons was chosen based on performance evaluation, aiming to strike a balance among model complexity and generalization. By randomly deactivating a portion of neurones during training, dropout is used as a regularization strategy to prevent overfitting and make sure the model doesn't become too dependent on certain network routes.

3) Logistic Regression

The statistical method that underpins the LR framework is mostly used to solve binary classification issues, where the outcome is two groups of variables. Equation (9) is utilized to determine the likelihood of an input belonging to a given class utilizing the logistic function (sigmoid function).

$$h_\theta\left(x^{(i)}\right) = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\ldots+\theta_n x_n)}} \qquad (9)$$

where $x_0, x_1,\ldots, x_n$ denote the input feature values and the related parameters, which are changed as the learning progresses, are denotes as $\theta_0 + \theta_1+\ldots+\theta_n$. These parameters are then used to return the prediction model. The probability that the input is positive is represented by the output, which may take on values ranging from 0 to 1. When a linear connection between features and binary output is assumed, LR becomes easily interpretable and efficient. These days, a lot of people use logistic regression to create predictive studies. To balance training time and convergence efficiency for moderately large datasets, the logistic regression model in this research was built up with a maximum iteration limit of 10.

I. Model Evaluation

Once a model has been trained, its performance must be thoroughly evaluated. This research evaluates the suggested system's detection capabilities using the confusion matrix, the ROC curve, and important assessment metrics including Acc, Prec, Rec, and F1score. The Confusion Matrix provides a thorough analysis of TP, FP, TN, and FN, offering valuable information on certain misclassifications that might compromise the dependability of the system. This analysis helps determine whether the system is more susceptible to false alarms or missed detections, thereby informing further model tuning and optimization. As shown in Equations (10), (11), (12), and (13), all performance metrics were systematically applied across different datasets to quantitatively demonstrate the robustness and effectiveness of the proposed model.

- **True positive rate (TPR):** The accuracy of the projected and actual class values is indicated by these favourably expected results.
- **False positive rate (FPR):** A negative attribute that has been accurately predicted is the fact that both the expected and real class values are negative.
- **False negative rate (FNR):** A situation when the anticipated class is legitimate but the actual class is false.
- **True negative rate (TNR):** An example of a class that is positive while the expected class is negative.

Accuracy (Acc): Accuracy is the proportion of relevant samples among the recovered samples, which is formulated in Equation (10).

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \quad (10)$$

Precision (Prec): This metric measures the success rate of positive observations as a percentage of all positive observations anticipated, which is formulated in Equation (11).

$$Precision = \frac{TP}{TP+FP} \quad (11)$$

Recall (Rec): The sum of all false negatives and true positives is divided by this total to get the true positive rate, which can be quantitatively calculated as, which is formulated in Equation (12).

$$Recall = \frac{TP}{TP+FN} \quad (12)$$

F1-score: The F1score is calculated by averaging the Prec and Rec. Accuracy, as measured by the system's performance relative to the projected data, is mirrored by precision. r is the ratio of the total amount of predicted data to all anticipated data for prediction and is formulated in Equation (13).

$$F1\ Score = \frac{2\ (Precision \times Recall)}{Precision \times Recall} \quad (13)$$

ROC Curve: The appropriate decision boundary may also be found by using the ROC curve, which shows the trade-off among the FPR and the TPR (recall) at different classification thresholds. The entire performance is measured by the AUC; a higher AUC indicates greater discriminating between attack and normal cases.

IV. Result Analysis and Discussion

The performance of propose models with graphically and theoretical descriptions is discussed in this section. The experiments were conducted in a Python environment on the Google Colab platform, utilizing key machine learning libraries such as PyTorch and Scikit-learn. The setup employed an 8 GB NVIDIA T4 GPU for accelerated computation. The local development machine ran on Windows 10, powered by a 12th Gen Intel Core i5-1235U processor with 12 GB of RAM. As shown in Table II, all evaluated models, KNN, MLP, and Logistic Regression demonstrated strong performance. Among them, the KNN model achieved a best outcomes, with an Acc of 99.57%, Prec of 99.80%, Rec of 99.55%, and an F1score of 99.67%, making it the most suitable for integration into the AI-enhanced DevSecOps pipeline.

TABLE II. Model Performance in DevSecOps Pipeline for Security on TII-SSRC-23 Dataset

Matrix	KNN	MLP	LR
Accuracy	99.57	99.30	96.60
Precision	99.80	99.80	97.74
Recall	99.55	99.15	97.15
F1 Score	99.67	99.47	97.44

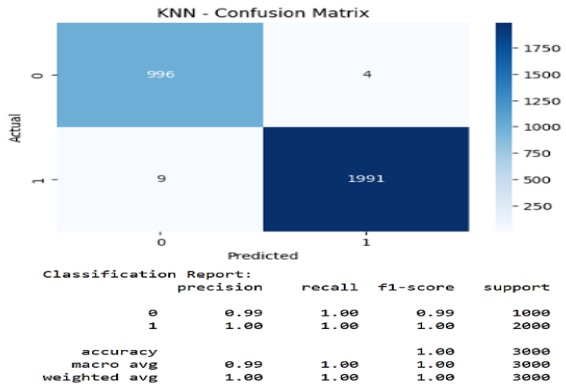


Fig. 8. Classification Report and Confusion Matrix of KNN

The KNN classification model's ability in accurately differentiating between two classes is shown in Figure 8. Class '1' has perfect scores while class '0' has somewhat lesser precision, according to the classification report, which also displays good Prec, Rec, and f1scores. The Confusion Matrix indicates that 996 out of 1000 instances of class '0' were correctly classified, with only 4 misclassified as class '1'. Also, out of 2000 occurrences of class '1', 1991 were properly predicted, while 9 were incorrectly categorized as class '0'. These outcomes show that the model performed well on the dataset, with few mistakes and good consistency across measures.

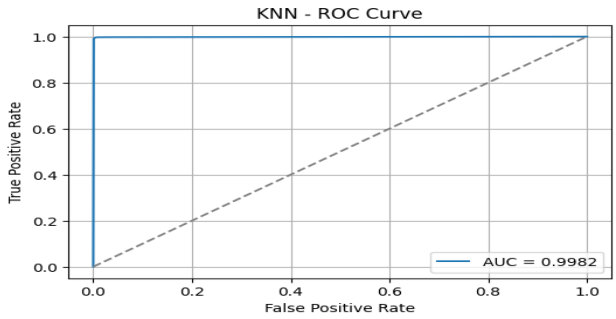
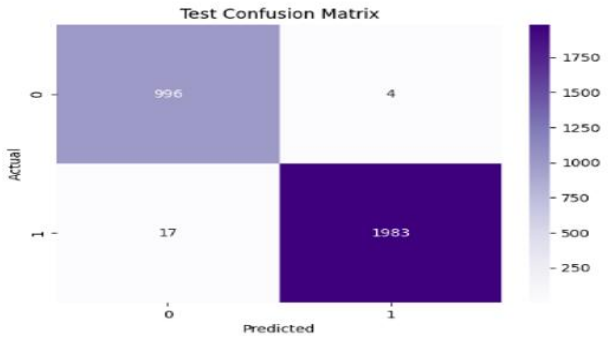


Fig. 9. ROC-AUC Curve of KNN

Figure 9 presents a ROC curve for a KNN model. The FPR is shown on an x-axis, which goes from 0.0 to 1.0, while the TPR is shown on a y-axis. A solid blue line depicts the ROC curve, which remains close to the top-left corner of the plot, where both the FP rate is low and the TP rate is high. A dashed gray line represents the random classifier baseline. The legend in the bottom right corner shows an AUC value of 0.9982. The curve demonstrates that the model maintains a low FP rate while achieving a high true positive rate across various thresholds.



Classification Report (Test):

	precision	recall	f1-score	support
0	0.98	1.00	0.99	1000
1	1.00	0.99	0.99	2000
accuracy			0.99	3000
macro avg	0.99	0.99	0.99	3000
weighted avg	0.99	0.99	0.99	3000

Fig. 10. Classification Report and Confusion Matrix of MLP

Figure 10 presents the classifications made by the MLP model on the test dataset. The Confusion Matrix shows that there were fifteen cases of class '1' being wrongly tagged as class '0' and four cases of class '0' being wrongly projected as class '1'. These low error values indicate minimal confusion between the two classes. The classification report reflects this, showing a Rec of 0.99 for class '1' and a precision of 0.98 for class '0'—the only values below 1.00. The heatmap uses lighter shades of purple to highlight these low misclassification counts, offering a clear view of the small deviations in the MLP model's predictions compared to the true labels.

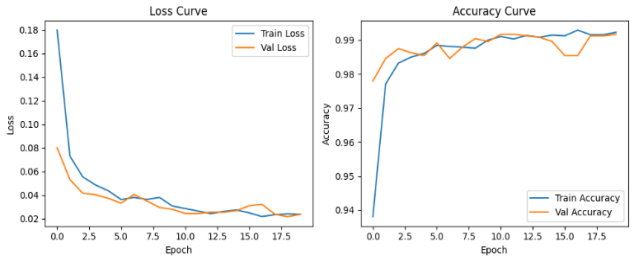
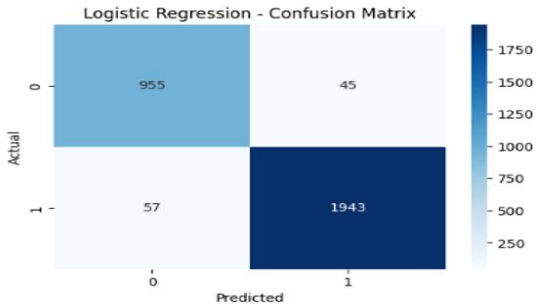


Fig. 11. Accuracy and Loss curve of MLP

Figure 11 illustrates the training progress of a MLP model, with both loss and accuracy curves indicating effective learning and strong generalization. There are only little variations in the validation loss towards the end, with the training and validation losses sharply declining in the early epochs and stabilizing at low levels. The training and validation accuracies increase rapidly and level off around 0.99. These results indicate that the MLP model learns efficiently, achieves high performance on both training and validation data, and maintains consistent behavior throughout training, suggesting its reliability and suitability for practical applications.



Classification Report:

	precision	recall	f1-score	support
0	0.94	0.95	0.95	1000
1	0.98	0.97	0.97	2000
accuracy			0.97	3000
macro avg	0.96	0.96	0.96	3000
weighted avg	0.97	0.97	0.97	3000

Fig. 12. Classification Report and Confusion Matrix of LR

Figure 12 shows the performance of a LR model, highlighting its results for class '1', which achieved the highest metrics. Class '1' had a Prec of 0.98, Rec of 0.97, and an f1score of 0.97, based on 2000 samples. The confusion matrix shows that 1943 instances of actual class '1' were correctly predicted, while only 57 were misclassified as class '0'. These results are visually represented in the heatmap, where the darker blue shade corresponds to the high count of correct predictions for class '1'.

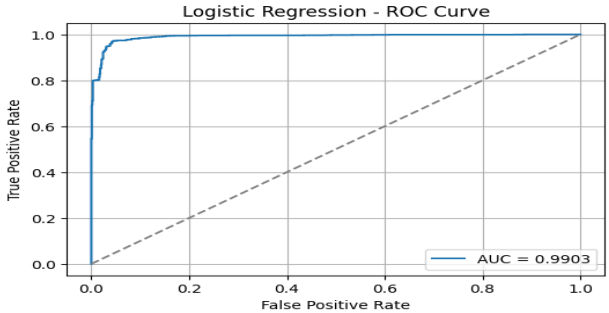


Fig. 13. ROC-AUC Curve of LR

Figure 13 shows the ROC curve for a LR model, with the TPR plotted against the FPR. The solid blue line rises sharply toward the top-left corner, indicating strong classification performance. Random classifiers are shown by dashed grey lines. High accuracy in discriminating among positive and negative classes is shown by the model's AUC of 0.9903.

A. CI/CD Pipeline

The CI/CD pipeline integrates AI-driven security into automated database deployment. It starts with data preprocessing, followed by model training and validation. The best-performing model is saved and deployed, with automated security scans embedded into the workflow. Continuous monitoring ensures updates when threats or data drift are detected, enabling secure, adaptive, and efficient DevSecOps operations.

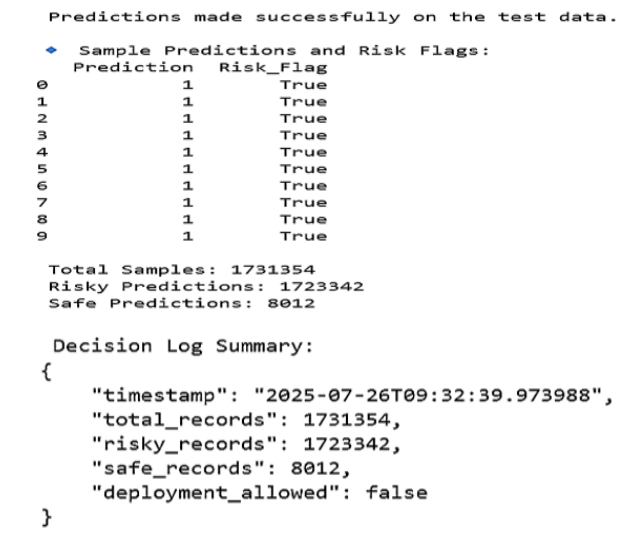


Fig. 14. Prediction Results and Deployment Decision Summary of Cyber Threat Detection System

Figure 14 displays the output of the cyber threat detection system integrated within the deployment pipeline. The first part displays the prediction results, where out of 1,731,354 total samples, 1,723,342 were classified as risky and only 8,012 as safe, clearly highlighting the system's ability to accurately identify threats. The second part shows the corresponding Decision Log Summary, where the automated analysis timestamped for traceability resulted in "deployment_allowed": false, thereby blocking the deployment. This seamless integration ensures that potentially harmful code is proactively stopped before reaching production, maintaining both the security and integrity of the software pipeline.

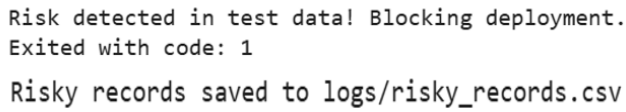


Fig. 15. Deployment Block Alert and Risk Logging

Figure 15 illustrates the system's automated action within the DevSecOps pipeline, where deployment is halted after detecting risks in test data, as shown by the exit message, and all identified risky records are stored in logs/risky_records.csv for further review. This prevents potentially vulnerable builds from being deployed, strengthening the overall security of the software delivery process and promoting a proactive approach to managing cyber threats.

B. Comparison and Discussion

Table III presents a comparative evaluation of machine learning models used within the AI-enhanced DevSecOps pipeline. The suggested KNN and MLP models outperformed the competition, with KNN reaching a peak Acc of 99.57%. This reflects its strong capability in handling well-structured data with clear class boundaries, making it effective for tasks such as anomaly detection and access control verification in secure pipelines. MLP, with slightly lower but still exceptional accuracy, performs well in capturing complex, non-linear patterns, making it suitable for dynamic deployment environments. Models like LR and DT are straightforward and interpretable but tend to underperform in scenarios involving complex data distributions. J48 achieves moderate accuracy but struggles with scalability as dataset complexity increases. Earlier implementations of KNN produce reasonable results but often involve higher computational costs. Naïve Bayes delivers fast results but is less effective when features are interdependent, which is common in DevSecOps data. LSTM, while effective in handling sequence-

based inputs, does not perform optimally in this static classification task. Overall, the proposed models outperform existing methods due to their adaptability, precision, and reliability, making them highly suitable for secure and automated operations in the DevSecOps pipeline.

TABLE III. PERFORMANCE COMPARISON OF ML MODELS IN THE AI-ENHANCED DEVSECOPS PIPELINE

Matrix	Accuracy	Precision	Recall	F1 Score
KNN	99.57	99.80	99.55	99.67
MLP	99.30	99.80	99.15	99.47
LR	96.60	97.74	97.15	97.44
DT	96	96	98	97
J48	95.65	95.70	95.70	95.70
KNN	95.04	95.04	95.04	95.05
NB	94.2	94.8	90	92.4
LSTM	90.9	90.4	90.2	90.3

The proposed AI-enhanced DevSecOps pipeline holds significant importance as it seamlessly integrates automated security into database deployment, ensuring real-time protection against threats while maintaining operational efficiency. By leveraging advanced ML models like KNN, LR and MLP, the pipeline achieves superior accuracy, precision, and recall compared to traditional methods, thereby reducing FP and enhancing trust in automated decision-making. Its ability to preprocess large-scale datasets, detect anomalies with near-perfect accuracy, and block risky deployments proactively strengthens the overall resilience of software delivery processes. Moreover, continuous monitoring and adaptive retraining allow the system to handle evolving threats, minimize human intervention, and maintain compliance within secure pipelines. This makes the proposed approach highly advantageous for enterprises seeking scalable, reliable, and secure DevSecOps practices.

V. CONCLUSION AND FUTURE SCOPE

Automating security within DevOps pipelines is vital for addressing the growing sophistication of cyber threats. DevSecOps promotes security as an integral part of the development lifecycle, leveraging tools to improve security posture without sacrificing speed or efficiency. The integration of AI into DevSecOps has further enhanced the capability to detect, respond to, and mitigate threats proactively. In this study, an AI-enhanced DevSecOps framework was developed to secure automated database deployments by embedding machine learning into the CI/CD pipeline. Multiple models, including KNN, MLP, and LR, were trained and evaluated to determine the most effective solution. KNN achieved the highest performance, with 99.57% Acc and 99.67% F1score, showcasing its superior threat detection capability. The best-performing model was seamlessly integrated into the DevSecOps workflow, enabling real-time intrusion detection while preserving development speed and agility. However, some limitations remain: relying on a single data source may restrict generalizability, and the simulation environment may not fully capture the dynamic nature of real-world deployment pipelines. Future research will aim to validate this framework in diverse operational contexts and extend its application across cloud-native and large-scale production environments. Future work will focus on deploying the proposed AI-enhanced DevSecOps framework in real-time, cloud-native environments to assess scalability and resilience.

REFERENCES

[1] M. Castellaro, S. Romaniz, J. C. Ramos, C. Feck, and I. Gaspoz, “Aplicar el Modelo de Amenazas para incluir la Seguridad en el Modelado de Sistemas,” *Proc. V Congr. Iberoam. Segur. Informática—CIBSI*, 2016.

[2] Y. Valdés-Rodríguez, J. Hochstetter-Diez, J. Díaz-Arancibia, and R. Cadena-Martínez, “Towards the Integration of Security Practices in Agile Software Development: A Systematic Mapping Review,” *Appl.*

Sci., 2023, doi: 10.3390/app13074578.

[3] A. Goyal, "Driving Continuous Improvement in Engineering Projects with AI-Enhanced Agile Testing and Machine Learning," *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 3, no. 3, pp. 1320–1331, Jul. 2023, doi: 10.48175/IJARSCT-14000T.

[4] A. Gupta, "Comparative Study of Different SDLC Models," *Int. J. Res. Appl. Sci. Eng. Technol.*, 2021, doi: 10.22214/ijraset.2021.38736.

[5] D. Patel, "Zero Trust and DevSecOps in Cloud-Native Environments with Security Frameworks and Best Practices," *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 3, no. 3, 2023.

[6] I. K. Aksakalli, T. Celik, A. B. Can, and B. Tekinerdogan, "A Model-Driven Architecture for Automated Deployment of Microservices," *Appl. Sci.*, vol. 11, no. 20, 2021, doi: 10.3390/app11209617.

[7] G. Modalavalasa, "The Role of DevOps in Streamlining Software Delivery: Key Practices for Seamless CI/CD," *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 1, no. 12, pp. 258–267, Jan. 2021, doi: 10.48175/IJARSCT-8978C.

[8] S. S. S. Neeli, "Optimizing Database Management with DevOps: Strategies and Real-World Examples," *J. Adv. Dev. Res.*, vol. 11, no. 1, 2020.

[9] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, "Challenges and solutions when adopting DevSecOps: A systematic review," *Inf. Softw. Technol.*, vol. 141, Jan. 2022, doi: 10.1016/j.infsof.2021.106700.

[10] D. Herzalla, W. T. Lunardi, and M. Andreoni, "TII-SSRC-23 Dataset: Typological Exploration of Diverse Traffic Patterns for Intrusion Detection," *IEEE Access*, 2023, doi: 10.1109/ACCESS.2023.3319213.

[11] X. Ramaj, M. Sánchez-Gordón, V. Gkioulos, S. Chockalingam, and R. Colomo-Palacios, "Holding on to Compliance While Adopting DevSecOps: An SLR," *Electronics (Switzerland)*. 2022. doi: 10.3390/electronics11223707.

[12] A. Bahaa, A. Abdelaziz, A. Sayed, L. Elfangary, and H. Fahmy, "Monitoring real time security attacks for iot systems using devsecops: A systematic literature review," *Information (Switzerland)*. 2021. doi: 10.3390/info12040154.

[13] G. Liu, H. Zhao, F. Fan, G. Liu, Q. Xu, and S. Nazir, "An enhanced intrusion detection model based on improved kNN in WSNs," *Sensors*, vol. 22, no. 4, p. 1407, 2022.

[14] L. Wang, "Research and Implementation of Machine Learning Classifier Based on KNN," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 677, no. 5, pp. 1–5, Dec. 2019, doi: 10.1088/1757-899X/677/5/052038.

[15] A. Rosay, K. Riou, F. Carlier, and P. Leroux, "Multi-layer perceptron for network intrusion detection: From a study on two recent data sets to deployment on automotive processor," *Ann. des Telecommun. Telecommun.*, 2022, doi: 10.1007/s12243-021-00852-0.

[16] R. Patel, "Automated Threat Detection and Risk Mitigation for ICS (Industrial Control Systems) Employing Deep Learning in Cybersecurity Defence," *Int. J. Curr. Eng. Technol.*, vol. 13, no. 6, 2023.