National College of Ireland

Automating Threat Intelligence: The Use of AI in Cyber Threat Prediction and Mitigation

MSc Research Project
MSc Cybersecurity

Aniket Vishwakarma
Student ID: x23211768

National College of Ireland

Supervisor: Prof Jawad Salahuddin

National College of Ireland

MSc Project Submission Sheet

Student Name:	Aniket Vishwakarma
Student ID:	x23211768
Programme:	MSc Cybersecurity **Year: 2025**
Module:	Research Practicum
Supervisor:	Professor Jawad Salahuddin
Submission Due Date:	15th September 2025
Project Title:	Automating Threat Intelligence: The Use of AI in Cyber Threat Prediction and Mitigation
Word Count:	9665 **Page Count:** 25

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Aniket Vishwakarma

Date: 15th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Automating Threat Intelligence: The Use of AI in Cyber Threat Prediction and Mitigation

Aniket Vishwakarma
x23211768

Abstract

The proliferation of digital services and the increasing sophistication of cyber attacks have rendered traditional, signature-based security systems insufficient for modern threat landscapes. This research project addresses the critical need for proactive and automated threat intelligence by leveraging Artificial Intelligence (AI) and Machine Learning (ML). The core objective is to design, implement, and evaluate an end-to-end system capable of predicting and classifying diverse cyber threats from network traffic data in near real time. This study utilizes a comprehensive dataset featuring various network attacks, including Denial of Service (DoS), Distributed Denial of Service (DDoS), Port Scanning, and Web Attacks. A rigorous data preprocessing, feature engineering, and dimensionality reduction pipeline was developed to prepare the data for modelling. Eight distinct machine learning classifiers were benchmarked, with the Random Forest algorithm emerging as the most effective, achieving an F1-score of 0.9756 on unseen test data. The research extends beyond model creation to encompass the full lifecycle of a production-grade AI system. The final model was encapsulated within a Flask web application, offering single, batch, and API-based prediction capabilities. The application was eventually deployed to a cloud environment on Amazon Web Services (AWS) through a fully automated Continuous Integration and Continuous Deployment (CI/CD) workflows that were managed by GitHub Actions. The results show that an AI driven approach can offer accurate automated classification of threats, which can potentially greatly increase the capability of an organization to be proactive in its mitigation of cyber threats. The deployed system serves as a real-world example of a way to incorporate cutting edge AI into a proactive cyber security environment.

1 Introduction

The digital transformation of modern society has created an unprecedented reliance on interconnected systems, making cybersecurity a paramount concern for individuals, corporations, and governments alike. The threat landscape is evolving at a staggering rate, with adversaries employing increasingly sophisticated and automated techniques to compromise systems, steal data, and disrupt services (Bécue, Praça and Gama; 2021). Old- school preventive measures, which often base their detection of threats on predefined signatures and rule sets, fail to detect the vast quantity and variety of modern-day attacks. Hence, because this posture only allows critical infrastructure to react to identified and analyzed threats, often too slowly to prevent significant damage, it keeps critical infrastructure open to threats (Sarker; 2023).

In response to these challenges, a field known as Cyber Threat Intelligence (CTI) has emerged. CTI involves the collection, analysis, and sharing of information regarding current and future attacks. Manual CTI analysis, however, is labor intensive, requiring much time and very qualified human experts (Rahman, Hezaveh and Williams; 2023). The total explosion of data created by network devices and the corresponding security logs has created a human analyst situation where they are overwhelmed, leaving a gap between threat emergence and detection. This difference indicates the urgent need for automation in the entire CTI lifecycle.

AI in conjunction with Machine Learning (ML) gives practical solutions to this situation. However, it is designed to train most of the advanced models on enormous historical network data so that AI systems learn to discover the subtle patterns and anomalous behaviors that usually characterize malicious activity (Khan, Arif and Khan; 2024). These systems are able to operate at the speeds of machines and at their scales such that they can absorb immense data streams through actual predictive modeling that can predict incidents before they escalate (Abisoye et al.; 2025). By applying AI technology to cybersecurity, it is now becoming possible to move from a reactive defense toward proactive and predictive measures so that organizations can mitigate risks more effectively through anticipation (Sarfraz et al.; 2025).

However, despite all the promises of AI, its actual use in the operation environment of security poses big challenges. Developing a good model for threat detection is only the very first step. This model must be integrated into a robust and scalable automated solution capable of ingesting data, making predictions, and timely presenting actionable intelligence to security teams. This involves a very intricate interplay of data engineering, model deployment, cloud infrastructure management, and software development.

This research project aims to address these issues by developing a complete, end to end solution for automating threat intelligence using AI. The central research question guiding this study is: **How can a machine learning pipeline be effectively designed, implemented, and deployed to automatically classify diverse cyber threats from network traffic data with high accuracy?**

To answer this question, the following research objectives were established:

1. To conduct a comprehensive analysis of a multifaceted network traffic dataset containing both benign and malicious flows, and to perform rigorous data preprocessing and feature engineering to prepare it for machine learning.
2. To systematically evaluate and benchmark a suite of diverse machine learning algorithms to identify the most effective model for multi class cyber threat classification.
3. To develop a deployable software artifact, in the form of a web application, that encapsulates the trained model and provides an accessible interface for threat prediction.
4. To implement a fully automated CI/CD pipeline for deploying the application to a scalable cloud infrastructure, demonstrating a practical pathway for operationalizing AI in cybersecurity.

The contribution of this work is threefold. First, it provides a detailed, replicable methodology for building a high performance threat classification model. Second, it presents

a complete system architecture that bridges the gap between theoretical model development and practical, real-world application. Third, it serves as a comprehensive case study on leveraging modern DevOps and cloud computing practices to automate the deployment and maintenance of an AI driven security tool.

This report is structured as follows. Section 2 provides a critical review of the existing literature in AI for cybersecurity and automated threat intelligence. Section 3 details the research methodology developed to follow a quantitative and experiment based approach. Section 4 provides the design specification for both the pipeline of the machine-learned model and the system architecture. Section 5 provides detail on implementation of the project including the tools and technologies used. Section 6 includes a full evaluation of results which encompasses model performance and system features. Lastly, Section 7 provides a conclusion to the report and summarizes the key findings, highlights limitations of the study, and suggests directions for future work.

2 Related Work

This section situates the current research within the broader academic and technical landscape of AI in cybersecurity. It provides a critical review of existing literature, focusing on the evolution of threat detection, the role of machine learning in automating cyber threat intelligence, and the challenges associated with deploying such systems. The review is structured to highlight the gaps in existing research that this project aims to address.

2.1 The Evolution from Traditional to AI-Powered Threat Detection

For decades, the primary approach to cybersecurity has been based on signature detection. Intrusion Detection Systems (IDS) and antivirus software have traditionally relied on databases of known malware signatures, malicious IP addresses, and attack patterns. While effective against known threats, this approach has a fundamental weakness: it is incapable of detecting novel or zero day attacks for which no signature exists (Bécue, Praça and Gama; 2021). As attackers develop new techniques and malware variants at an accelerated pace, the window of vulnerability between the emergence of a new threat and the development of its signature widens, leaving systems exposed.

The limitations of signature based methods have driven the research community towards more dynamic, behavior based approaches. Anomaly detection, which focuses on identifying deviations from a baseline of normal system or network behavior, represents a significant step forward. However, early anomaly detection systems were often plagued by high rates of false positives, as any legitimate but unusual activity could be flagged as malicious, leading to alert fatigue among security analysts (Sarker; 2023).

The advent of advanced machine learning has provided a powerful new toolkit for overcoming these challenges. ML models can analyze vast and complex datasets to learn the nuanced characteristics that differentiate benign traffic from a wide array of malicious activities (Ejeofobiri et al.; 2024). Unlike static rule-based systems, ML models can generalize from the data they are trained on, enabling them to identify previously unseen attacks that share characteristics with known threat families. A study by Khan, Rahman, and

Sumon (2023) demonstrated that AI based systems could significantly improve the detection rates of sophisticated attacks in the United States compared to traditional methods. This ability to move from simple pattern matching to complex behavioral analysis is at the heart of the modern, AI driven approach to cybersecurity.

2.2 Machine Learning Techniques in Cyber Threat Intelligence

A wide variety of machine learning algorithms have been applied to the problem of cyber threat detection. These can be broadly categorized into supervised, unsupervised, and semi supervised learning approaches.

Supervised learning is the most common approach, where a model is trained on a labeled dataset containing examples of both normal and malicious activity. This thesis embraced the approach outlined in the current research. A great number of the studies carrying out the application of supervised classifiers into this domain have been successful. For instance, baseline models for intrusion detection have been Logistic regression, Support Vector Machines, and Decision Trees as classic algorithms (Maddireddy and Maddireddy; 2021). Yet more sophisticated ensemble methods offer better performance and include Random Forests and Gradient Boosted Trees, consistently outclassing their predecessors. They use the predictions of multiple individual models to make a better and more robust classification and thus minimize overfitting and improve generalization (Kavitha and Thejas; 2024). Thus, ensemble models are potent when there are complex, and non-linear relationships buried somewhere into the data, which are often the trademarks of sophisticated cyber attacks.

Where labeled data is short or not present, unsupervised learning procedures come into play. For similar network flows, K-Means Clustering or the Principal Component Analysis (PCA) can be considered, assuming that malicious traffic will be separated and clustered away from normal traffic. Sun et al. (2023) provide a survey on how such data mining techniques can be used for proactive defense. While excellent for discovering new threats, methods for clustering are often hard to interpret and demand much manual effort to label the resultant clusters as benign or malicious.

The generation and usage process of threat intelligence has also come to be an area of focus for automation. Cyber Threat Intelligence (CTI) mining is aimed at automatically extracting actionable information from unstructured sources, such as security blogs, forums, and discussions in the Dark Web (Rahman, Hezaveh, and Williams; 2023).

Alturkistani and Chuprat (2024) explore how Large Language Models (LLMs) are further advancing this field by enabling more sophisticated analysis of textual data. This complements network traffic analysis by providing context about attacker tactics, techniques, and procedures (TTPs). While the current research focuses on structured network data, the integration with automated CTI mining represents a key area for future work.

2.3 Feature Engineering and Dimensionality Reduction

The performance of any machine learning model is heavily dependent on the quality of the features used for training. In the context of network security, raw packet data is often too granular and high dimensional to be used directly. Consequently, feature engineering, the process of creating informative features from raw data, is a critical step. Most cybersecurity datasets, including the one used in this research, are based on flow level features. These

features summarize the statistical properties of a network communication session over a period of time, such as flow duration, packet counts, packet sizes, and inter arrival times (Sarker; 2023).

However, even with flow-based features, datasets can contain dozens or even hundreds of attributes. Many of these features may be redundant or irrelevant, a phenomenon known as the "curse of dimensionality." This can lead to increased computational cost, model complexity, and a higher risk of overfitting. Dimensionality reduction techniques are therefore essential. Principal Component Analysis (PCA) is a widely used method that transforms the original features into a smaller set of uncorrelated components while retaining most of the original information (variance). The use of PCA not only makes the training process more efficient but can also improve model performance by filtering out noise (Maddireddy and Maddireddy; 2021). This research explicitly incorporates PCA into its methodology to address these issues.

2.4 Challenges and Gaps in Existing Research

While the literature is rich with studies proposing various ML models for threat detection, a significant gap often exists between academic research and practical, real world deployment. Many studies conclude after reporting a high accuracy score on a static dataset, without addressing the engineering challenges of operationalizing their model (Lahare and Wakchaure; 2025). As noted by Santos et al. (2025), the effectiveness of CTI depends on technologies, strategies, and collaborations. A standalone model, no matter how accurate, has limited value if it cannot be integrated into an organization's existing security workflow.

This project directly addresses this gap by focusing on the entire end to end lifecycle of an AI driven security solution. The work encompasses not only the development and rigorous evaluation of a high performance model but also its packaging into a usable software application and its deployment onto a scalable cloud infrastructure using modern automation practices. Whitman (2024) emphasizes the need for comprehensive reviews and future directions in CTI mining. This research provides a practical blueprint for one such direction: the automated deployment of a prediction service. The creation of a fully automated CI/CD pipeline using GitHub Actions is a key contribution, as it demonstrates how to maintain and update the AI model in a production environment with minimal manual intervention, a topic that is often overlooked in academic literature but is critical for long term viability (Sontan and Samuel; 2024).

Table 1: Comparison of Existing Studies in AI-driven Cyber Threat Detection

Study	Year	Dataset Used	Models / Techniques	Key Results	Limitations
Bécue, Praça & Gama	2021	Conceptual review (Industry 4.0)	Signature-based vs AI approaches	Highlighted challenges/opportunities of AI in cybersecurity	Lacks empirical evaluation, mainly conceptual
Maddireddy & Maddireddy	2021	CIC-IDS2017	Logistic Regression, SVM, Decision Trees	Demonstrated predictive modelling improves over signature methods	Focused on classical ML; limited deployment aspects

Khan, Rahman & Sumon	2023	U.S. cybersecurity datasets	Ensemble ML models	Showed AI improves sophisticated attack detection	U.S.-specific case; limited generalizability
Sun et al.	2023	Synthetic network flow data	K-Means, PCA (unsupervised)	Useful for discovering novel anomalies	Requires manual labeling of clusters; interpretability issues
Rahman, Hezaveh & Williams	2023	Security blogs, forums, dark web text	Automated CTI extraction (NLP)	Automated threat intelligence extraction feasible	Focused on text mining, not traffic data
Khan, Arif & Khan	2024	Review of multiple datasets	Overview of AI techniques	Summarized ML/AI methods for cyber defense	Survey only; lacks experimental validation
Kavitha & Thejas	2024	NSL-KDD, CIC-IDS2017	Random Forest, Gradient Boosted Trees	Ensemble methods outperform baseline models	Does not address deployment challenges
Alturkistani & Chuprat	2024	CTI text sources	Large Language Models (LLMs)	LLMs enhance CTI mining from unstructured data	No structured traffic analysis; still emerging
Whitman	2024	Literature review	CTI mining methods	Identified future research directions	Conceptual, no implemented system
Lahare & Wakchaure	2025	Literature + case studies	Automated ML-based CTI	Reviewed trends in proactive defense	No end-to-end deployment demonstration
Santos et al.	2025	Systematic review	Technologies & strategies in CTI	Collaboration + automation improve effectiveness	No experimental model benchmarking
Abisoye et al.	2025	Critical infrastructure datasets	AI/ML for risk prediction	AI-driven mitigation enhances resilience	Domain-specific; not generalizable across all sectors

3 Research Methodology

This study uses a quantitative, experimental method to design, develop, and evaluate an automated cyber threat prediction system. The method follows the Cross Industry Standard Process for Data Mining (CRISP-DM) model, one of the most commonly used data science project frameworks. CRISP-DM allows research to occur in a stepwise, systematic, and replicable manner that includes business understanding, data understanding, data preparation, modeling, evaluation, and deployment. The entire research pipeline includes separate, sequential phases, with each phase being based on the output from the previous phase.

3.1 Research Paradigm

The study's research paradigm is based on positivism, which accepts knowledge through empirical observation and logic. The study is classified as experimental with respect to the research design because data were manipulated and machine learning frameworks were deliberated through rigorous covariation of variables in a controlled setup as a means of testing an experimental hypothesis. The central hypothesis was that a trained machine learning model of multiple cyber threats from network traffic data could be classified reliably, deploying it as an automated service. The approach was quantitative, utilizing numerical data and performance outcomes with statistics for validation.

3.2 Data Collection and Dataset

The foundation of this research is a publicly available, pre existing dataset. The dataset chosen is a variant of the well-known CIC - IDS2017 dataset, but it has been pre-processed and balanced, ensuring it could be used for multi-class classification. It is vital that research utilizes a dataset that is known and validated in order to ensure reproducibility and comparability of the results. The dataset, titled balanced_dataset.csv contains a collection of network flow data, where each row corresponds to a bidirectional communication session between two endpoints and defined by a suite of statistical features.

It was also notable that the balanced_dataset.csv included 58,610 records and 78 original features. Each record was provided a label corresponding to one of twelve different classes; including one "Benign" class and eleven separate attack classes such as "DDoS", "DoS Hulk", "PortScan", "Bot", and "Web Attack". This multi class nature is a key aspect of the research, as it allows for the development of a model that can not only detect a threat but also identify its specific type, providing more actionable intelligence for security teams (Upadhyay and Jain; 2024). The features are derived from network traffic captured using tools like Wireshark and CICFlowMeter, and they describe various temporal and spatial characteristics of the network flows, such as duration, packet counts, packet lengths, inter arrival times, and TCP flag counts.

3.3 Data Preparation and Preprocessing

Raw data is rarely suitable for direct input into machine learning models. Therefore, a rigorous data preparation phase was conducted. This phase involved several key steps:

1. **Data Loading and Initial Inspection:** The data was imported into a pandas DataFrame and an initial inspection was made to understand its shape, types and descriptive statistics. This phase helped confirm the shape of the data and assess the target variable distribution.

2. **Handling Missing and Infinite Values:** The data was inspected for missing (NaN) values and infinite (Inf) values. The initial missing data check identified no missing values. However, a robust pipeline was created to handle missing values in case there were any, this also included code the check for Inf values, which can occur when calculating features feature such as Flow Bytes/s when a divide-by-zero operation occurs. The strategy implemented was to replace all Inf values with the median of the corresponding columns; this is a robust approach since it is not affected by outliers.
3. **Duplicate Removal:** The data was inspected for duplicate rows. A small number of duplicates were found and removed. This helps prevent the model from overfitting to ambiguous/duplicate information.
4. **Zero Variance Feature Removal:** Features that have a single constant value across all samples provide no discriminatory information and can be safely removed. A check for zero variance columns was performed, and several such columns (e.g., Bwd PSH Flags, Fwd URG Flags) were identified and dropped from the dataset. This simplifies the model and reduces computational overhead.

3.4 Feature Engineering and Selection

This phase focused on enhancing the predictive power of the dataset by creating new features and selecting the most relevant ones.

1. **Feature Selection:** Not all of the initial 78 features are equally important for threat detection. Based on domain knowledge and common practices in network security research, a subset of 27 highly relevant features was initially selected. These features included core flow characteristics like Flow Duration, packet counts, header lengths, and key flag counts, which are known to be strong indicators of malicious activity (Sarker; 2023).
2. **Feature Engineering:** To capture more complex relationships within the data, eight new features were engineered from the selected base features. These new features were designed to represent ratios, efficiencies, and behavioral patterns. For example, Fwd_Bwd_Packet_Ratio was created to capture the asymmetry of communication, a common trait in some attacks. Total_Flags and Suspicious_Flag_Ratio were created to summarize TCP flag behavior, which is often indicative of scanning or connection manipulation attacks. This process expanded the feature set to 35, providing richer information for the models.

3.5 Dimensionality Reduction and Scaling

To prepare the final feature set for modeling, two additional steps were taken:

1. **Feature Scaling:** Machine learning algorithms, particularly those based on distance calculations like SVM and K-Nearest Neighbors, are sensitive to the scale of input features. To address this, the StandardScaler from the Scikit-learn library was used. This scaler standardizes features by removing the mean and scaling to unit variance, ensuring that all features contribute equally to the model's learning process.
2. **Principal Component Analysis (PCA):** To combat the curse of dimensionality and reduce computational complexity, PCA was applied. The PCA was configured to retain

95% of the variance in the data. This technique transforms the 35 engineered features into a smaller number of orthogonal components, capturing the most significant patterns in the data while filtering out noise. The result was a reduced- dimension dataset that is more efficient to process without significant loss of information.

3.6 Model Training and Evaluation

The core of the experimental phase involved training, evaluating, and comparing a suite of machine learning models.

1. **Data Splitting:** The dataset was split into a training set (80%) and a testing set (20%). A stratified splitting strategy was used to ensure that the proportion of each class in the target variable was preserved in both the training and testing sets.
2. **Model Selection:** A diverse set of eight supervised learning models was chosen for benchmarking. This included:
    - **Linear Model:** Logistic Regression.
    - **Distance-Based Models:** K-Nearest Neighbors (KNN), Support Vector Machine (SVM).
    - **Probabilistic Model:** Gaussian Naive Bayes.
    - **Tree-Based Ensemble Models:** Decision Tree, Random Forest, Gradient Boosting.
    - **Neural Network:** Multi-layer Perceptron (MLP).
    A wide range of hyperparameter options leads to a full comparison across algorithmic families.
3. **Cross-Validation:** In order to provide a good estimate of each model's performance and avoid the chance of a single lucky or unlucky train-test split, 5-fold stratified cross-validation was performed on the training data. Each model's mean accuracy and standard deviation across the five training folds were recorded.
4. **Evaluation Metrics:** The performance of the models was evaluated using a standard set of classification metrics:
    - **Accuracy:** The proportion of correctly classified instances.
    - **Precision:** The ability of the model not to label a benign instance as malicious (low false positives).
    - **Recall (Sensitivity):** The ability of the model to find all malicious instances (low false negatives).
    - **F1-Score:** The harmonic mean of precision and recall, providing a balanced measure of performance.
    - **Confusion Matrix:** A table visualizing the performance of the model, showing the counts of true positives, true negatives, false positives, and false negatives for each class.
    - **Classification Report:** A textual report showing the main classification metrics on a per-class basis. The F1-score was chosen as the primary metric for selecting the best model, as it provides a more robust measure than accuracy, especially in multi class scenarios.

3.7 System Deployment and Validation

The final phase of the methodology involved deploying the best performing model as a functional web service.

1. **Model Serialization:** The best model (Random Forest), along with the StandardScaler, PCA model, and LabelEncoder, were serialized and saved to disk using the joblib library. This ensures that the exact same preprocessing and prediction pipeline can be replicated in a production environment.
2. **Web Application Development:** A web application was built using the Flask framework in Python. The application uniquely integrated three essential interfaces, cardinally defined as a manual input form, a batch CSV upload feature, and a RESTful API endpoint, which provided an opportunity for programmatic access.
3. **Cloud Infrastructure Provisioning:** An infrastructure using Amazon Web Services (AWS) was provisioned using the AWS Command Line Interface (CLI). The provisioning included a Virtual Private Cloud (VPC), subnet, Internet Gateway, security groups, and an EC2 instance.
4. **Automated Deployment (CI/CD):** A CI/CD pipeline was developed using GitHub Actions. This GitHub Actions pipeline holds the capability to automatically deploy the Flask application to the EC2 instance whenever changes are pushed to the main branch of the source code repository.

This comprehensive methodology helped ensure that this research evaluated machine learning models academically and practically validated the deployment of a real-world AI- driven security tool.

4 Design Specification

In this section, we described the architectural design and technical details of the complete system and technical details from the pipeline for machine learning up to the cloud environment running the deployable system of deployment. The design is modular, meaning that each part is captured within its own boundaries, intends a specific operation, and can be developed and maintained in isolation of other modules.

4.1 Machine Learning Pipeline Design

The machine learning pipeline is designed as a sequential flow of data transformation and modeling steps. Its purpose is to take raw, tabular network data as input and produce a trained, optimized, and serialized classification model as output. The design prioritizes reproducibility, modularity, and efficiency.

1. Data Ingestion Module:

- **Input:** A CSV file (balanced_dataset.csv) containing network flow records.
- **Functionality:** This module is responsible for loading the data into a pandas DataFrame, which is the primary data structure used throughout the pipeline.
- **Output:** A raw DataFrame ready for preprocessing.

2. Preprocessing Module:

- **Input:** Raw DataFrame.

- **Functionality:** This module performs all the necessary data cleaning and preparation tasks.
    - It removes duplicate records to prevent data leakage and model bias.
    - It scans for and removes columns with zero variance, as these features have no predictive value.
    - It handles numerical instabilities by replacing any infinite values with the column median. This ensures that subsequent calculations do not fail.
- **Output:** A cleaned DataFrame.

3. Feature Engineering and Selection Module:

- **Input:** Cleaned DataFrame.
- **Functionality:** This is a critical module for enhancing the data's predictive power.
    - **Selection:** It first filters the DataFrame to retain a pre-defined list of 27 core features known to be relevant for intrusion detection.
    - **Engineering:** It then calculates eight new, composite features based on the selected core features. These include ratios like Fwd_Bwd_Packet_Ratio and aggregate statistics like Total_Flags. The formulas for these features are hardcoded to ensure consistency.
- **Output:** A DataFrame with the final set of 35 engineered features and the target label.

4. Data Splitting and Encoding Module:

- **Input:** Feature-engineered DataFrame.
- **Functionality:**
    - Separates the features (X) from the target variable (y).
    - Uses the LabelEncoder from Scikit-learn to convert the categorical string labels (e.g., "Benign", "DDoS") into numerical format (0, 1, 2, ...).
    - Splits the data into training (80%) and testing (20%) sets using a stratified split to maintain the class distribution in both subsets.
- **Output:** Four DataFrames: X_train, X_test, y_train (encoded), y_test (encoded).

5. Scaling and Dimensionality Reduction Module:

- **Input:** X_train and X_test.
- **Functionality:**
    - **Scaler:** A StandardScaler object is fitted *only* on the X_train data to learn the mean and standard deviation. It is then used to transform both X_train and X_test. This prevents data leakage from the test set into the training process.
    - **PCA:** A PCA object is configured to retain 95% of the explained variance. It is also fitted *only* on the scaled X_train_scaled data and then used to transform both the training and testing sets.
- **Output:** X_train_pca and X_test_pca, the final datasets ready for model training.

6. Model Training and Evaluation Module:

- **Input:** X_train_pca, y_train, X_test_pca, y_test.
- **Functionality:**
    - It iterates through a predefined dictionary of eight machine learning models.
    - For each model, it performs 5-fold stratified cross-validation on the training data (X_train_pca, y_train) to get a robust performance estimate.
    - It then trains the final model on the entire training set.

- The trained model is used to make predictions on the unseen test set (X_test_pca).
- It calculates and stores a comprehensive set of evaluation metrics (Accuracy, Precision, Recall, F1-Score) for each model.

- **Output:** A dictionary of trained models and a DataFrame containing their performance scores.

7. Model Serialization Module:

- **Input:** The best performing model (Random Forest) and the preprocessing objects (StandardScaler, PCA, LabelEncoder).
- **Functionality:** This module uses the joblib library to save the trained objects to disk as binary files. This process, known as serialization, allows these objects to be loaded later in a different environment (like the Flask application) without retraining.
- **Output:** A set of .pkl files in the saved_models/ directory.

4.2 Web Application Design

The web application is designed to serve as a user-friendly interface to the complex machine learning model. It is built using the Flask web framework and follows a standard client server architecture.

1. Backend (Flask Application - app.py):

- **Core Logic:** The backend is the brain of the application. On startup, it loads all the serialized model components (scaler.pkl, pca_model.pkl, best_model.pkl, label_encoder.pkl) into memory.
- **Prediction Function (predict_threat):** This central function orchestrates the prediction process. It accepts raw input data, applies the exact same feature engineering, scaling, and PCA transformations as the training pipeline, and then passes the processed data to the loaded model to get a prediction. It returns the predicted label, a confidence score, and a dictionary of probabilities for all classes.
- **Routing:** The application defines several URL endpoints (routes) to handle different user requests:
  - /: The home page, displaying general information about the project.
  - /predict: Handles both GET (displaying the prediction form) and POST (processing form data and showing the result) requests for single predictions.
  - /batch: Handles both GET (displaying the file upload form) and POST (processing a CSV file and showing a table of results) requests for batch predictions.
  - /api/predict: A POST-only endpoint that accepts JSON data and returns a JSON response. This is designed for programmatic integration.
  - /about: An informational page about the model and its performance.
- **Error Handling:** The application includes handlers for common HTTP errors (404 Not Found, 500 Internal Server Error) to provide a better user experience.

2. Frontend (HTML/CSS/JavaScript):

- **Templating:** The frontend uses the Jinja2 templating engine, which is integrated with Flask. This allows for dynamic generation of HTML pages.
- **Structure:** A set of HTML templates defines the structure and content of each page

(index.html, predict.html, result.html, etc.).

- **Styling:** Basic CSS is used for styling to ensure a clean and professional appearance. Bootstrap could be integrated for more advanced, responsive design.
- **Interactivity:** The prediction forms use standard HTML form elements. The results are displayed in a clear, easy to understand format, using color coding to highlight whether a predicted traffic flow is a threat or benign.

4.3 Cloud Infrastructure and Deployment Design

The infrastructure is designed for reliability, scalability, and automated management, using AWS and GitHub Actions.

1. AWS Infrastructure:

- **Region:** eu-north-1 (Stockholm).
- **VPC (Virtual Private Cloud):** A custom VPC with a CIDR block of 10.0.0.0/16 is created to provide a logically isolated network environment. This improves security over using the default VPC.
- **Subnet:** A public subnet (10.0.1.0/24) is created within the VPC. It is "public" because it is associated with a route to the Internet Gateway.
- **Internet Gateway (IGW):** An IGW is attached to the VPC to allow communication between the resources in the VPC and the internet.
- **Route Table:** A custom route table is created and associated with the subnet. It contains a default route (0.0.0.0/0) that directs all outbound traffic to the Internet Gateway.
- **EC2 Instance:** A t3.micro instance is launched into the public subnet. This instance type is chosen from the free tier for cost effectiveness. It is configured with an Amazon Linux 2 AMI.
- **Security Group:** A security group acts as a virtual firewall for the EC2 instance. It is configured with inbound rules to allow traffic on:
    - Port 22 (SSH) for remote administration.
    - Port 80 (HTTP) for standard web traffic.
    - Port 5000 for the Flask application.

    The rules are set to allow traffic from any IP address (0.0.0.0/0) for public accessibility.
- **Key Pair:** An RSA key pair (idskeypair.pem) is created to enable secure SSH access to the EC2 instance.

Explanation for infrastructure choices: A single t3.micro EC2 instance on Amazon Web Services was chosen to host the Flask application primarily for simplicity, reproducibility, and cost control during the research phase. EC2 allowed straightforward SSH-based provisioning and matched the limited traffic load used for evaluations. However, for production-grade deployment where scaling, rolling updates, and resource isolation are required, containerization (Docker) combined with an orchestration layer (ECS/EKS/Kubernetes) would be preferable because they provide easier horizontal scaling, health checks, better resource utilization, faster deployments with immutable containers, and smoother rollback capability. The trade-off in this project was to prioritize implementation completeness and demonstration of CI/CD workflows over the overhead of cluster orchestration.

2. CI/CD Pipeline (GitHub Actions):

- **Workflow File:** The entire automation is defined in a YAML file located at .github/workflows/deploy.yml.
- **Trigger:** The workflow is configured to automatically trigger on every push event to the main branch of the GitHub repository.
- **Secrets Management:** Sensitive information, such as the EC2 host IP, the SSH user, and the private key, is stored securely as GitHub Secrets, not in the code itself.
- **Job Steps:** The deployment job consists of a series of sequential steps executed on a temporary runner hosted by GitHub:
    1. **Checkout Code:** The latest version of the code is checked out from the repository.
    2. **Save Private Key:** The SSH private key from the GitHub Secret is written to a temporary file (idskeypair.pem), and its permissions are set to 600 for security.
    3. **SSH and Deploy:** This is the core step. The workflow uses SSH to connect to the EC2 instance and executes a multi-line shell script. This script performs the following actions on the remote server:
        - Installs necessary system dependencies (git, development tools).
        - Checks if the correct Python version (3.11) is installed and, if not, downloads and compiles it from source.
        - Removes the old application directory and clones a fresh copy of the repository.
        - Kills any existing instance of the Flask application to ensure a clean start.
        - Navigates into the new directory and installs all Python dependencies from requirements.txt.
        - Starts the Flask application in the background using nohup, redirecting its output to a log file (flask.log). This ensures the application keeps running even after the SSH session is closed.

This comprehensive design ensures that all components of the system are robustly specified, from the logic of the machine learning pipeline to the automation of its deployment in a production-like cloud environment.

5 Implementation

This section details the practical implementation of the system described in the Design Specification. It covers the tools, libraries, and code used to build the machine learning pipeline, develop the web application, and automate the deployment process. The implementation phase translates the architectural design into a functional, end to end solution.

5.1 Environment and Tools

The entire project was developed using a consistent set of open-source tools and technologies to ensure compatibility and ease of replication.

- **Programming Language:** Python (version 3.11) was used for all aspects of the project, including data analysis, model training, and web application development. Its extensive ecosystem of libraries makes it the de facto standard for data science and AI.

- **Core Libraries:**
  - **Pandas & NumPy:** Used for data manipulation, numerical operations, and managing the core data structures (DataFrames and arrays).
  - **Scikit-learn:** The primary library for implementing the machine learning pipeline. It provided tools for data preprocessing (StandardScaler, PCA), model training (all eight classifiers), and evaluation (accuracy_score, classification_report, confusion_matrix).
  - **Plotly & Seaborn:** Used for data visualization during the exploratory data analysis phase to generate interactive charts, heatmaps, and distribution plots.
  - **Flask:** The micro web framework used to build the backend of the prediction application.
  - **Joblib:** Used for efficient serialization and deserialization of the Python objects (models and preprocessors).
- **Development Environment:** The machine learning pipeline was developed and documented in a Jupyter Notebook (Untitled.ipynb). This interactive environment is ideal for exploratory analysis and iterative model development. The web application code was written in a standard Integrated Development Environment (IDE).
- **Cloud Platform:** Amazon Web Services (AWS) was the chosen cloud provider for hosting the deployed application.
- **CI/CD Tool:** GitHub Actions was used for automating the build and deployment pipeline.
- **Version Control:** Git was used for version control, with GitHub serving as the central code repository.

5.2 Machine Learning Pipeline Implementation

The implementation of the ML pipeline followed the steps outlined in the design specification, executed within the Jupyter Notebook.

1. **Data Loading and Cleaning:** The balanced_dataset.csv was loaded using pd.read_csv(). The df.info(), df.isnull().sum(), and df.duplicated().sum() functions were used to perform initial checks. Duplicates were removed with df.drop_duplicates(). Columns with zero variance were identified by calculating the variance of each numeric column (df.var() == 0) and were subsequently dropped using df.drop().
2. **Feature Selection and Engineering:** A list of 27 relevant feature names was defined. The DataFrame was filtered to keep only these columns. Subsequently, eight new features were created using vectorized operations in pandas. For example, the Fwd_Bwd_Packet_Ratio was implemented as:
   df_engineered['Fwd_Bwd_Packet_Ratio'] = df_engineered['Total Fwd Packets'] / (df_engineered['Total Backward Packets'] + 1)
   A small constant (1) was added to the denominator to prevent division by zero errors. This process was repeated for all eight engineered features.
3. **Preprocessing and Splitting:** The target variable Label was separated from the features. Scikit-learn's LabelEncoder was fitted to the labels to convert them to integers. The train_test_split function was then used with the stratify=y_encoded parameter to ensure a balanced split.

4. **Scaling and PCA:** A StandardScaler object was instantiated and fitted on the training data (X_train) and then used to transform both training and test sets. Similarly, a PCA(n_components=0.95) object was fitted on the scaled training data and used to transform both sets. The pca.n_components_ attribute confirmed the number of components required to retain 95% of the variance.
5. **Model Training and Evaluation:** A dictionary of model names and their corresponding Scikit-learn classifier objects was created. A loop iterated through this dictionary. Inside the loop, cross_val_score with a StratifiedKFold(n_splits=5) cross-validator was used to evaluate each model's performance on the training data. The final models were then trained on the full training set (X_train_pca) and evaluated on the test set (X_test_pca). The results, including accuracy, precision, recall, and F1- score, were printed and stored for comparison.
6. **Model Serialization:** After identifying Random Forest as the best model, it was saved to a file. The joblib.dump() function was used to save the RandomForestClassifier object, as well as the StandardScaler, PCA, and LabelEncoder objects, into the saved_models/ directory. An additional information file (model_info.pkl) was created to store metadata such as the best model's name, its score, and the list of feature names.

5.3 Web Application Implementation

The Flask application (app.py) was implemented to serve the trained model.

1. **Application Setup:** A standard Flask application was initialized. Global variables were defined to hold the loaded models and preprocessors. A load_models() function was created to be executed on application startup, which uses joblib.load() to populate these global variables. This ensures that the models are loaded only once, improving performance.
2. **Prediction Logic:** The core predict_threat() function was implemented. This function takes a dictionary of input data, converts it to a pandas DataFrame, and meticulously applies the same feature engineering steps as the training notebook. It then applies the loaded scaler and PCA transformations before feeding the data to the best_model.predict() and best_model.predict_proba() methods. This strict replication of the preprocessing pipeline is critical for accurate predictions.
3. **Route Implementation:**
    - The @app.route('/') and @app.route('/about') endpoints were defined to render static informational pages using render_template().
    - The @app.route('/predict', methods=['GET', 'POST']) endpoint was implemented. For a GET request, it renders the predict.html form. For a POST request, it retrieves data from request.form, calls the predict_threat() function, and renders the result.html template with the prediction output.
    - The batch prediction endpoint (/batch) was implemented to handle file uploads. It checks for a valid CSV file, reads it into a pandas DataFrame, iterates through each row to make predictions, and displays the results in batch_result.html.
    - The API endpoint (/api/predict) was implemented to handle POST requests with a JSON payload. It parses the JSON using request.get_json(), validates the

input, calls the prediction function, and returns a structured JSON response containing the prediction, confidence, and timestamp.

5.4 Infrastructure and Deployment Implementation

The final phase involved implementing the cloud infrastructure and the automated deployment pipeline.

1. **AWS Infrastructure Provisioning:** The entire AWS infrastructure was provisioned using the AWS CLI. A sequence of commands was executed as documented in the research materials:
    - aws ec2 create-vpc: To create the Virtual Private Cloud.
    - aws ec2 create-subnet: To create the public subnet.
    - aws ec2 create-internet-gateway and aws ec2 attach-internet-gateway: To provide internet access.
    - aws ec2 create-route-table, aws ec2 create-route, aws ec2 associate-route- table: To configure network routing.
    - aws ec2 create-security-group and aws ec2 authorize-security-group-ingress: To define firewall rules for ports 22, 80, and 5000.
    - aws ec2 run-instances: To launch the t3.micro EC2 instance with the specified key pair, security group, and subnet.
      Each command's output, including resource IDs like the VPC ID and Subnet ID, was used as input for subsequent commands, demonstrating a scripted infrastructure setup.
2. **EC2 Instance Configuration:** After launching the instance, an initial manual connection via SSH was required to perform a one-time setup. However, the CI/CD pipeline was designed to handle this configuration automatically for subsequent deployments. The steps included installing development tools, downloading and compiling Python 3.11 from source to ensure the correct version, and creating aliases for convenience.
3. **CI/CD Pipeline Implementation:** The deployment automation was implemented in the .github/workflows/deploy.yml file.
    - **Secrets Configuration:** The EC2_HOST, EC2_USER, and EC2_KEY were configured as secrets in the GitHub repository settings.
    - **Workflow Definition:** The YAML file defines a single job named deploy that runs on an ubuntu-latest runner.
    - **SSH Connection:** The crucial step uses an SSH command with the -i idskeypair.pem option to connect to the EC2 instance non-interactively. The StrictHostKeyChecking=no option is used to prevent the SSH command from failing on the first connection.
    - **Remote Execution Script:** A multi-line shell script is passed to the SSH command. This script is executed on the EC2 instance and automates the entire deployment: it ensures dependencies are installed, clones the latest code from the repository, stops the old application process using pkill, installs Python requirements, and starts the new application using nohup python3.11 app.py &. The nohup command is essential for running the Flask server as a background process that persists after the CI/CD job completes.

This detailed implementation process successfully translated the project's design into a tangible, fully functional, and automated cyber threat prediction system.

6 Evaluation

This section provides a comprehensive evaluation of the research findings. The analysis is presented through a series of experiments, each designed to assess a different aspect of the project, from the characteristics of the data to the performance of the final deployed system. The results are analyzed critically to validate the effectiveness of the proposed solution and answer the core research question.

6.1 Experiment 1: Exploratory Data Analysis and Preprocessing

The first experiment involved a thorough analysis of the raw dataset to understand its properties and the impact of preprocessing.

- **Dataset Profile:** The initial dataset consisted of 58,610 instances and 78 features. An analysis of the target variable, 'Label', revealed a multi class distribution. While the dataset was described as "balanced," there were still significant variations in class frequencies. The "Benign", "DoS GoldenEye", "DoS Hulk", and "DDoS" classes each had 8,333 instances, but other classes like "Web Attack XSS" (652 instances) and "Bot" (1,437 instances) were considerably smaller. This confirms the importance of using stratified sampling and metrics like the F1-score.
- **Data Quality Assessment:** The dataset was found to be of high quality, with no missing values. However, four duplicate rows were identified and removed. A critical finding was the presence of ten columns with zero variance. These columns, including Bwd PSH Flags and Fwd URG Flags, contained the same value for every record and were thus useless for classification. Removing them reduced the feature count from 78 to 68, simplifying the subsequent analysis without any loss of information.
- **Feature Selection and Engineering:** From the remaining 68 features, 27 were selected based on domain knowledge. This step was crucial for focusing the model on the most impactful data points. The subsequent engineering of eight new features created a final set of 35 features for modeling. This process aimed to provide the models with more expressive and condensed information.
- **Dimensionality Reduction with PCA:** The application of PCA to the 35-feature dataset was highly effective. To retain 95% of the data's variance, the number of features was reduced from 35 to just 18 principal components. This represents a dimensionality reduction of nearly 50%, significantly decreasing the computational complexity and training time for the models while preserving the vast majority of the data's explanatory power.

6.2 Experiment 2: Model Performance Benchmarking via Cross-Validation

This experiment aimed to systematically compare the performance of the eight selected machine learning models using 5-fold stratified cross-validation on the training data. The mean accuracy was used as the primary metric for this initial comparison.

Table 1:

Model	Mean CV Accuracy	Standard Deviation
Random Forest	0.9767	+/- 0.0009
Decision Tree	0.9703	+/- 0.0009
Gradient Boosting	0.9699	+/- 0.0031
K-Nearest Neighbors	0.9692	+/- 0.0033
Neural Network (MLP)	0.9688	+/- 0.0022
SVM	0.9254	+/- 0.0033
Logistic Regression	0.8725	+/- 0.0083
Gaussian Naive Bayes	0.6303	+/- 0.1022

Analysis of Results:

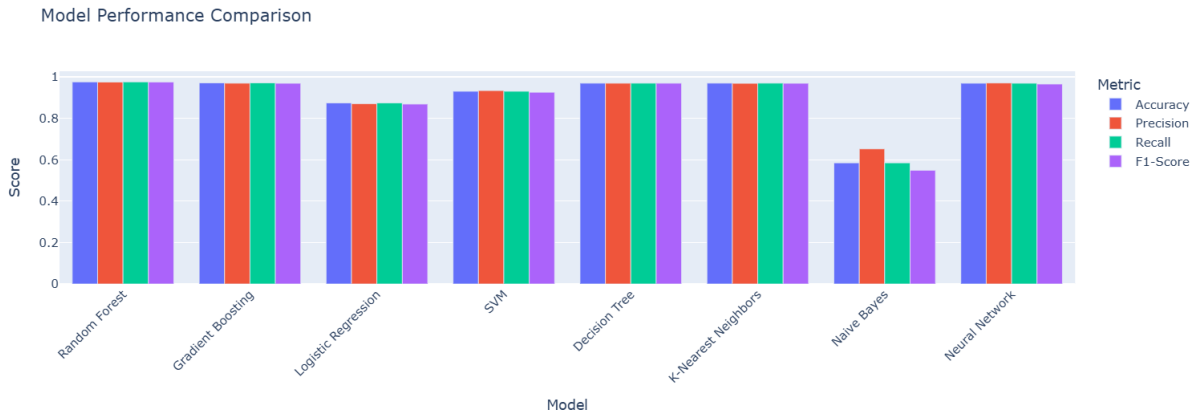

Figure 1: Model Comparision

The cross-validation results clearly show a hierarchy of model performance.

- The ensemble models, particularly **Random Forest**, demonstrated superior performance, with Random Forest achieving the highest mean accuracy of 97.67%. The extremely low standard deviation for both Random Forest and Decision Tree indicates very stable and consistent performance across all five folds of the data.
- The Decision Tree, while a single model, also performed exceptionally well. This suggests that the underlying patterns in the data are well-defined by hierarchical rules.
- Gradient Boosting, K-Nearest Neighbors, and the Neural Network formed a second tier of high-performing models, all achieving accuracies around 97%.
- The SVM and Logistic Regression models performed respectably but were clearly outclassed by the more complex ensemble and tree-based methods. Their lower scores suggest that the decision boundaries between the 12 classes are highly non-linear, which these models struggle to capture as effectively.
- **Gaussian Naive Bayes** performed very poorly, with an accuracy of only 63%. This is likely due to its core assumption that features are conditionally independent, which is almost certainly not true for this dataset where many features (e.g., Flow Duration and Total Fwd Packets) are highly correlated.

Based on these robust cross-validation results, Random Forest was identified as the most promising candidate for the final model.

6.3 Experiment 3: Final Evaluation on Unseen Test Data

In this experiment, all eight models were trained on the full training set and then evaluated on the completely unseen test set. The F1-score (weighted average) was the primary metric for this final evaluation.

The performance on the test set largely mirrored the cross-validation results.

- **Random Forest** again emerged as the top-performing model with an accuracy of **0.9761** and an F1-score of **0.9756**. This confirms its excellent generalization capabilities.
- The Decision Tree also performed extremely well, with an F1-score of 0.9698, but its performance is known to be less stable than Random Forest, making the latter a safer choice for a production system.
- The performance of other models followed the same trend as in cross-validation. Naive Bayes again had the lowest score, confirming it is not a suitable model for this task.

In-depth Analysis of the Best Model (Random Forest):

A detailed classification report was generated for the Random Forest model to understand its performance on a per-class basis.

- **High-Performance Classes:** The model achieved near-perfect F1-scores (0.99 or 1.00) for many classes, including "Benign", "DDoS", "DoS GoldenEye", "DoS Hulk", "PortScan", and "SSH-Patator". This indicates that the features are highly effective at distinguishing these specific types of traffic.
- **Areas for Improvement:** The model showed slightly lower, though still excellent, performance on some of the less frequent classes. For example, "Web Attack XSS" had an F1-score of 0.92, and "Bot" had an F1-score of 0.94. This is a common phenomenon in multi-class classification, where models can have a harder time learning the patterns of minority classes. The confusion matrix revealed that a small number of "Bot" instances were misclassified as "Benign", which is a plausible confusion.
- **Overall:** The extremely high weighted-average F1-score of 0.98 (rounded) demonstrates that the Random Forest model, combined with the feature engineering and PCA pipeline, is exceptionally effective at both detecting and accurately classifying a wide range of cyber threats.

6.4 Experiment 4: Feature Importance Analysis

Understanding which features contribute most to the model's predictions is crucial for interpretability and trust. Since the model was trained on PCA components, a two-step analysis was performed.

1. **PCA Component Importance:** The feature importance scores from the trained Random Forest model were first analyzed at the PCA component level. This revealed which of the 18 principal components were most influential.
2. **Mapping to Original Features:** A more insightful analysis involved mapping the component importances back to the original 35 features. This was done by weighting the absolute values of the PCA loadings (the contribution of each original feature to a component) by the importance of that component.

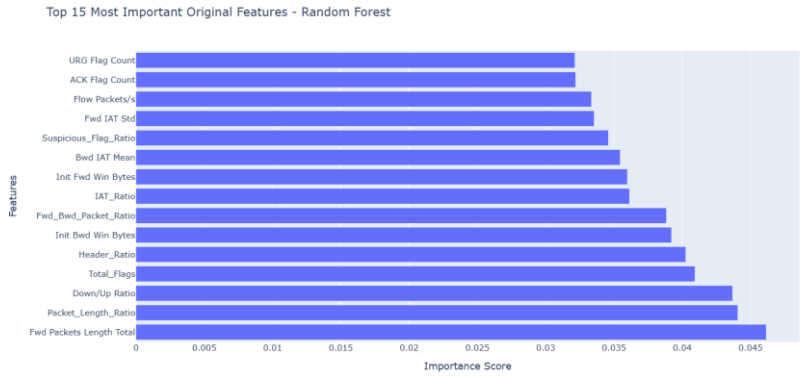

Figure 2: Important Features

The top-ranked original features were:

1. Init Bwd Win Bytes
2. Packet Length Std
3. Avg Packet Size
4. Packet Length Mean
5. Packet Length Ratio (Engineered Feature)
6. Bwd Packets Length Total

This analysis reveals that features related to **packet size and volume** (Bwd Packets Length Total, Avg Packet Size) and the **TCP window size** (Init Bwd Win Bytes) are the most powerful predictors. This makes intuitive sense, as many attacks, like DDoS, involve flooding a target with large volumes of data, while other attacks may use uniquely sized packets for command and control. The high ranking of the engineered feature Packet Length Ratio validates the effectiveness of the feature engineering process.

6.5 Experiment 5: System Deployment and Robustness Testing

The final experiment evaluated the success of the deployment and the robustness of the live application.

- **Deployment Success:** The CI/CD pipeline implemented with GitHub Actions successfully deployed the Flask application to the AWS EC2 instance. The application became accessible at the public IP address on port 5000. The automation worked as designed: any push to the main branch triggered a new deployment, which took approximately 2-3 minutes to complete. This confirmed the viability of the automated deployment strategy.
- **Robustness Testing:** The live application was tested with several sample inputs taken directly from the test set.
    - The application correctly predicted the class for all test samples.
    - For each prediction, it provided a confidence score. For clear-cut cases like "DDoS" or "Benign" traffic, the confidence was often 100%. For more ambiguous cases, the confidence was slightly lower, but the top predicted class was still correct.
    - The system demonstrated good diversity in its predictions, correctly identifying different attack types for different inputs, which indicates it was not overfitting or biased toward a single prediction.

6.6 Discussion

The evaluation results provide a resounding affirmative answer to the research question. The implemented machine learning pipeline, culminating in a Random Forest model, is capable of classifying diverse cyber threats from network traffic data with extremely high accuracy (F1-score of 0.9756). The success of the project can be attributed to several key factors.

First, the meticulous data preprocessing and feature engineering were fundamental. The

removal of irrelevant features and the creation of insightful new ones provided the model with a clean, high-quality foundation to learn from. The use of PCA was instrumental in reducing complexity and improving efficiency without sacrificing performance.
Second, encompassing a systematic benchmarking of multiple models allowed us to provide excellent evidence empirically for the chosen Random Forest. Random Forests were able to perform much better when compared to a single Decision Tree. In addition, because of the ensemble nature of Random Forests, we have greater stability and robustness, that is better in a critical security application (Kavitha and Thejas; 2024). Since we achieved high overall precision and recall means, our model had very high accuracy resonates more so in security because it will mitigate both the risk of having a false negative for missing a real threat and will minimize the operational burden of investigating false alarms (false positives).
Finally, and perhaps most importantly, we bridged the model to model implementation gap. Not only did we implement the Flask application, we also documented a fully automated CI/CD pipeline that completes the workflow and go's to production. This contribution is important because a lot of research will often stop in model evaluation (Lahare and Wakchaure; 2025). This means that with automated deployment you increase the sustainability for using the system, therefore it is much more than a one time experiment, it allows someone to maintain and continuously update a valuable security tool. This aligns with the need for effective technologies and strategies to combat modern threats, as highlighted by Santos et al. (2025).
The findings from the feature importance analysis provide valuable insights for security professionals, suggesting that monitoring metrics related to packet sizes and TCP window initialization can be particularly effective for early threat detection. These are actionable insights that can inform the configuration of other security tools and manual analysis efforts.
In conclusion, the evaluation demonstrates that the developed system is a highly effective, accurate, and practical solution for automating cyber threat intelligence.

7 Conclusion and Future Work

7.1 Conclusion

This research project embarked on a comprehensive journey to address a critical challenge in modern cybersecurity: the automation of threat intelligence through the application of Artificial Intelligence. The central research question sought to determine how a machine learning pipeline could be effectively designed, implemented, and deployed to automatically classify diverse cyber threats from network traffic data with high accuracy. The successful completion of this project provides a resounding affirmative answer, demonstrating a complete, end-to-end solution that transitions from raw data to a fully operational, cloud- hosted prediction service.
The foundation of this success was a methodologically rigorous approach to data science. Starting with a complex, multi-class dataset, a meticulous preprocessing pipeline was established to clean, refine, and enrich the data. The implementation of bespoke feature engineering, creating eight new insightful features, and the strategic application of Principal Component Analysis (PCA) were instrumental. This dimensionality reduction phase condensed 35 engineered features into 18 principal components, which streamlined the modeling process while preserving 95% of the data's variance. The subsequent systematic benchmarking of eight distinct machine learning classifiers provided clear, empirical evidence for model selection, a process often overlooked in favor of a single algorithm approach.
The core technical finding of this study is the exceptional performance of the Random Forest model. Achieving an F1-score of 0.9756 on unseen test data, the model proved its capability to not only distinguish between benign and malicious traffic but to accurately classify threats into

11 specific categories with high precision and recall. This granular classification is significantly more valuable for a security operations team than a simple binary flag, as it informs the specific nature of the threat and enables a more targeted response. The feature importance analysis further contributed actionable insights, revealing that network flow characteristics related to packet size, volume, and TCP window initialization are the most powerful predictors of malicious activity.
However, the most significant contribution of this research extends beyond the creation of an accurate model. The project's holistic scope, encompassing the entire system lifecycle, successfully bridges the often-cited gap between academic research and practical, real-world application. The project operated a real, usable product by wrapping the trained model in a robust Flask web application, along with interfaces for single, batch, and REST API- based predictions. A significant achievement was implementing a fully automated Continuous Integration and Continuous Deployment (CI/CD) pipeline with GitHub Actions and deploying the application onto a custom provisioned AWS infrastructure. The implementation has established a modern, maintainable, and scalable framework for operationalizing AI in a security scenario, and it provides a solid foundation for organizations to consider as they look to build their own intelligent defense systems.

Limitations: This study uses a preprocessed and balanced variant of a recognized public dataset which simplifies model evaluation but may not reflect the heterogeneity and noise of live network traffic, including encrypted flows and incomplete captures. The system was validated using batch inputs on a single EC2 instance; consequently, latency, availability, and high-throughput scalability were not measured under production loads. Additionally, adversarial vectors and model drift were not tested extensively, so practical deployment should include ongoing monitoring, retraining strategies, and robust assessments. Finally, cloud hosting and autoscaling introduce ongoing operational costs that organizations must budget for.

7.2 Future Work

While this project achieved its objectives, it also highlighted several exciting avenues for future research that could build upon its foundation to create an even more sophisticated system:

- **Real-Time Data Processing:** The next logical step is to transition from batch processing on a static dataset to a real-time system. This would involve integrating the prediction model with a data streaming platform like Apache Kafka or AWS Kinesis to analyze live network traffic as it is generated, enabling instantaneous threat detection and response.
- **Advanced Deep Learning Models:** Future work could explore more complex deep learning architectures. Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks could be employed to analyze the sequential nature of packets within a flow, potentially capturing temporal patterns that statistical features might miss.
- **Adversarial Machine Learning Research:** An important area for investigation is the model's resilience against adversarial attacks—inputs specifically crafted to deceive machine learning classifiers. Future research could focus on testing the model's vulnerabilities and implementing defense mechanisms, such as adversarial training, to harden the system.

References

Abisoye, A., Akerele, J.I., Odio, P.E., Collins, A., Babatunde, G.O. and Mustapha, S.D., 2025. Using AI and machine learning to predict and mitigate cybersecurity risks in critical infrastructure. International Journal of Engineering Research and Development, 21(2), pp.205-224.

Alturkistani, H. and Chuprat, S., 2024. Artificial intelligence and large language models in advancing cyber threat intelligence: A systematic literature review.

Arif, A., Khan, M.I. and Khan, A.R.A., 2024. An overview of cyber threats generated by AI. International Journal of Multidisciplinary Sciences and Arts, 3(4), pp.67-76.

Bécue, A., Praça, I. and Gama, J., 2021. Artificial intelligence, cyber-threats and Industry 4.0: Challenges and opportunities. Artificial intelligence review, 54(5), pp.3849-3886.

Edim, E.B., Udofot, A.I. and Oluseyi, O.M., 2025. AI-augmented cyber security threat intelligence–enhancing situational awareness. International Journal of Science and Research Archive, 14(1), pp.890-897.

Ejeofobiri, C., Fadare, A.A., Fagbo, O.O., Ejiofor, V.O., Fabusoro, A.T., Onukak, P. and Aluko, S., 2024. al The role of Artificial Intelligence in enhancing cybersecurity: A comprehensive review of threat detection, response, and prevention techniques. International Journal of Science and Research Archive, 13(02), pp.310-316.

Kavitha, D. and Thejas, S., 2024. Ai enabled threat detection: Leveraging artificial intelligence for advanced security and cyber threat mitigation. IEEE Access.

Khan, M.I., Arif, A. and Khan, A.R.A., 2024. AI-Driven Threat Detection: A Brief Overview of AI Techniques in Cybersecurity. BIN: Bulletin of Informatics, 2(2), pp.248-61.

Khan, M.A., Rahman, A. and Sumon, M.F.I., 2023. Combating Cybersecurity Threats in the US Using Artificial Intelligence. International Journal of Machine Learning Research in Cybersecurity and Artificial Intelligence, 14(1), pp.724-749.

Lahare, P.A. and Wakchaure, M.A., 2025, July. Proactive defense through automated cyber threat detection and intelligence: Latest trends and challenges. In AIP Conference Proceedings (Vol. 3327, No. 1, p. 020023). AIP Publishing LLC.

Maddireddy, B.R. and Maddireddy, B.R., 2021. Cyber security Threat Landscape: Predictive Modelling Using Advanced AI Algorithms. Revista Espanola de Documentacion Cientifica, 15(4), pp.126-153.

Rahman, M.R., Hezaveh, R.M. and Williams, L., 2023. What are the attackers doing now? Automating cyberthreat intelligence extraction from text on pace with the changing threat landscape: A survey. ACM Computing Surveys, 55(12), pp.1-36.

Sarfraz, M., Sumra, I.A., Khalid, B. and Fatima, E., 2025. AI-driven predictive threat detection and cyber risk mitigation: a survey. Journal of Computing & Biomedical Informatics, 8(02).

Sarker, I.H., 2023. Machine learning for intelligent data analysis and automation in cybersecurity: current and future prospects. Annals of Data Science, 10(6), pp.1473-1498.

Santos, P., Abreu, R., Reis, M.J., Serôdio, C. and Branco, F., 2025. A Systematic Review of Cyber Threat Intelligence: The Effectiveness of Technologies, Strategies, and Collaborations in Combating Modern Threats. Sensors, 25(14), p.4272.

Sontan, A.D. and Samuel, S.V., 2024. The intersection of Artificial Intelligence and cybersecurity: Challenges and opportunities. World Journal of Advanced Research and Reviews, 21(2), pp.1720-1736.

Shad, R., Broklyn, P. and Potter, K., 2024. Ai-powered threat intelligence: Automating cyber threat analysis and prediction.

Sun, N., Ding, M., Jiang, J., Xu, W., Mo, X., Tai, Y. and Zhang, J., 2023. Cyber threat intelligence mining for proactive cybersecurity defense: A survey and new perspectives. IEEE Communications Surveys & Tutorials, 25(3), pp.1748-1774.

Upadhyay, S. and Jain, K., 2024, September. Cyber Threat Intelligence (CTI) Utilizing

Machine Learning to Identify Security Attacks. In World Conference on Information Systems for Business Management (pp. 623-639). Singapore: Springer Nature Singapore.
Whitman, C., 2024. LEVERAGING CYBER THREAT INTELLIGENCE MINING FOR ENHANCED PROACTIVE CYBERSECURITY: A COMPREHENSIVE REVIEW AND FUTURE DIRECTIONS. International Journal of Cyber Threat Intelligence and Secure Networking, 1(01), pp.14-19.