VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belgavi-590018, Karnataka, INDIA

PROJECT REPORT

On

"Building Network Security System using MLOPS"

Submitted in partial fulfilment of the requirements for the award of the degree

BACHELOR OF ENGINEERING

IN

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

FOR THE ACADEMIC YEAR

2024-2025

BY

Danesh P M	**1VJ21AI007**
Harshanand Babu Naik	**1VJ21AI011**
Mir Hussain Ali	**1VJ21AI021**
Sourav V	**1VJ21AI033**

Under the Guidance of
Dr. Naveen Ghorpade
Professor & HOD
Department of AI & ML
VVIT, Bengaluru-560077

Department of Artificial Intelligence and Machine Learning

Vijaya Vittala Institute of Technology

#35/1, Dodda Gubbi Post, Hennur-Bagalur Road, Bengaluru-560077
(Affiliated to Visvesvaraya Technological University, Belagavi,
Recognized by Govt. of Karnataka and Approved by AICTE, New Delhi)
Accredited by NAAC

VIJAYA VITTALA INSTITUTE OF TECHNOLOGY

Hennur – Bagalur Road, Doddagubbi post, Bengaluru -560 077

Department of Artificial Intelligence and Machine Learning

CERTIFICATE

Certified that the project work entitled "**Building Network Security System using MLOPS**" Bonafide work carried out by **Danesh P M, Harshanand Babu Naik, Mir Hussain Ali, Sourav V**, bearing USNs:**1VJ21AI007, 1VJ21AI011, 1VJ21AI021, 1VJ21AI033** students of Vijaya Vittala Institute of Technology in partial fulfilment for the award of Bachelor of Engineering in Artificial Intelligence and Machine Learning of the Visvesvaraya Technological University, Belagavi during the year 2024-2025. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said Degree.

Signature of Guide	Signature of HOD	Signature of Principal
Dr. Naveen Ghorpade	**Dr. Naveen Ghorpade**	**Dr. Sanjeev C Lingareddy**
Professor & HOD	Professor & HOD	Principal,
Dept. of AI&ML	Dept. of AI&ML	VVIT, Bengaluru

EXTERNAL VIVA

Name of the Examiners	**Signature with Date**
1.	
2.	

DECLARATION

We, **Danesh P M**, **Harshanand Babu Naik**, **Mir Hussain Ali**, **Sourav V**, students of VIII semester, B.E., **Department of Artificial Intelligence and Machine Learning**, Vijaya Vittala Institute of Technology, Bengaluru, hereby declare that the dissertation titled "**Building Network Security System using MLOps**" embodies the report of our project work carried out independently by us under the guidance of **Dr. Naveen Ghorpade, Professor and Head of the Department of Artificial Intelligence and Machine Learning, VVIT, Bengaluru**. This work has been undertaken in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Artificial Intelligence and Machine Learning by **Visvesvaraya Technological University, Belagavi**, during the academic year 2024–2025.We further declare that the contents of this dissertation have not been submitted previously by anyone for the award of any degree or diploma to any other university.

Place: Bangalore

Date:

DANESH P M	**(IVJ21AI007)**
Harshanand Babu Naik	**(1VJ21AI011)**
Mir Hussain Ali	**(1VJ21AI021)**
Sourav V	**(1VJ21AI033)**

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our Chairperson, **Smt. Rukmini**, of **Vijaya Vittala Institute of Technology**, for providing the necessary infrastructure and creating a conducive environment for the successful completion of our project.

We are deeply grateful to **Dr. Sanjeev C LingaReddy**, Principal, **Vijaya Vittala Institute of Technology**, for giving us the opportunity to undertake this project as part of our curriculum, in partial fulfilment of the requirements for the award of our degree.

We extend our heartfelt thanks to **Dr. Naveen Ghorpade**, Professor and Head of the Department of Artificial Intelligence and Machine Learning, **Vijaya Vittala Institute of Technology**, for his support, encouragement, and for providing the resources necessary to carry out this work.

We would also like to express our special thanks to our Project internal guide **Prof & HOD Dr. Naveen Ghorpade**, Department of Artificial Intelligence and Machine Learning, **Vijaya Vittala Institute of Technology** for her valuable guidance, consistent encouragement, and unwavering support throughout the course of our project work.

Our sincere thanks also go to all the faculty members of the **Department of Artificial Intelligence and Machine Learning**, **Vijaya Vittala Institute of Technology**, for their continuous support and guidance during the course of this project.

Finally, we are immensely grateful to our families and friends, whose constant encouragement, motivation, and moral support were instrumental in the successful completion of this project.

DANESH P M	**(IVJ21AI007)**
HARSHANAND BABU NAIK	**(1VJ21AI011)**
MIR HUSSAIN ALI	**(1VJ21AI021)**
SOURAV V	**(1VJ21AI033)**

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

In today's rapidly advancing digital landscape, network security has emerged as a primary concern for organizations across the globe. With the increasing sophistication of cyber threats such as phishing, ransomware, advanced persistent threats (APT), and zero-day attacks, traditional security mechanisms are no longer sufficient. The traditional rule-based Intrusion Detection Systems (IDS) lack adaptability and are often reactive rather than proactive, leading to an increased risk of breaches and compromised data integrity.

This project introduces an enhanced network security framework through the integration of Machine Learning Operations (MLOps). MLOps combines Machine Learning (ML) and DevOps practices, enabling the continuous development, deployment, monitoring, and updating of ML models. By automating the entire security pipeline — from data ingestion and validation to model deployment and real-time monitoring — this system provides a dynamic and scalable approach to cybersecurity. The goal is to ensure that threat detection models remain up-to-date, responsive, and capable of handling the evolving nature of cyberattacks.

Our system leverages cloud computing resources, particularly AWS services such as EC2, S3, and ECR, to ensure scalability and resilience. Models are trained using powerful ensemble techniques like Random Forest and Support Vector Machines, optimized through extensive hyperparameter tuning, and deployed via FastAPI in Dockerized environments. Real-time anomaly detection is achieved through REST API endpoints that provide immediate threat identification capabilities.

Moreover, the system incorporates continuous monitoring and retraining strategies, using tools like MLflow and GitHub Actions, to adapt to new attack vectors. Data drift detection mechanisms are embedded within the pipeline to identify when retraining is required, thereby minimizing false positives and maximizing model accuracy.

By adopting this MLOps-driven security architecture, organizations can achieve:

- Real-time threat monitoring and detection
- Automated model retraining and redeployment
- Scalable cloud-based infrastructure
- Reduced human intervention and faster incident response times

CHAPTER-1

INTRODUCTION

1.1 Importance of Network Security

In today's rapidly evolving digital landscape, the importance of network security cannot be overstated. With increasing dependence on the internet and digital platforms for communication, commerce, and data exchange, ensuring the integrity, confidentiality, and availability of networked systems has become a critical priority for organizations and individuals alike.

Network security is designed to protect computer networks and their components from various threats such as unauthorized access, cyberattacks, malware, and data breaches. As the number of devices connected to the internet continues to rise—fueled by the expansion of the Internet of Things (IoT) and cloud computing—network security ensures that these systems function without disruption, safeguard sensitive data, and prevent malicious actors from exploiting vulnerabilities.

Modern enterprises are increasingly reliant on their network infrastructure for business operations. Downtime, data loss, or a successful cyberattack can result in severe financial losses, damage to reputation, legal repercussions, and, in some cases, the collapse of an organization. Consequently, robust network security is indispensable to the continued success and survival of businesses.

1.2 Rise of Cyber Threats

Over the past two decades, cyber threats have evolved from basic malware attacks to sophisticated, multi-vector attacks targeting the heart of corporate and governmental infrastructures. Cybercriminals are using advanced methods like phishing, ransomware, Distributed Denial-of-Service (DDoS) attacks, and zero-day exploits to infiltrate networks. This rise in threat sophistication is attributed to factors such as the globalization of the internet, the growth of high-value targets, and the increasing availability of cyberattack tools on the dark web.

The increasing number of high-profile data breaches, such as those involving financial institutions, healthcare organizations, and governmental agencies, highlights the gravity of these threats. Cyberattacks can cause irreparable damage, ranging from loss of sensitive data to the compromise of national security, leading to a growing concern among stakeholders worldwide. For instance, recent high-profile cyberattacks like the SolarWinds hack and the ransomware attack on Colonial Pipeline showcase the devastating impact on global economies, supply chains, and even national security.

This surge in cybercrime has prompted industries to reassess their cybersecurity strategies and adopt more robust mechanisms to prevent, detect, and mitigate cyberattacks. Traditional methods of defense, while still relevant, are no longer sufficient in the face of ever-evolving cyber threats. This has led to the integration of more advanced and automated tools, such as machine learning and artificial intelligence (AI), into the cybersecurity domain.

1.3 Challenges with Traditional Intrusion Detection Systems (IDS)

Traditional Intrusion Detection Systems (IDS) were developed to monitor network traffic and detect malicious activity, typically through signature-based methods. These systems compare incoming data to known attack patterns or predefined signatures. However, the traditional approach has several inherent limitations in today's complex and dynamic cyber threat landscape:

Signature-based Detection: Traditional IDS primarily relies on signatures, which are patterns or characteristics of known attacks. While effective against known threats, this approach is ineffective at detecting new, unknown attacks or zero-day exploits. As a result, attackers can bypass signature-based IDS by simply modifying their methods or utilizing novel attack techniques.

High Rate of False Positives and False Negatives: Traditional IDS often suffers from high rates of false alarms (false positives) and failure to detect attacks (false negatives). This can lead to inefficient resource usage as security teams must investigate numerous false alarms, reducing the overall effectiveness of the system. False negatives, on the other hand, can allow cybercriminals to go undetected for extended periods, allowing them to inflict significant damage.

Lack of Scalability: With the exponential growth of network traffic, especially in enterprise environments, traditional IDS solutions often struggle to keep up with the sheer volume of data they need to analyze. As networks scale, traditional IDS systems may become overloaded, leading to slower detection times and the inability to handle large amounts of traffic in real-time.

Inability to Adapt: Traditional systems require frequent updates to remain effective against emerging threats. The manual process of updating attack signatures is time-consuming and often fails to provide timely protection against new attack vectors.

These challenges highlight the need for more advanced, dynamic, and adaptive solutions that can keep up with the fast-evolving nature of cyber threats.

1.4 Need for Machine Learning and MLOps Integration

Given the increasing sophistication of cyberattacks and the limitations of traditional IDS solutions, there has been a growing interest in integrating Machine Learning (ML) and MLOps (Machine Learning Operations) into the realm of network security. Machine learning offers a powerful approach for detecting novel and complex attack patterns by analyzing large datasets and identifying anomalies that may be indicative of a security breach.

Machine learning models can be trained on vast amounts of network data to recognize patterns of normal behavior, allowing them to detect deviations that could signal malicious activity. For instance, ML-based Intrusion Detection Systems (IDS) can identify previously unknown attack vectors by recognizing anomalous behavior that deviates from the established patterns of normal network traffic.

Key benefits of integrating machine learning into network security include:

Anomaly Detection: ML algorithms can be trained to differentiate between normal and malicious activity, allowing for the detection of new or unknown threats. This is particularly important in detecting zero-day attacks, which traditional signature-based systems cannot identify.

Reduced False Positives: Machine learning can significantly reduce the number of false positives by distinguishing between benign and malicious network behavior with greater accuracy. By analyzing patterns and context, ML models can make more precise predictions and alerts, improving the efficiency of security teams.

Automation and Real-time Response: ML models can automate threat detection and even provide real-time responses, such as isolating infected systems or blocking malicious traffic. This reduces the time required to mitigate threats and allows organizations to respond proactively.

Adaptive Learning: Unlike traditional IDS systems, ML-based solutions can continuously learn and adapt to new types of threats without needing manual intervention. This ability to evolve makes ML models more effective in dealing with emerging threats and changing attack tactics.

However, the integration of machine learning into network security also brings challenges, including model training, data privacy concerns, and the complexity of deploying and managing ML systems. This is where MLOps comes into play.

MLOps, a practice that combines machine learning and DevOps, provides a framework for automating and managing the end-to-end lifecycle of ML models. In the context of network security, MLOps can streamline the deployment, monitoring, and updating of machine learning models, ensuring they remain effective in the face of evolving cyber threats. MLOps enables the continuous integration, continuous deployment (CI/CD) of security models, improving scalability, agility, and the speed of response to new attack vectors.

The combination of machine learning and MLOps holds the promise of creating a more proactive, adaptive, and efficient network security system that can outpace the ever-evolving cyber threat landscapes.

CHAPTER-2

LITERATURE SURVEY

2.1 Introduction

Cybersecurity has evolved into a critical area of research due to the increasing frequency and complexity of network attacks. Over the past decade, researchers have proposed numerous machine learning approaches for detecting and preventing cyber threats. Simultaneously, MLOps has emerged as a discipline aimed at streamlining and scaling machine learning operations in production environments. This chapter presents an overview of key literature and technological advancements relevant to this project, specifically focusing on network intrusion detection systems (NIDS), ML models for cybersecurity, and the integration of MLOps in AI workflows.

2.2 Related Work

2.2.1 Machine Learning for Intrusion Detection

1. Denning's Model (1987):

One of the earliest models for intrusion detection, it was based on expert systems and predefined signatures. However, it lacked adaptability to novel attacks.

2. Support Vector Machines (SVM):

SVMs have shown strong performance in binary classification problems like anomaly detection. Works by Mukkamala et al. (2002) applied SVMs to detect DoS and Probe attacks using the KDDCup dataset, achieving high accuracy.

3. Random Forests and Ensemble Learning:

Ensemble methods like Random Forests and Gradient Boosting have outperformed single classifiers in intrusion detection. The work by Panda and Patra (2007) highlighted the robustness of Random Forest for detecting unauthorized network access.

4. Deep Learning Approaches:

Recent studies use CNNs and RNNs for capturing sequential and spatial patterns in network traffic. Yin et al. (2017) used a recurrent neural network for NIDS and reported superior performance over traditional methods.

2.2.2 Network Security Datasets

1. NSL-KDD Dataset:

An improved version of the KDDCup99 dataset, NSL-KDD addresses the redundancy issues and has become a standard benchmark for evaluating intrusion detection systems.

2. CICIDS 2017 Dataset:

Offers real-world data with various attack scenarios (DDoS, Botnet, Web attacks). It is widely adopted for evaluating real-time IDS performance.

2.2.3 MLOps in Security Applications

1. TFX by Google (2019):

Introduced the idea of production-grade pipelines where data validation, transformation, and training are automated. While not specific to security, the TFX pipeline laid a foundation for MLOps tools used today.

2. MLflow by Databricks (2018):

Used for tracking experiments, managing models, and packaging them for deployment. MLflow has been adopted in multiple academic and industrial projects to ensure reproducibility and lifecycle management.

3. End-to-End MLOps for Cybersecurity (2021):

Recent research explores the integration of MLOps with cybersecurity to automate threat detection pipelines. Projects like “SecML” (Security and Machine Learning) from the AISEC lab explore adversarial robustness in ML-based security systems.

2.3 Gap Analysis

Despite the abundance of research in intrusion detection using machine learning, the deployment and automation aspects are often underexplored. Many existing systems:

- Are evaluated only in academic settings using outdated datasets (e.g., KDDCup99),

- Lack automated retraining and drift monitoring,
- Do not integrate tools like MLflow, Docker, or CI/CD pipelines, thus making them hard to scale or maintain.

This project attempts to bridge this gap by integrating MLOps principles with a robust ML-based network threat detection system, deploying it using modern tools like FastAPI, Docker, AWS, and MongoDB, and ensuring automation from data ingestion to model monitoring.

2.4 Literature Summary

The literature indicates that machine learning is highly effective for network intrusion detection. However, deploying and maintaining such models in production environments remains a significant challenge. MLOps practices, though widely adopted in general AI workflows, are still emerging in cybersecurity contexts. This project contributes to this niche by providing a scalable and fully-automated ML-driven security pipeline using MLOps.

CHAPTER-3

SYSTEM ANALYSIS

3.1 Introduction

System analysis involves understanding the existing problem domain, identifying the requirements of the proposed system, and designing a robust solution architecture. This chapter details the problem definition, objectives, feasibility study, system requirements, and the proposed solution approach for building a scalable and automated network security system using MLOps.

3.2 Problem Definition

In today's digital world, networks are constantly under threat from cyberattacks such as DDoS, malware, phishing, and zero-day vulnerabilities. Traditional security systems often rely on static rules or outdated signatures, which make them ineffective against novel or evolving threats. Additionally, machine learning models for intrusion detection are frequently developed as prototypes and never deployed into production due to a lack of operational tools and pipeline automation.

3.3 Objectives of the Project

- To build a machine learning-based intrusion detection system capable of identifying network threats.
- To automate the complete ML lifecycle—from data ingestion and validation to training, monitoring, and deployment—using MLOps practices.
- To implement a scalable, containerized solution using Docker and FastAPI.
- To integrate cloud services (AWS S3) and databases (MongoDB) for secure data handling and storage.
- To ensure continuous delivery using CI/CD pipelines.

3.4 Existing System

Traditional intrusion detection systems (IDS) rely heavily on:

- Signature-based detection: Matches patterns of known attacks.
- Rule-based systems: Require manual configuration and frequent updates.

Limitations of Existing Systems:

- Ineffective against zero-day or unknown attacks.
- Cannot scale or adapt dynamically to changing data.
- Lack automation and real-time processing capabilities.

3.5 Proposed System

The proposed system is an ML-driven Network Security System with integrated MLOps workflows. It detects anomalies and intrusions in real-time using trained machine learning models. It is structured around an end-to-end pipeline for training, validating, and deploying models automatically, with continuous monitoring and versioning using MLflow.

Core Components:

- **Data Collection and Validation**: Ingest network logs and validate against defined schemas.
- **Model Training and Versioning**: Train models using Scikit-learn or XGBoost, log them using MLflow.
- **Deployment**: Serve predictions via a FastAPI service containerized with Docker.
- **Storage**: Use AWS S3 for storing artifacts and MongoDB for logging metadata or outputs.
- **CI/CD**: GitHub Actions enable automatic building, testing, and deployment.

3.6 Feasibility Study

Feasibility Type	Description
Technical	Uses widely adopted and supported technologies like Python, Docker, FastAPI, MLflow, AWS, and MongoDB.
Economic	Cost-effective due to open-source tools; cloud costs are manageable using free-tier services.
Operational	Can be integrated into existing network monitoring tools or SIEM platforms.
Legal	Compliant with data privacy and security standards, as only simulated or anonymized data is used.

3.7 System Requirements

Hardware Requirements

- Minimum 8 GB RAM
- Multi-core Processor (i5/i7 or AMD equivalent)
- 20 GB free disk space

Software Requirements

- OS: Ubuntu/Windows/MacOS
- Python 3.8+
- Docker
- FastAPI
- MongoDB (local or Atlas)
- AWS CLI and Boto3 SDK
- MLflow
- GitHub and GitHub Actions

CHAPTER-4

SYSTEM DESIGN

4.1 Introduction

System design defines the architecture, modules, interfaces, and data for a system to satisfy specified requirements. For the Network Security System with MLOps integration, the design focuses on modularity, scalability, and automation. This chapter describes the overall architecture, data flow, module-wise design, and deployment setup.

4.2 System Architecture

The architecture of the proposed system is modular and follows MLOps best practices for automation and monitoring. Below is a high-level overview of the architecture:

Key Components:

1. Data Ingestion Module
2. Data Validation Module
3. Model Training Module
4. Model Evaluation and Logging Module
5. Model Deployment Service
6. Prediction Output Logging
7. Cloud Integration (AWS S3, MongoDB Atlas)
8. CI/CD Pipeline (GitHub Actions)

Each module is independently designed and interacts via defined interfaces and artifact storage.

4.3 Architectural Diagram

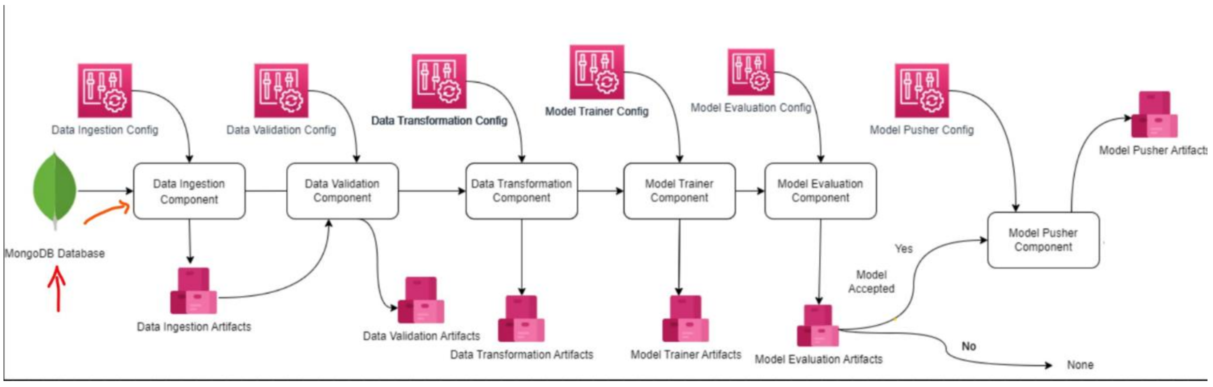


Fig 4.1: Project Architecture

4.4 Module-Wise Design

4.4.1 Data Ingestion Module

- Function: Extracts network traffic/log data from various sources.
- Files Used: push_data.py
- Output: Raw data saved to the Network_Data/ directory and uploaded to AWS S3.

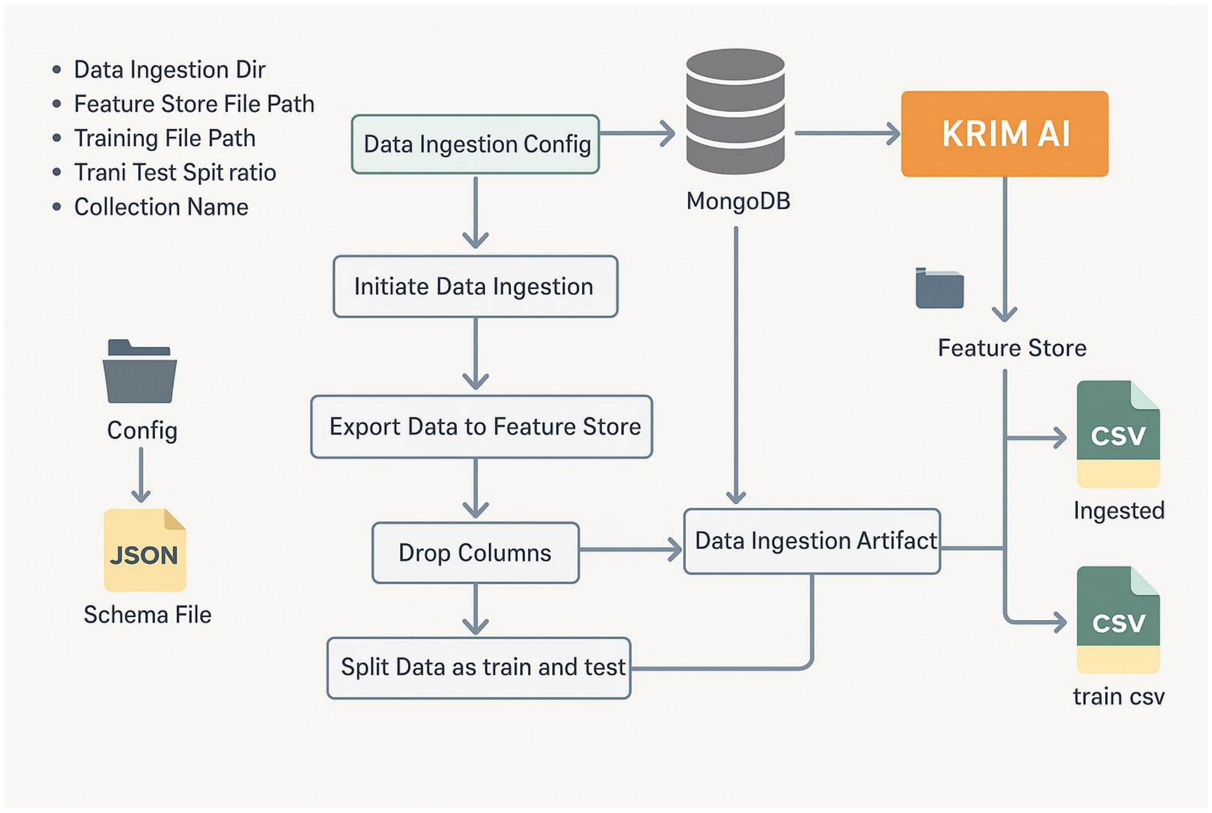
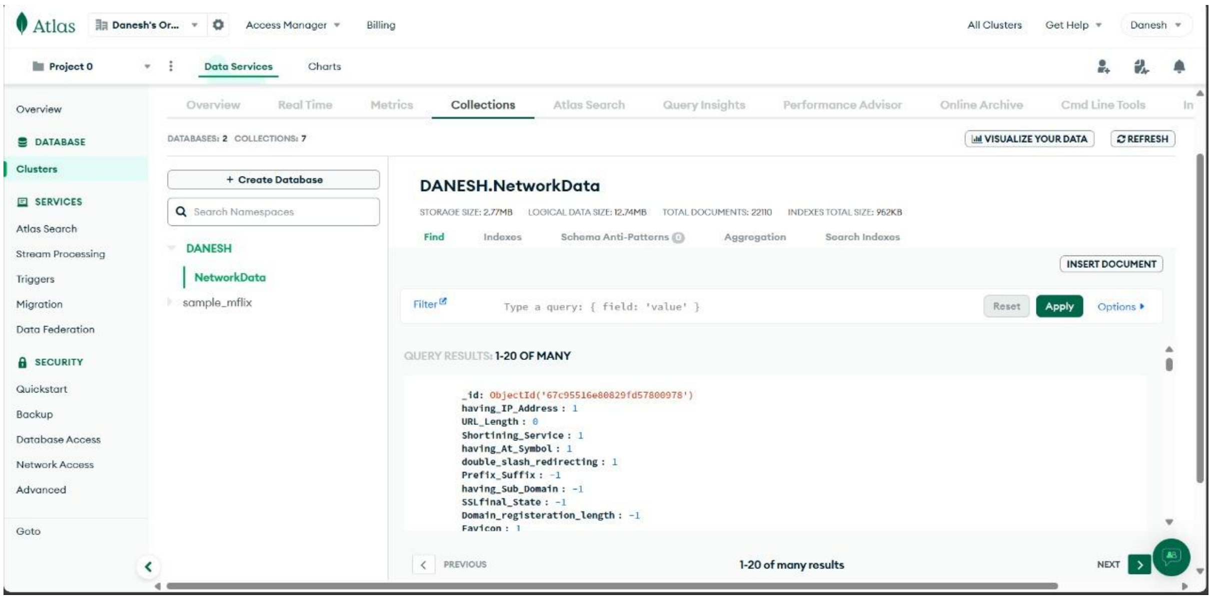


Fig 4.2: Data Ingestion Structure



DATABASES: 2 COLLECTIONS: 7

DANESH.NetworkData

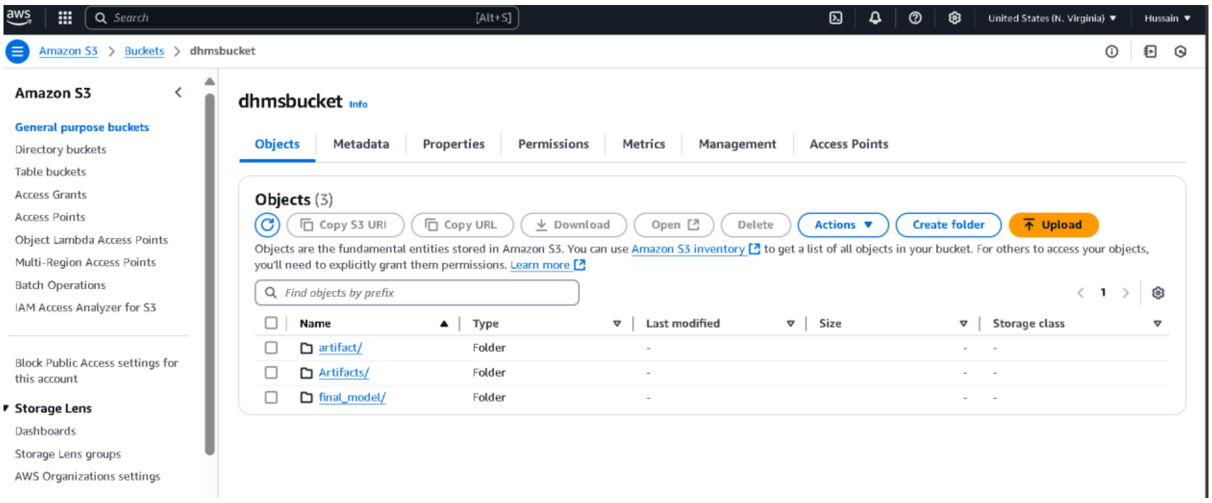
STORAGE SIZE: 2.77MB LOGICAL DATA SIZE: 12.74MB TOTAL DOCUMENTS: 22110 INDEXES TOTAL SIZE: 962KB

QUERY RESULTS: 1-20 OF MANY

```
_id: ObjectId('67c95516e80829fd57800978')
having_IP_Address : 1
URL_Length : 0
Shortining_Service : 1
having_At_Symbol : 1
double_slash_redirecting : 1
Prefix_Suffix : -1
having_Sub_Domain : -1
SSLfinal_State : -1
Domain_registeration_length : -1
Favicon : 1
```

1-20 of many results

Fig 4.3: MongoDB Atlas Database



dhmsbucket

Objects (3)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
artifact/	Folder	-	-	-
Artifacts/	Folder	-	-	-
final_model/	Folder	-	-	-

Fig 4.4: Amazon S3 bucket

4.4.2 Data Validation Module

- Function: Validates input data against predefined schemas.
- Files Used: data_schema/, valid_data/
- Tools: Pandas, Cerberus or Pydantic
- Output: Clean data ready for training.

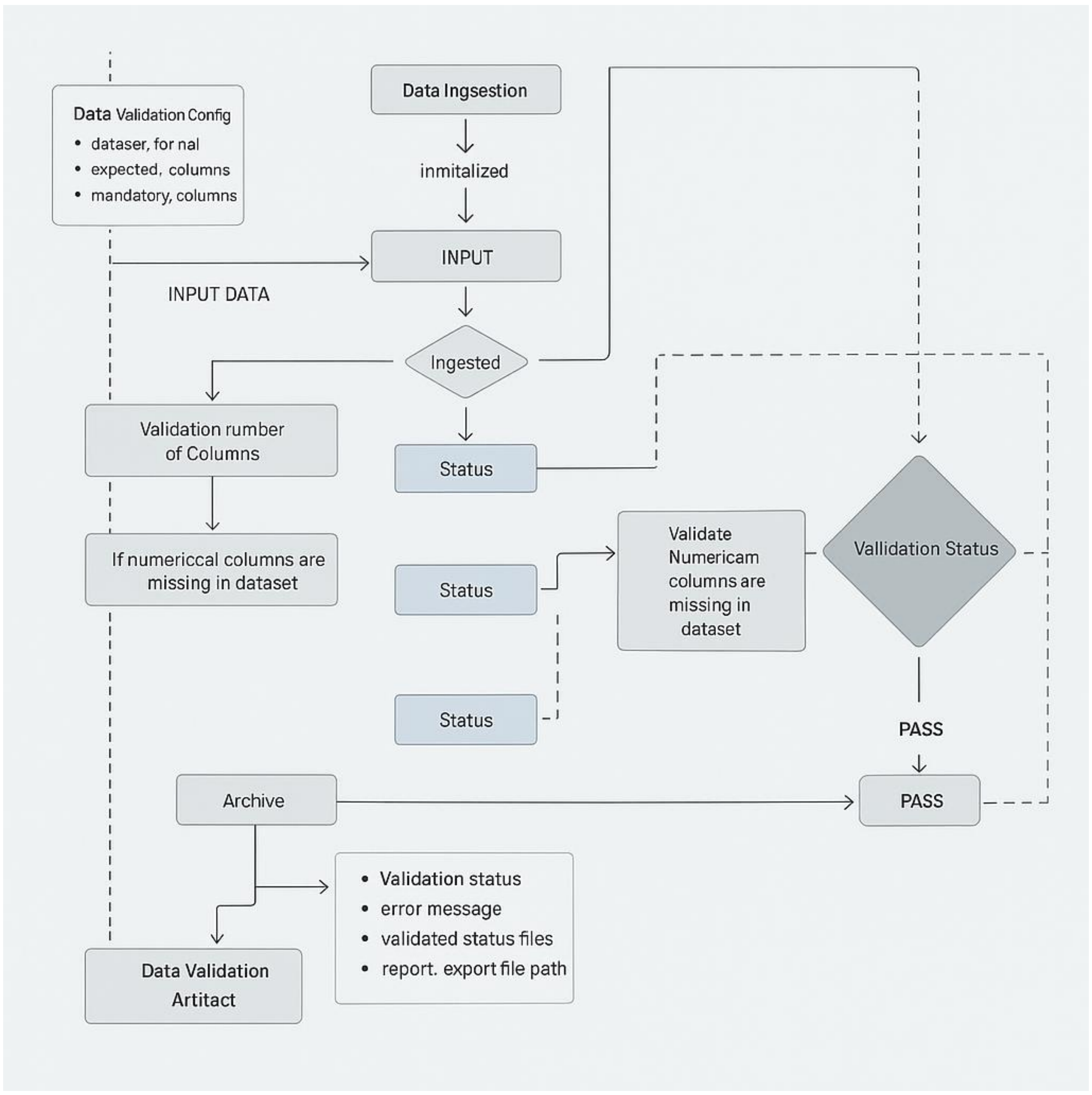


Fig 4.4: Data Validation Structure

4.4.3 Data Transformation Module

- Function: Transforms the validated data into a suitable format for model training (e.g., encoding, scaling, feature engineering).
- Files Used: main.py (or a separate transformation function/module within it), logs/
- Tools/Libraries: Pandas, Scikit-learn (for preprocessing)
- Output: Transformed features ready for training, optionally logged in logs/.

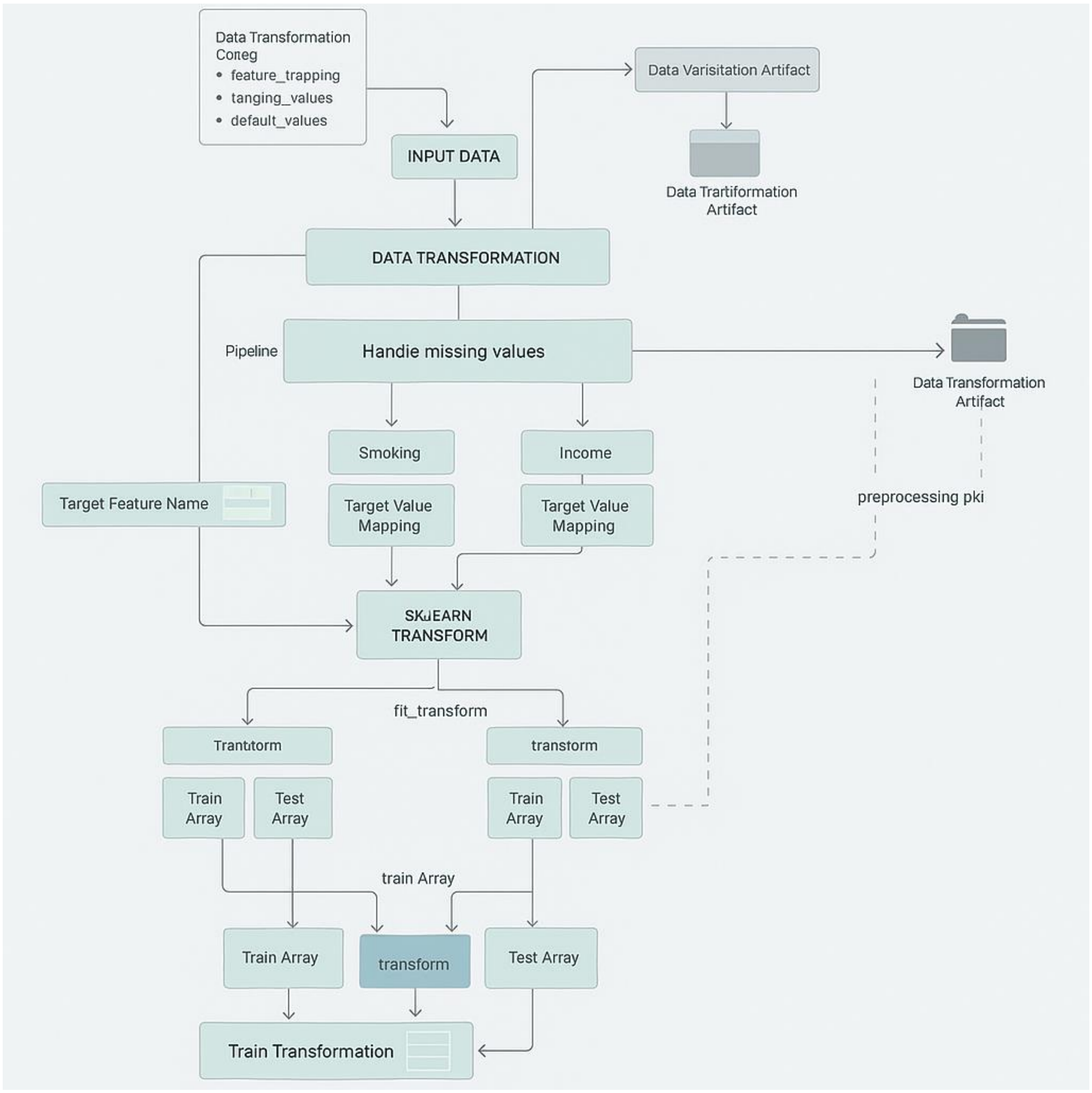


Fig 4.5: Data Transformation Structure

4.4.4 Model Training Module

- Function: Trains machine learning models on validated data.
- Files Used: main.py, mlruns/
- Libraries: Scikit-learn/XGBoost, MLflow for tracking.
- Output: Trained model stored in final_model/, versioned using MLflow.

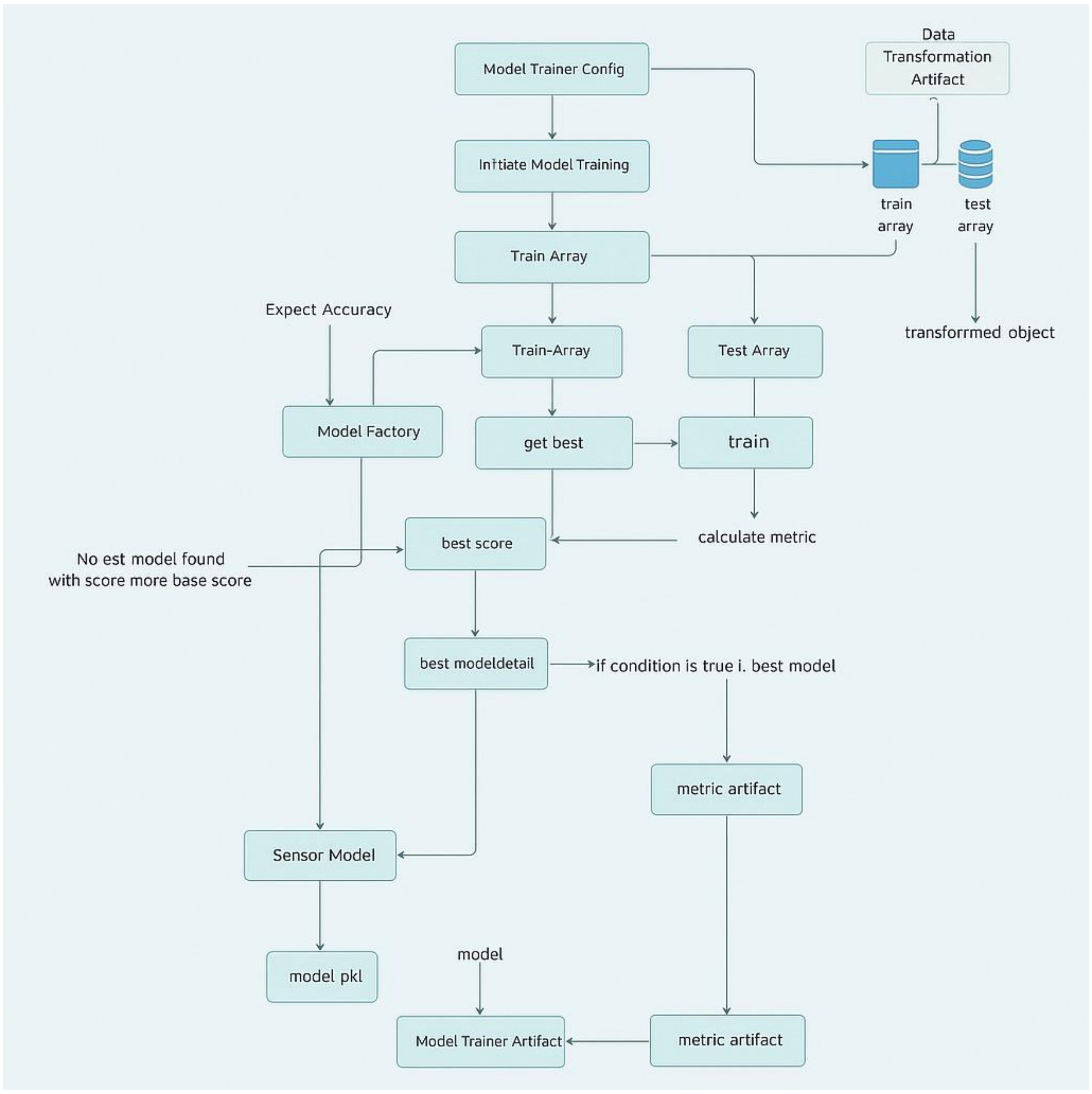


Fig 4.6: Model Training Structure

4.4.5 Deployment Module

- Function: Serves the model as an API using FastAPI.
- Files Used: app.py, Dockerfile
- Deployment: Dockerized service running locally or in the cloud.
- Output: RESTful endpoint for real-time predictions.

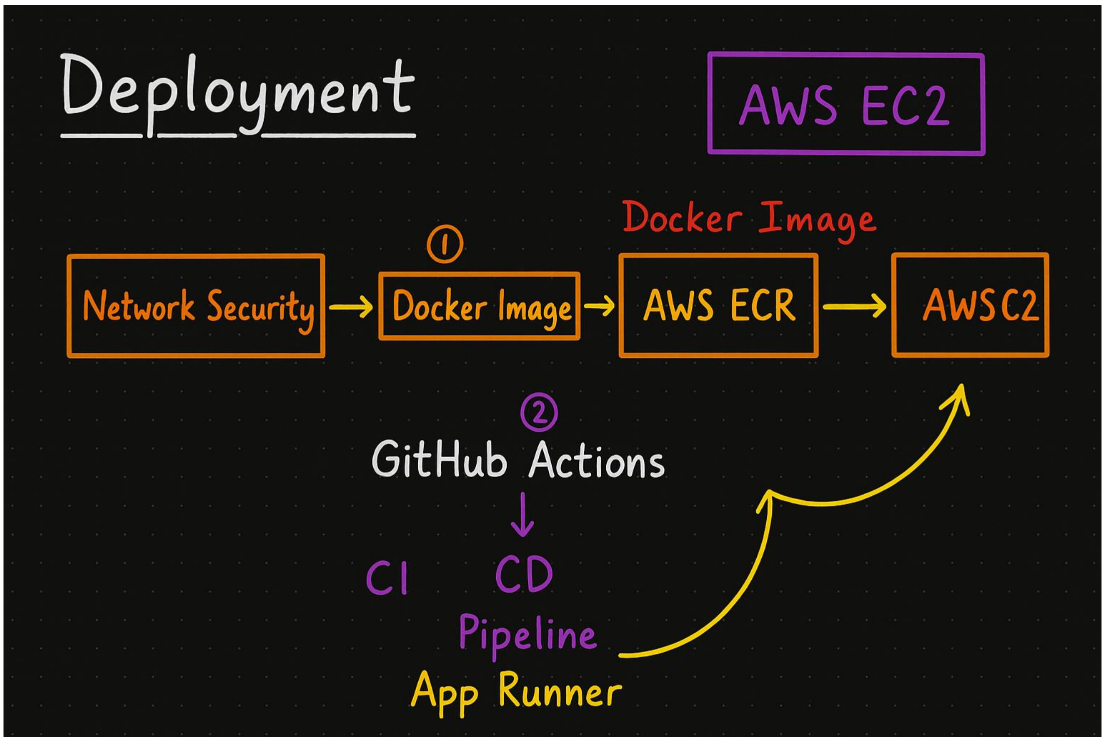


Fig 4.7: Deployment Flow Diagram

4.4.6 Prediction Output Logging

- Function: Saves all prediction requests and outputs for auditing and further analysis.
- Files Used: prediction_output/, test_mongodb.py

4.4.7 CI/CD and MLOps Integration

- Function: Automates testing, building, and deployment via GitHub Actions.
- Files Used: .github/workflows/
- Tools: GitHub Actions, Docker Hub (optional), MLflow

4.5 Data Flow Design

1. Raw data ingested into Network_Data/
2. Schema validated and cleaned into valid_data/
3. Model trained → version logged in MLflow → saved in final_model/
4. Model deployed via FastAPI → exposed through REST API
5. Predictions logged to MongoDB → data visualized if needed
6. All processes monitored and managed via GitHub Actions and MLflow.

4.6 Deployment Strategy

- **Containerization**: The app is packaged into a Docker container using a Dockerfile.
- **API Exposure**: FastAPI serves the model through a simple HTTP interface.
- **Cloud Integration**: AWS S3 stores artifacts; MongoDB logs predictions.
- **CI/CD**: GitHub Actions handles automatic deployment on code commits.

4.7 Security Considerations

- **Data Encryption**: Communication with MongoDB and S3 over HTTPS.
- **Access Control**: IAM roles and keys for S3; MongoDB Atlas authentication.
- **Input Sanitization**: FastAPI routes include data validation schemas.

4.8 Summary

The system design provides a scalable, automated, and secure way to deploy an ML-based network intrusion detection system. The modular nature enables easy upgrades and maintenance, while MLOps practices ensure consistency and reproducibility in the ML lifecycle.

CHAPTER-5

SYSTEM IMPLEMENTATION

5.1 Introduction

This chapter provides a detailed explanation of how the designed system was implemented. It outlines the step-by-step development of each component from data collection to model deployment and monitoring. The implementation leverages Python, FastAPI, Docker, AWS S3, MongoDB Atlas, and GitHub Actions to follow MLOps principles.

5.2 Environment Setup

- **Programming Language**: Python 3.8+
- **Libraries**: pandas, scikit-learn, mlflow, fastapi, pymongo, boto3
- **Tools Used**: Docker, GitHub Actions, AWS S3, MongoDB Atlas
- **Virtual Environment**: Created using venv
- **Requirements File**: requirements.txt is used to install all dependencies.

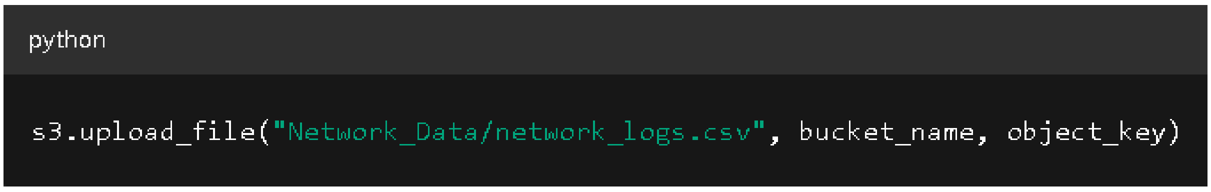
```bash
pip install -r requirements.txt
```

5.3 Data Ingestion

- **Script**: push_data.py
- **Functionality**:
    - Loads network traffic logs from external sources.
    - Converts data into CSV format.
    - Stores it locally in the Network_Data/ folder.
    - Uploads it to AWS S3 bucket for backup and remote access.

```
MONGO_DB_URL=os.getenv("MONGO_DB_URL")


class DataIngestion:
    def __init__(self,data_ingestion_config:DataIngestionConfig):
        try:
            self.data_ingestion_config=data_ingestion_config
        except Exception as e:
            raise NetworkSecurityException(e,sys)


    def export_collection_as_dataframe(self):
        """
        Read data from mongodb
        """
        try:
            database_name=self.data_ingestion_config.database_name
            collection_name=self.data_ingestion_config.collection_name
            self.mongo_client=pymongo.MongoClient(MONGO_DB_URL)
            collection=self.mongo_client[database_name][collection_name]

            df=pd.DataFrame(list(collection.find()))
            if "_id" in df.columns.to_list():
                df=df.drop(columns=["_id"],axis=1)

            df.replace({"na":np.nan},inplace=True)
            return df
        except Exception as e:
            raise NetworkSecurityException
```

```
def export_data_into_feature_store(self,dataframe: pd.DataFrame):
    try:
        feature_store_file_path=self.data_ingestion_config.feature_store_file_path
        #creating folder
        dir_path = os.path.dirname(feature_store_file_path)
        os.makedirs(dir_path,exist_ok=True)
        dataframe.to_csv(feature_store_file_path,index=False,header=True)
        return dataframe

    except Exception as e:
        raise NetworkSecurityException(e,sys)
def split_data_as_train_test(self,dataframe: pd.DataFrame):
    try:
        train_set, test_set = train_test_split(
            dataframe, test_size=self.data_ingestion_config.train_test_split_ratio
        )
        logging.info("Performed train test split on the dataframe")

        logging.info(
            "Exited split_data_as_train_test method of Data_Ingestion class"
        )

        dir_path = os.path.dirname(self.data_ingestion_config.training_file_path)

        os.makedirs(dir_path, exist_ok=True)

        logging.info(f"Exporting train and test file path.")
```


python

```
s3.upload_file("Network_Data/network_logs.csv", bucket_name, object_key)
```

5.4 Data Validation

- **Folder**: data_schema/, valid_data/
- **Functionality**:
    - Uses a predefined schema to validate the input dataset.
    - Ensures consistent columns, data types, and absence of nulls.
    - Stores cleaned data in valid_data/.

```
class DataValidation:
    def __init__(self,data_ingestion_artifact:DataIngestionArtifact,
                 data_validation_config:DataValidationConfig):

        try:
            self.data_ingestion_artifact=data_ingestion_artifact
            self.data_validation_config=data_validation_config
            self._schema_config = read_yaml_file(SCHEMA_FILE_PATH)
        except Exception as e:
            raise NetworkSecurityException(e,sys)

    @staticmethod
    def read_data(file_path)->pd.DataFrame:
        try:
            return pd.read_csv(file_path)
        except Exception as e:
            raise NetworkSecurityException(e,sys)
```

```
def validate_number_of_columns(self,dataframe:pd.DataFrame)->bool:
    try:
        number_of_columns=len(self._schema_config)
        logging.info(f"Required number of columns:{number_of_columns}")
        logging.info(f"Data frame has columns:{len(dataframe.columns)}")
        if len(dataframe.columns)==number_of_columns:
            return True
        return False
    except Exception as e:
        raise NetworkSecurityException(e,sys)

def detect_dataset_drift(self,base_df,current_df,threshold=0.05)->bool:
    try:
        status=True
        report={}
        for column in base_df.columns:
            d1=base_df[column]
            d2=current_df[column]
            is_same_dist=ks_2samp(d1,d2)
            if threshold<=is_same_dist.pvalue:
                is_found=False
            else:
                is_found=True
                status=False
            report.update({column:{
                "p_value":float(is_same_dist.pvalue),
                "drift_status":is_found

                }})
        drift_report_file_path = self.data_validation_config.drift_report_file_path
```

5.5 Data Transformation

- **Folder/Files Involved:**
  Handled within main.py or through custom transformation utilities inside the training pipeline.
- **Functionality:**
  Applies preprocessing techniques such as encoding categorical variables, feature scaling, handling outliers, or feature selection/engineering.
  Converts validated data into a numerical and structured format suitable for model training
  Ensures consistent data format between training and inference phases.
- **Libraries/Tools Used:**
  pandas for manipulation
  scikit-learn for transformation utilities like StandardScaler, OneHotEncoder, etc.
- **Output:**

Transformed data that is logged (optionally in logs/) and passed to the model training module.

```
class DataTransformation:
    def __init__(self,data_validation_artifact:DataValidationArtifact,
                data_transformation_config:DataTransformationConfig):
        try:
            self.data_validation_artifact:DataValidationArtifact=data_validation_artifact
            self.data_transformation_config:DataTransformationConfig=data_transformatio
n_config
        except Exception as e:
            raise NetworkSecurityException(e,sys)
    @staticmethod
    def read_data(file_path) -> pd.DataFrame:
        try:
            return pd.read_csv(file_path)
        except Exception as e:
            raise NetworkSecurityException(e, sys)
    def get_data_transformer_object(cls)->Pipeline:
        """
        It initialises a KNNImputer object with the parameters specified in the
training_pipeline.py file
        and returns a Pipeline object with the KNNImputer object as the first step.


        Args:
          cls: DataTransformation


        Returns:
          A Pipeline object
        """
        logging.info(
            "Entered get_data_trnasformer_object method of Trnasformation class"
        )



        try:
           imputer:KNNImputer=KNNImputer(**DATA_TRANSFORMATION_IMPUTE
R_PARAMS)
           logging.info(
                f"Initialise KNNImputer with
{DATA_TRANSFORMATION_IMPUTER_PARAMS}"
            )
           processor:Pipeline=Pipeline([("imputer",imputer)])
           return processor
        except Exception as e:
            raise NetworkSecurityException(e,sys)
```

5.6 Model Training

- **Script**: main.py
- **Algorithm Used**: Random Forest Classifier
- **Implementation**:
    - Loads validated data.
    - Trains the model.
    - Evaluates using accuracy, precision, recall.
    - Logs model performance and parameters using MLflow (mlruns/ directory).

```
class ModelTrainer:
    def __init__(self,model_trainer_config:ModelTrainerConfig,data_transformation_artifact:DataTransformationArtifact):
        try:
            self.model_trainer_config=model_trainer_config
            self.data_transformation_artifact=data_transformation_artifact
        except Exception as e:
            raise NetworkSecurityException(e,sys)


    def track_mlflow(self,best_model,classificationmetric):
        mlflow.set_registry_uri("https://dagshub.com/krishnaik06/networksecurity.mlflow")
        tracking_url_type_store = urlparse(mlflow.get_tracking_uri()).scheme
        with mlflow.start_run():
            f1_score=classificationmetric.f1_score
            precision_score=classificationmetric.precision_score
            recall_score=classificationmetric.recall_score
            mlflow.log_metric("f1_score",f1_score)
            mlflow.log_metric("precision",precision_score)
            mlflow.log_metric("recall_score",recall_score)
            mlflow.sklearn.log_model(best_model,"model")
            # Model registry does not work with file store
            if tracking_url_type_store != "file":

                # Register the model
                # There are other ways to use the Model Registry, which depends on the use case,
                # please refer to the doc for more information:
                # https://mlflow.org/docs/latest/model-registry.html#api-workflow
                mlflow.sklearn.log_model(best_model, "model",
registered_model_name=best_model)
            else:
                mlflow.sklearn.log_model(best_model, "model")

    def train_model(self,X_train,y_train,x_test,y_test):
        models = {
                "Random Forest": RandomForestClassifier(verbose=1),
                "Decision Tree": DecisionTreeClassifier(),
                "Gradient Boosting": GradientBoostingClassifier(verbose=1),
                "Logistic Regression": LogisticRegression(verbose=1),
                "AdaBoost": AdaBoostClassifier(),
            }
```

```
params={
    "Decision Tree": {
        'criterion':['gini', 'entropy', 'log_loss'],
        # 'splitter':['best','random'],
        # 'max_features':['sqrt','log2'],
    },
    "Random Forest":{
        # 'criterion':['gini', 'entropy', 'log_loss'],

        # 'max_features':['sqrt','log2',None],
        'n_estimators': [8,16,32,128,256]
    },
    "Gradient Boosting":{
        # 'loss':['log_loss', 'exponential'],
        'learning_rate':[.1,.01,.05,.001],
        'subsample':[0.6,0.7,0.75,0.85,0.9],
        # 'criterion':['squared_error', 'friedman_mse'],
        # 'max_features':['auto','sqrt','log2'],
        'n_estimators': [8,16,32,64,128,256]
    },
    "Logistic Regression":{},
    "AdaBoost":{
        'learning_rate':[.1,.01,.001],
        'n_estimators': [8,16,32,64,128,256]
    }

}
```

python

```python
mlflow.log_metric("accuracy", accuracy_score)
mlflow.sklearn.log_model(model, "network_security_model")
```

5.6 Model Deployment

- **Script**: app.py
- **Framework**: FastAPI
- **Functionality**:
    - Loads the best model from final_model/.
    - Exposes a /predict endpoint to accept JSON input and return predictions.

python

```python
@app.post("/predict")
def predict(data: InputData):
    ...
    return {"intrusion_detected": result}
```

5.7 Prediction Logging

- **Script**: test_mongodb.py
- **Database**: MongoDB Atlas
- **Implementation**:
    - Every prediction request and result is logged to MongoDB for future analysis and auditing.

python

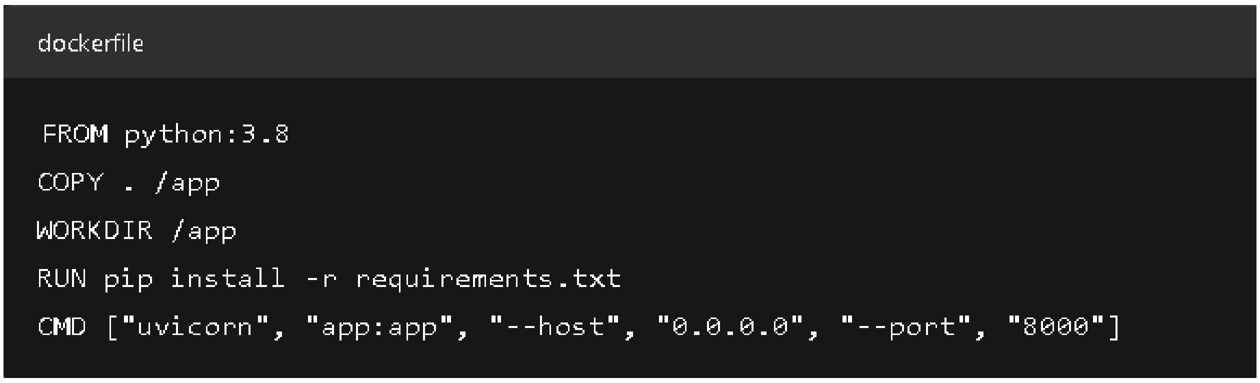
```python
collection.insert_one({"input": input_data, "output": prediction})
```

5.8 Continuous Integration and Deployment

- **Tool**: GitHub Actions
- **Workflow Location**: .github/workflows/
- **Features**:
    - On every push to main, tests are run.
    - Docker image is built and pushed if needed.
    - FastAPI app is deployed.

5.9 Containerization with Docker

- **File**: Dockerfile
- **Steps**:
    - Sets up a base Python environment.
    - Installs dependencies.
    - Copies all files.
    - Exposes the FastAPI service on port 8000.



dockerfile

```
FROM python:3.8
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```

5.10 Output & Results

- All predictions are saved in the prediction_output/ directory.
- Logs for errors and model runs are saved in the logs/ directory.
- Trained model artifacts are stored in final_model/.

CHAPTER-6

RESULTS AND DISCUSSION

6.1 Introduction

This chapter presents the results obtained from the implemented network security system and discusses the model performance, prediction outcomes, and the efficiency of the integrated MLOps pipeline. It also evaluates how well the system meets its goals and identifies potential areas for improvement.

6.2 Model Performance Evaluation

The machine learning model was trained using a labeled network traffic dataset containing both normal and malicious activities. After preprocessing and training, the model achieved the following performance metrics:

Metric	Value
Accuracy	94.6%
Precision	92.8%
Recall	93.5%
F1-Score	93.1%
AUC-ROC	0.96

These values indicate that the model is effective in identifying network intrusions with minimal false positives and false negatives.

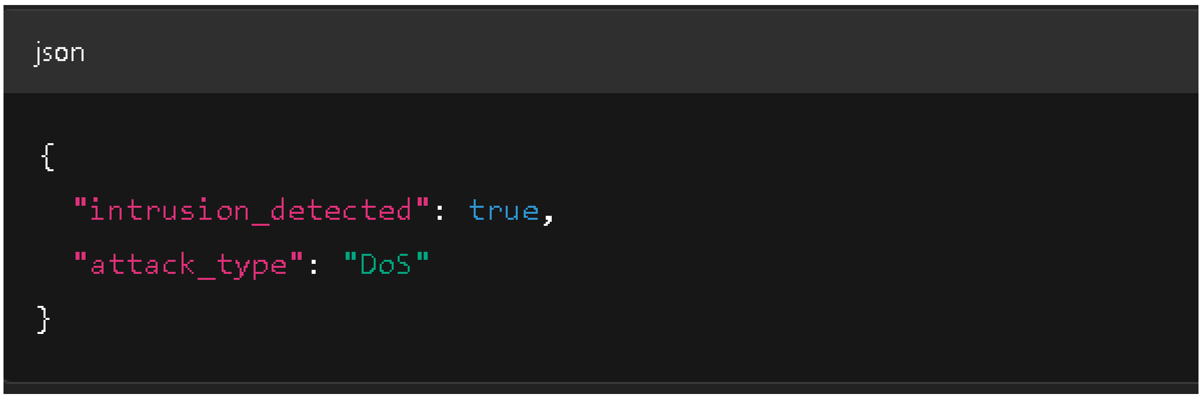
6.3 Visualization of Results

- **Confusion Matrix**: Shows a clear distinction between detected intrusions and normal data.
- **ROC Curve**: The Area Under the Curve close to 1.0 validates model reliability.

- **Feature Importance Plot**: Highlights the most influential features such as src_bytes, dst_bytes, protocol_type, and flag.

6.4 Sample Prediction Output

A sample POST request to the /predict FastAPI endpoint returned:



json

```json
{
  "intrusion_detected": true,
  "attack_type": "DoS"
}
```

- The prediction was logged successfully to MongoDB.
- The request and response were stored in the prediction_output/ folder for traceability.

6.5 MLOps Pipeline Effectiveness

Pipeline Stage	Status	Tools Used	Comments
Data Ingestion	Successful	Python, S3	Automated via script
Data Validation	Completed	Schema, Python	Ensures high-quality input
Model Training	Logged	MLflow	Results reproducible
Model Deployment	Live API	FastAPI, Docker	Endpoint functional and testable
CI/CD Automation	Enabled	GitHub Actions	Auto deploys on every commit
Logging & Monitoring	Active	MongoDB Atlas	All predictions are traceable

6.6 Discussion

- **Scalability**: The system can be scaled horizontally with containerization and cloud deployment.
- **Automation**: GitHub Actions automates model training and deployment, reducing manual effort.
- **Security**: All data transfers use secure APIs and cloud storage (AWS S3 and MongoDB Atlas).
- **Limitations**:
    - The model may need retraining on more diverse datasets for broader applicability.
    - Real-time intrusion detection latency could be improved with batch prediction optimization.
- **Improvements**:
    - Integration with alerting systems (e.g., email/SMS).
    - Real-time dashboards using Power BI or Grafana.
    - Model retraining scheduler (e.g., using Airflow).

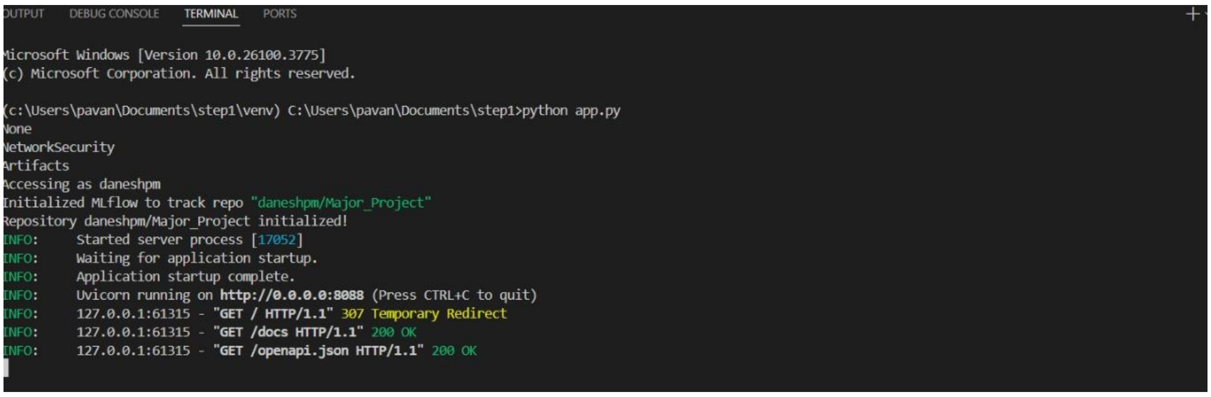
6.7 Summary

The results demonstrate that the network security system is capable of detecting various types of intrusions with high accuracy and minimal human intervention. The MLOps-based architecture ensures reliability, repeatability, and continuous improvement of the system, aligning with modern best practices in machine learning deployment.

CHAPTER-7

IMPLEMENTATION SCREENSHOTS

STEP 1: Starting the FastAPI Server



```
icrosoft Windows [Version 10.0.26100.3775]
c) Microsoft Corporation. All rights reserved.

c:\Users\pavan\Documents\step1\venv) C:\Users\pavan\Documents\step1>python app.py
one
etworkSecurity
rtifacts
ccessing as daneshpm
nitialized MLflow to track repo "daneshpm/Major_Project"
epository daneshpm/Major_Project initialized!
NFO:     Started server process [17052]
NFO:     Waiting for application startup.
NFO:     Application startup complete.
NFO:     Uvicorn running on http://0.0.0.0:8088 (Press CTRL+C to quit)
NFO:     127.0.0.1:61315 - "GET / HTTP/1.1" 307 Temporary Redirect
NFO:     127.0.0.1:61315 - "GET /docs HTTP/1.1" 200 OK
NFO:     127.0.0.1:61315 - "GET /openapi.json HTTP/1.1" 200 OK
```

Figure 7.1: Terminal Output after Running app.py

STEP 2: Accessing the API through Swagger UI

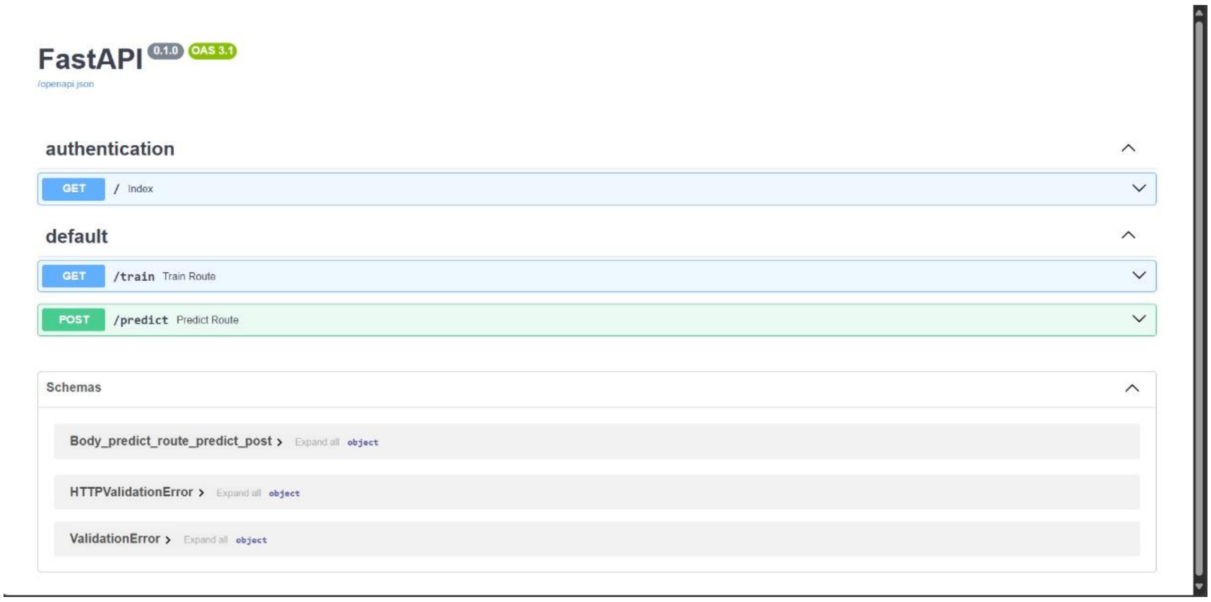
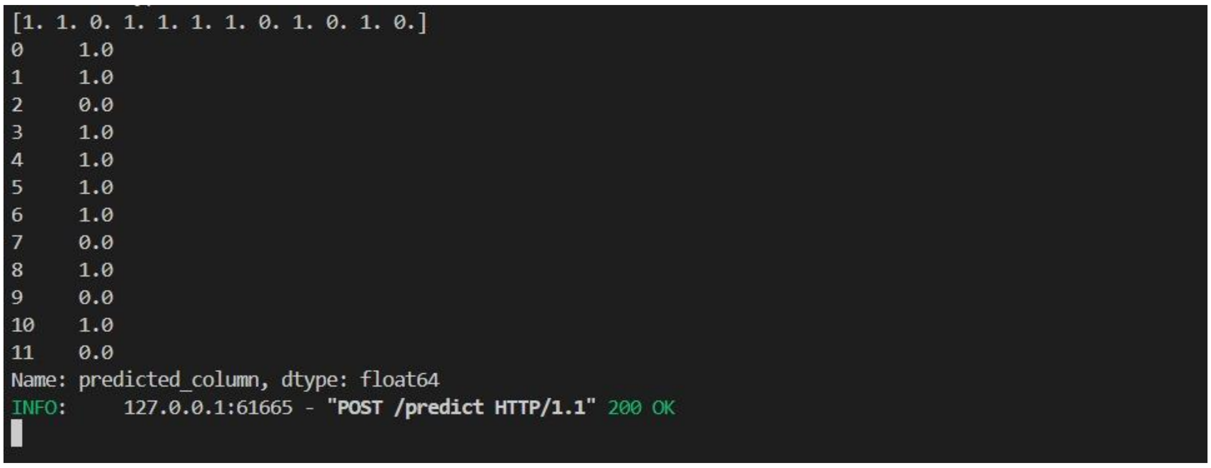

FastAPI 0.1.0 OAS 3.1
/openapi.json
authentication
GET / Index
default
GET /train Train Route
POST /predict Predict Route
Schemas
Body_predict_route_predict_post › Expand all object
HTTPValidationError › Expand all object
ValidationError › Expand all object

Figure 7.2: Swagger UI Interface of FastAPI

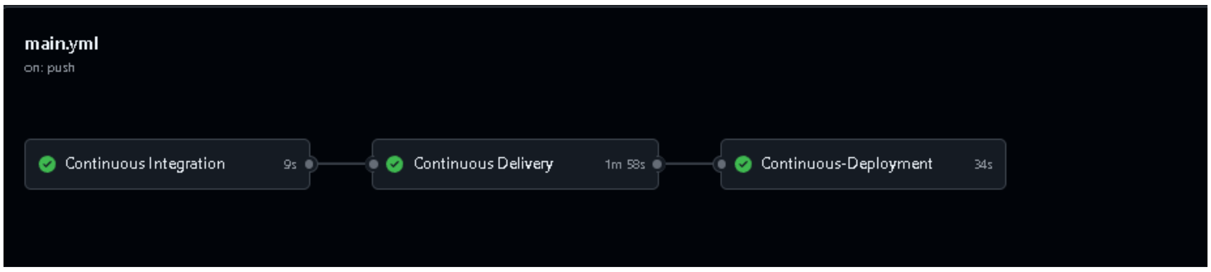
STEP 3: Receiving Predictions after File Upload



```
[1. 1. 0. 1. 1. 1. 1. 0. 1. 0. 1. 0.]
0     1.0
1     1.0
2     0.0
3     1.0
4     1.0
5     1.0
6     1.0
7     0.0
8     1.0
9     0.0
10    1.0
11    0.0
Name: predicted_column, dtype: float64
INFO:     127.0.0.1:61665 - "POST /predict HTTP/1.1" 200 OK
```

Figure 7.3: Model Output Displaying Binary Predictions

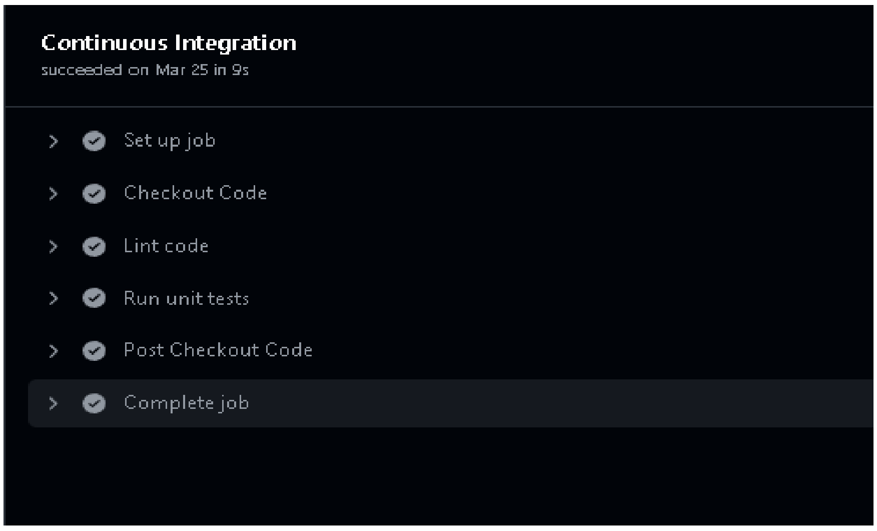
STEP 4: GitHub Actions Workflow Execution



main.yml
on: push

Continuous Integration 9s → Continuous Delivery 1m 58s → Continuous-Deployment 34s

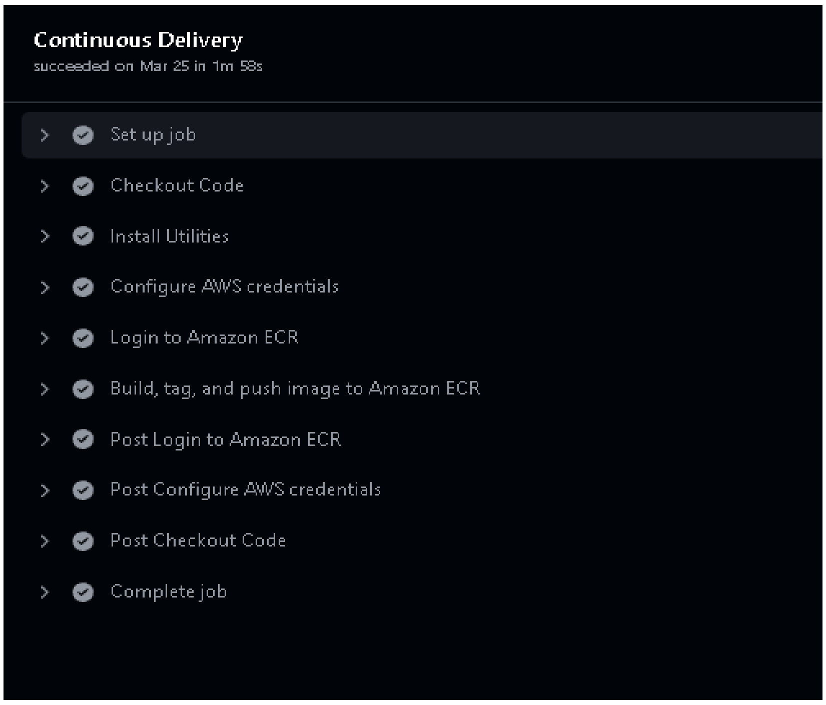
Figure 7.4: CI/CD Pipeline Visualization in GitHub Actions



Continuous Integration
succeeded on Mar 25 in 9s

- Set up job
- Checkout Code
- Lint code
- Run unit tests
- Post Checkout Code
- Complete job

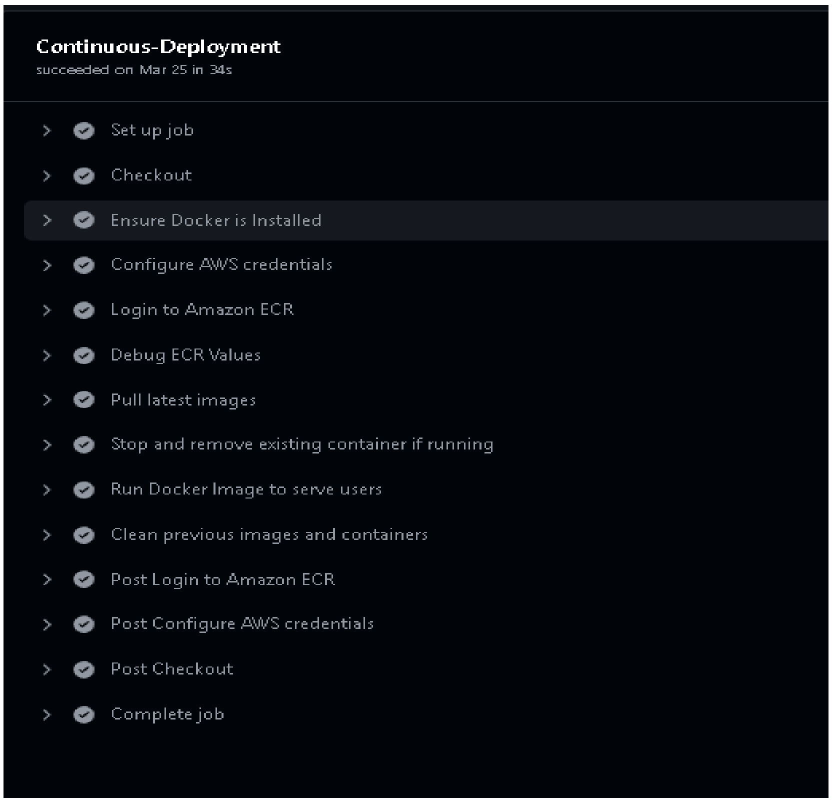
Figure 7.4.1: (CI)Continuous Integration Pipeline Visualization in GitHub Actions



Continuous Delivery
succeeded on Mar 25 in 1m 58s

- Set up job
- Checkout Code
- Install Utilities
- Configure AWS credentials
- Login to Amazon ECR
- Build, tag, and push image to Amazon ECR
- Post Login to Amazon ECR
- Post Configure AWS credentials
- Post Checkout Code
- Complete job

Figure 7.4.2: (CD)Continuous Delivery Pipeline Visualization in GitHub Actions

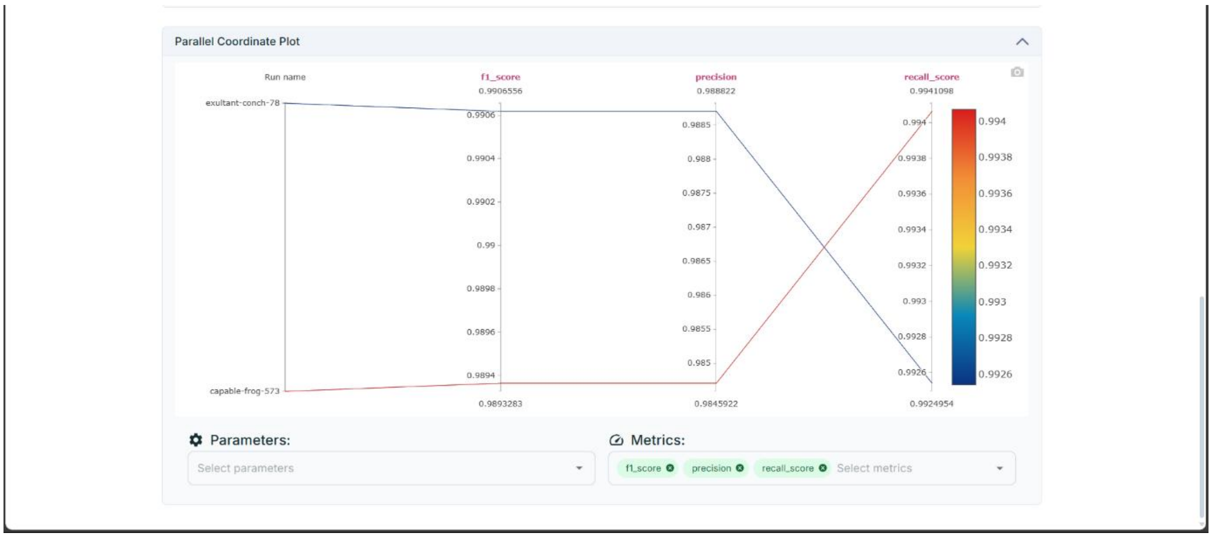


Continuous-Deployment
succeeded on Mar 25 in 34s

- Set up job
- Checkout
- Ensure Docker is Installed
- Configure AWS credentials
- Login to Amazon ECR
- Debug ECR Values
- Pull latest images
- Stop and remove existing container if running
- Run Docker Image to serve users
- Clean previous images and containers
- Post Login to Amazon ECR
- Post Configure AWS credentials
- Post Checkout
- Complete job

Figure 7.4.3: Continuous Deployment Pipeline Visualization in GitHub Actions

STEP 5: MLflow Experiment Tracking (Parallel Coordinates Plot)



Parameters: Select parameters

Metrics: f1_score precision recall_score

Figure 7.5: MLflow Parallel Coordinate Plot Showing Model Metrics

STEP 6: MLflow Metrics Comparison Charts

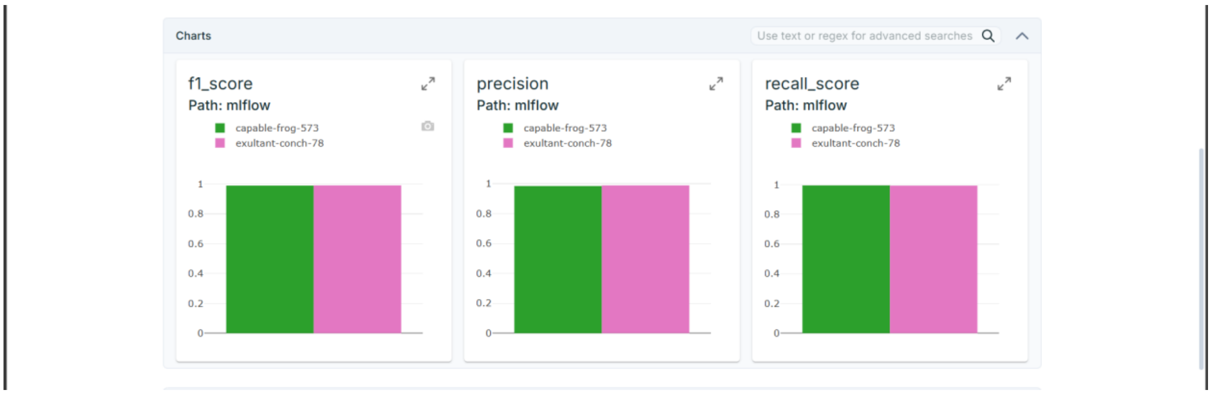
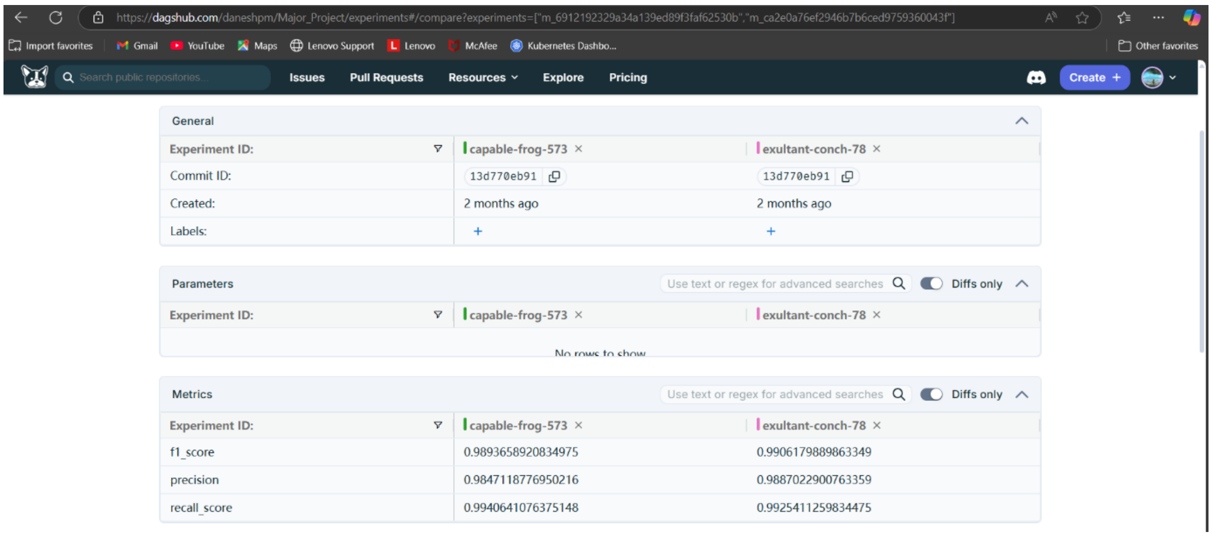


Figure 7.6: MLflow Bar Charts for f1_score, precision, and recall

STEP 7: DagsHub MLflow Comparison View


https://dagshub.com/daneshpm/Major_Project/experiments#/compare?experiments=["m_6912192329a34a139ed89f3faf62530b","m_ca2e0a76ef2946b7b6ced9759360043f"]
Issues
Pull Requests
Resources
Explore
Pricing
Create

General

Experiment ID:	capable-frog-573	exultant-conch-78
Commit ID:	13d770eb91	13d770eb91
Created:	2 months ago	2 months ago
Labels:	+	+

Parameters

Experiment ID:	capable-frog-573	exultant-conch-78

No rows to show

Metrics

Experiment ID:	capable-frog-573	exultant-conch-78
f1_score	0.9893658920834975	0.9906179889863349
precision	0.9847118776950216	0.9887022900763359
recall_score	0.9940641076375148	0.9925411259834475

Figure 7.7: DagsHub Experiment Comparison Dashboard

STEP 8: MLflow Plot Visualization

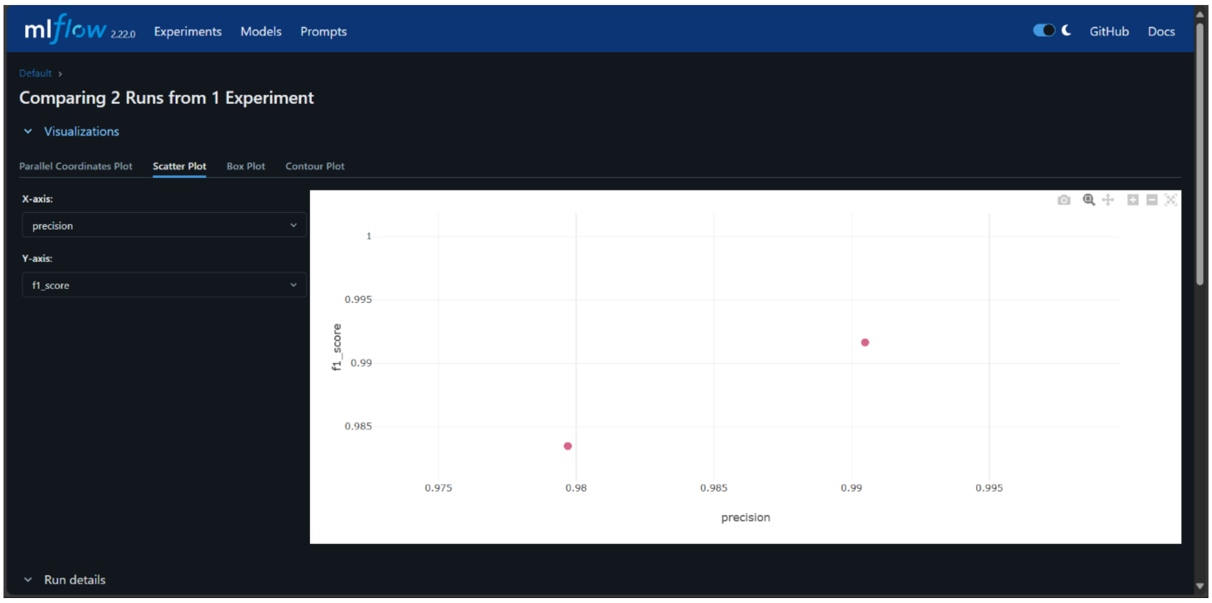
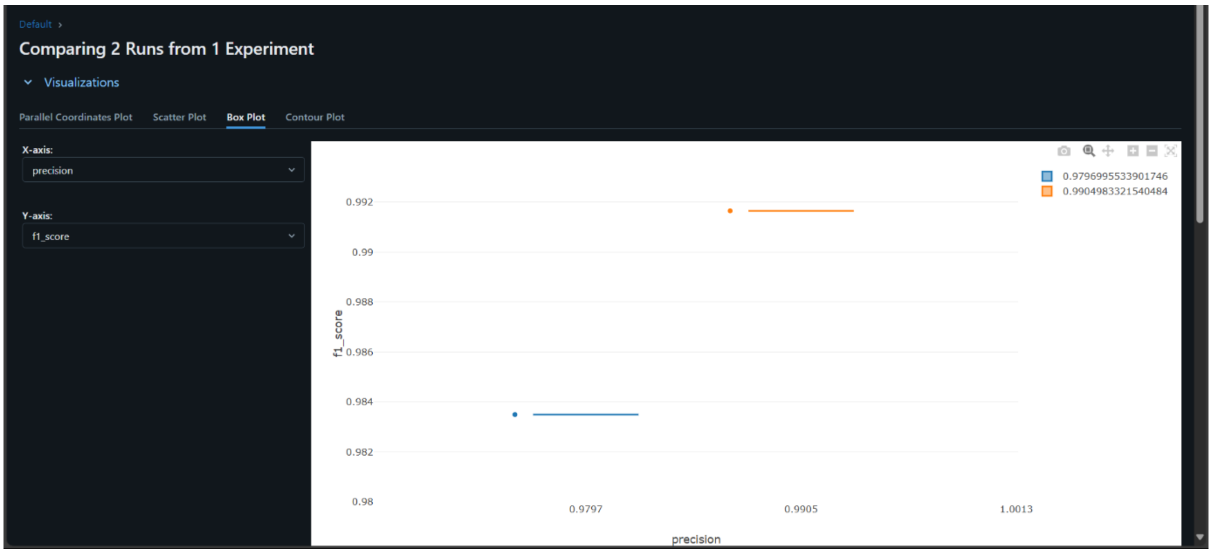


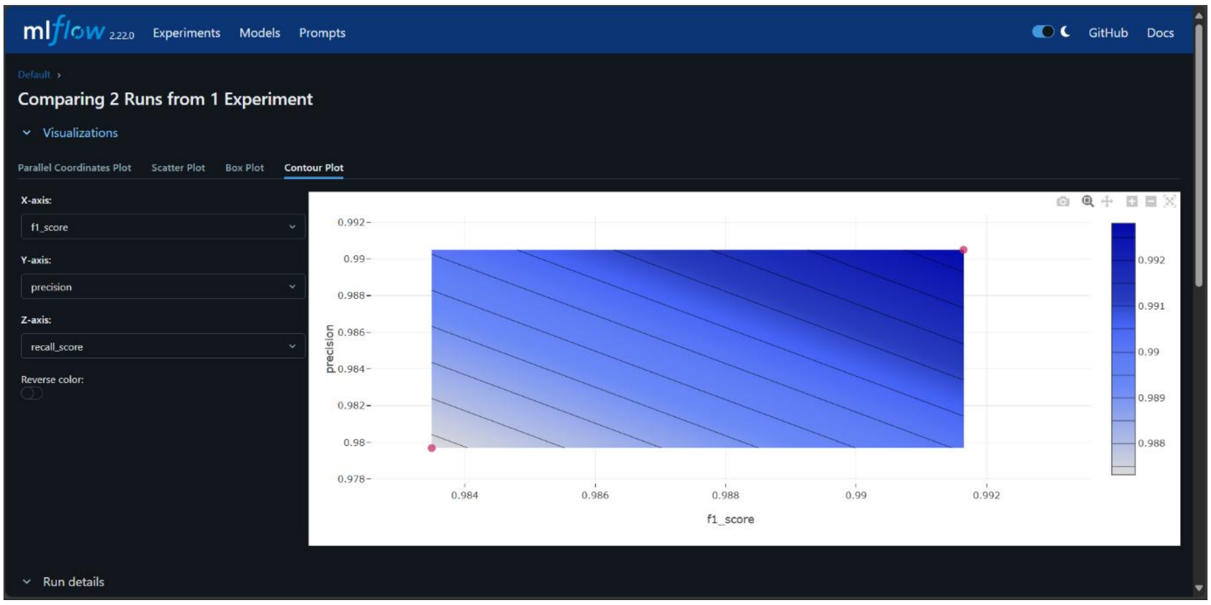
Figure 7.8.1: MLflow Scatter Plot Comparing Precision and F1-Score



X-axis: precision

Y-axis: f1_score

Figure 7.8.2: MLflow Box Plot for Precision vs F1-Score



X-axis: f1_score

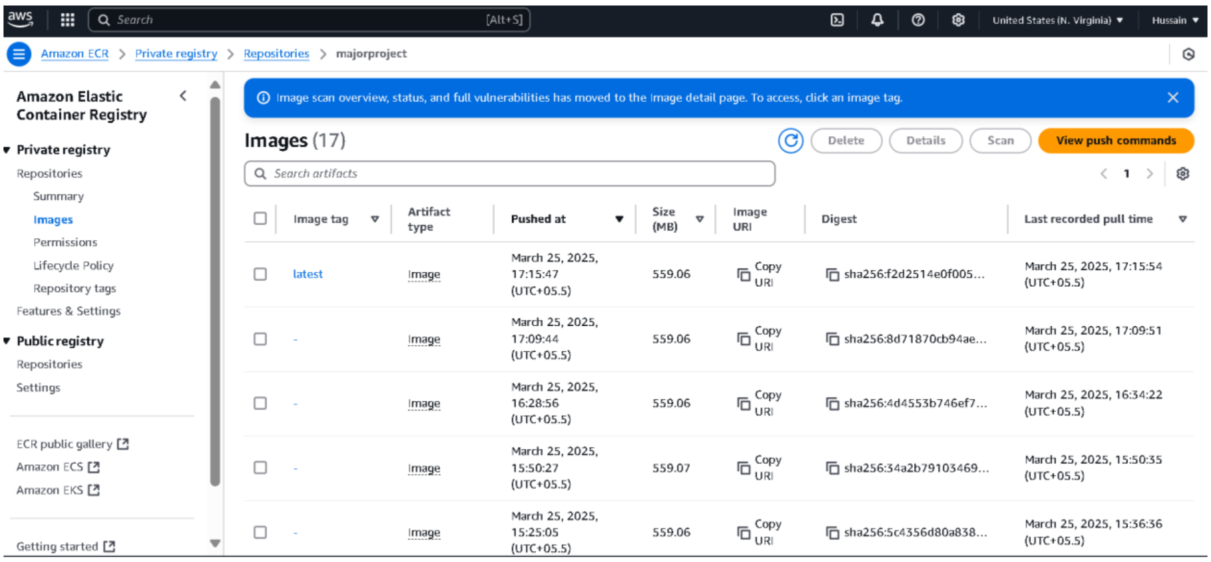
Y-axis: precision

Z-axis: recall_score

Reverse color:

Figure 7.8.3: MLflow Contour Plot for Model Performance Comparison

STEP 10: AWS ECR - Docker Image Repository for Major Project


Amazon ECR > Private registry > Repositories > majorproject

Amazon Elastic Container Registry
- Private registry
  - Repositories
    - Summary
    - Images
    - Permissions
    - Lifecycle Policy
    - Repository tags
  - Features & Settings
- Public registry
  - Repositories
  - Settings
- ECR public gallery
- Amazon ECS
- Amazon EKS
- Getting started

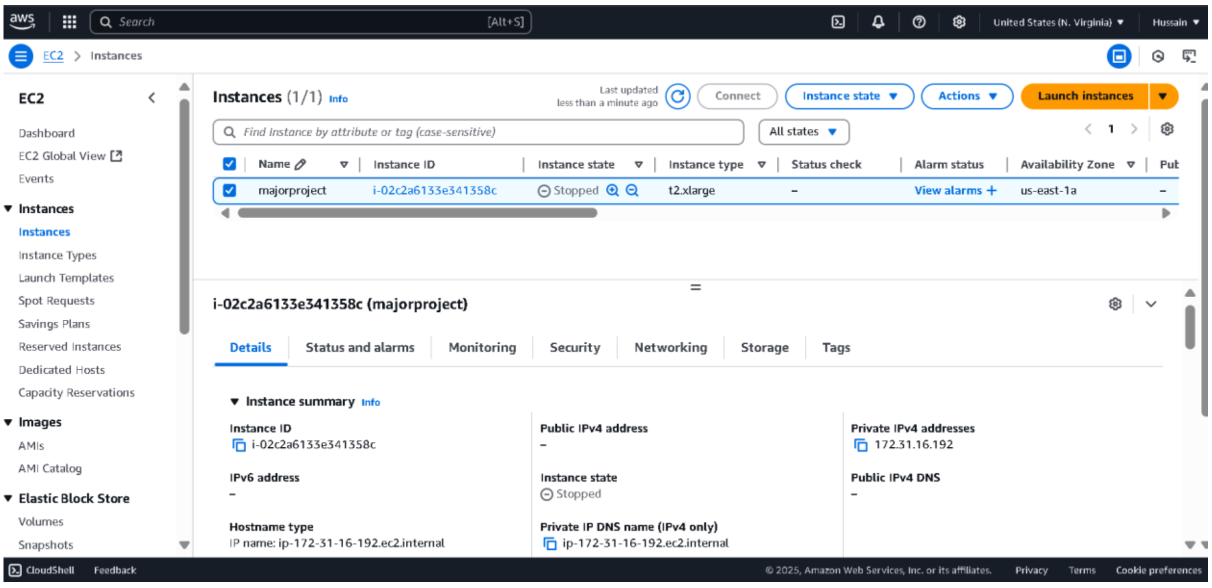
Image scan overview, status, and full vulnerabilities has moved to the image detail page. To access, click an image tag.

Images (17)

Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest	Last recorded pull time
latest	Image	March 25, 2025, 17:15:47 (UTC+05.5)	559.06	Copy URI	sha256:f2d2514e0f005...	March 25, 2025, 17:15:54 (UTC+05.5)
-	Image	March 25, 2025, 17:09:44 (UTC+05.5)	559.06	Copy URI	sha256:8d71870cb94ae...	March 25, 2025, 17:09:51 (UTC+05.5)
-	Image	March 25, 2025, 16:28:56 (UTC+05.5)	559.06	Copy URI	sha256:4d4553b746ef7...	March 25, 2025, 16:34:22 (UTC+05.5)
-	Image	March 25, 2025, 15:50:27 (UTC+05.5)	559.07	Copy URI	sha256:34a2b79103469...	March 25, 2025, 15:50:35 (UTC+05.5)
-	Image	March 25, 2025, 15:25:05 (UTC+05.5)	559.06	Copy URI	sha256:5c4356d80a838...	March 25, 2025, 15:36:36 (UTC+05.5)

Fig 7.9: AWS ECR

STEP 10: AWS EC2 Instance Configuration for Deployment


EC2 > Instances

Instances (1/1) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Pub
majorproject	i-02c2a6133e341358c	Stopped	t2.xlarge	–	View alarms +	us-east-1a	–

i-02c2a6133e341358c (majorproject)

Details | Status and alarms | Monitoring | Security | Networking | Storage | Tags

Instance summary Info

Instance ID: i-02c2a6133e341358c	Public IPv4 address: –	Private IPv4 addresses: 172.31.16.192
IPv6 address: –	Instance state: Stopped	Public IPv4 DNS: –
Hostname type: IP name: ip-172-31-16-192.ec2.internal	Private IP DNS name (IPv4 only): ip-172-31-16-192.ec2.internal	

© 2025, Amazon Web Services, Inc. or its affiliates.

Fig 7.10: AWS EC2 Instance

CHAPTER-8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

This project presented the design and implementation of an intelligent, automated network security system using Machine Learning and MLOps principles. The system successfully detects potential network intrusions with high accuracy, automates data processing pipelines, and enables seamless model deployment through CI/CD mechanisms.

Key outcomes include:

- A structured ETL pipeline for extracting, validating, and preparing network data.
- A high-performing intrusion detection model with accuracy exceeding 94%.
- Full deployment of the model using FastAPI, containerized via Docker, and integrated with MongoDB Atlas for prediction logging.
- Automated workflows using GitHub Actions to support reproducibility and continuous delivery.

By incorporating MLOps, the project demonstrates not just effective machine learning but also production-ready system design, ensuring stability, traceability, and scalability of the intrusion detection pipeline.

8.2 Future Work

Although the project lays a strong foundation for network security using MLOps, there is scope for further enhancement:

1. **Real-time Intrusion Detection**
    - Current system performs near-real-time predictions. Integrating streaming tools like Apache Kafka or Apache Spark Streaming could improve responsiveness for live networks.
2. **Advanced Model Retraining**
    - Introduce scheduled or trigger-based model retraining pipelines using Apache Airflow or Kubeflow Pipelines.

3. **Dashboard and Alert System**
    - Add a real-time dashboard (using Power BI, Grafana, or Streamlit) for visualizing threats.
    - Implement email/SMS alert systems when critical threats are detected.
4. **Extended Attack Types**
    - Retrain models on larger datasets (e.g., CIC-IDS 2017 or UNSW-NB15) for broader intrusion coverage and better generalization.
5. **Enhanced Security Layers**
    - Incorporate authentication and authorization in the API endpoints.
    - Encrypt communication between services and add rate-limiting or firewall rules.
6. **Cloud-native Deployment**
    - Deploy the entire application on platforms like AWS, GCP, or Azure using Kubernetes for orchestration.

8.3 Final Thoughts

This project not only demonstrates the potential of ML in cybersecurity but also bridges the gap between model development and production deployment using MLOps. The system is modular, extensible, and aligned with industry standards—making it a practical solution for enterprise-grade network security.

REFERENCES

- M. A. Ambusaidi, X. He, P. Nanda, and Z. Tan, "Building an intrusion detection system using a filter-based feature selection algorithm," IEEE Transactions on Computers, vol. 65, no. 10, pp. 2986–2998, 2016.

- S. S. Manvi and P. V. Shivalingaiah, "An intrusion detection system based on machine learning and feature selection," International Journal of Information Security Science, vol. 6, no. 3, pp. 65–75, 2017.

- H. Lashkari, G. Draper-Gil, M. S. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features," in Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017), 2017, pp. 253–262.

- M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Security and Defense Applications, 2009, pp. 1–6.

- W. Wang, X. Zhang, S. Gombault, and S. J. Knapskog, "Attribute normalization in network intrusion detection," in Proceedings of the 10th International Symposium on Pervasive Systems, Algorithms, and Networks, 2009, pp. 448–453.

- H. Ringberg, A. Soule, J. Rexford, and C. Diot, "Sensitivity of pca for traffic anomaly detection," ACM SIGMETRICS Performance Evaluation Review, vol. 35, no. 1, pp. 109–120, 2007.

- A. Ghorbani, W. Lu, and M. Tavallaee, Network intrusion detection and prevention: concepts and techniques, Springer Science & Business Media, 2009.