

# OJT - PRODUCT REQUIREMENTS DOCUMENT (PRD)

## Network Security / Intrusion Detection ML System

**Student Name:** Prince kumar maurya

**Roll No:** 240410700103

**Year & Section:** 2026 | Track: Data Scientist

**Project Title:** Network Security / Intrusion Detection ML System

**Stack / Framework:** Language & ML: Python, Pandas, NumPy, Scikit-learn, Imbalanced-learn(SMOTE), MLflow, Evidently AI

Data & ETL: MongoDB Atlas, PyMongo, YAML-based configuration, CSV/NumPy artifacts, Backend API: FastAPI, Unicorn, Pydantic,

Cloud & Deployment: AWS S3 (artifact storage), AWS ECR (Docker registry), AWS EC2 (deployment), IAM

DevOps & MLOps: Docker, GitHub Actions (CI/CD), setup.py packaging, Logging & Custom Exception Handling, Version Control: Git & GitHub

---

## 1. Problem Understanding

### 1.1 What is the problem statement in your own words?

Organizations generate huge amounts of network traffic data, and manually detecting malicious activity is slow, inefficient, and not scalable. The problem is the lack of an automated, production-ready system that can accurately detect network attacks and continuously deploy improved models in a reliable cloud environment.

### 1.2 Why does this problem exist or matter?

This problem exists because cyber threats are constantly evolving, while traditional security systems rely heavily on manual monitoring or static rules. As network traffic grows rapidly, it becomes impossible to detect attacks efficiently without automation. It matters because delayed or missed detection can lead to data breaches, financial loss, and serious security risks for organizations.

## 1.3 Target Users

The target users are Security Teams, SOC (Security Operations Center) Analysts, DevSecOps Engineers, and IT administrators who need automated systems to monitor network traffic and detect potential cyber threats efficiently.

## 1.4 Data Description

The system uses structured network traffic data containing features such as protocol type, connection duration, source bytes, destination bytes, flags, and other traffic-related attributes. Each record represents a network connection and is labeled as either Normal or Attack, which is used to train and evaluate the intrusion detection model.

---

## 1.5 Key inputs and expected outputs

Inputs	Process	Expected Outputs
1. Network traffic dataset (CSV files or MongoDB Atlas records)	1. Data ingestion and storage in feature store	1. Trained and validated intrusion detection model
2. Traffic features (protocol type, duration, source bytes, destination bytes, flags)	2. Data validation and drift detection	2. Predictions labeled as Normal or Attack
3. Target labels indicating Normal or Attack	3. Data transformation (cleaning, scaling, encoding, SMOTE)	3. Model performance metrics (Accuracy, Precision, Recall, F1-score)

<b>4.</b> YAML configuration files for schema and model settings	<b>4.</b> Model training, hyperparameter tuning, and evaluation	<b>4.</b> Versioned model artifacts stored in AWS S3
<b>5.</b> Existing deployed model (for performance comparison)	<b>5.</b> Automated model comparison and cloud deployment via Docker & CI/CD	<b>5.</b> Deployment-ready Docker image with live API endpoint
<b>6.</b> Cloud credentials and environment variables for secure deployment	<b>6.</b> Automated packaging and deployment using Docker, CI/CD, and AWS	<b>6.</b> Deployment-ready Docker image with live inference API endpoint

---

## 2. Functional Scope

### 2.1 Core Features (Must-Haves)

1. Automated data ingestion from CSV/MongoDB
2. Data validation and drift detection
3. Data cleaning and transformation (scaling, encoding, SMOTE)
4. Model training, tuning, and evaluation
5. Automatic model comparison and cloud deployment
6. FastAPI endpoint for real-time predictions

## 2.2 Stretch Goals

1. Real-time data streaming using Kafka for live intrusion detection
2. Model explainability using SHAP for better decision transparency
3. Monitoring dashboard to track model performance and data drift
4. Auto-scaling deployment using AWS ECS or App Runner
5. Automated alert system for high-risk or suspicious activities
6. Model versioning and rollback support for safer production updates

## 2.3 Libraries & Tools

1. **Python Libraries:** Pandas, NumPy, Scikit-learn, Imbalanced-learn, MLflow, Evidently
  2. **Database Tools:** MongoDB Atlas, PyMongo
  3. **API Framework:** FastAPI, Unicorn, Pydantic
  4. **Cloud Services:** AWS S3, ECR, EC2, IAM
  5. **DevOps Tools:** Docker, GitHub Actions, Git & GitHub
- 

## 3. System & Design Thinking

### 3.1 App Flow / Pipeline

Data Source (CSV / MongoDB) → Data Ingestion → Data Validation & Drift Detection → Data Preprocessing & SMOTE → Model Training & Hyperparameter Tuning → Model Evaluation & Comparison → Model Registry (S3) → Docker Packaging → CI/CD Deployment (ECR → EC2) → FastAPI Prediction Endpoint → Attack / Normal Classification Output.

### 3.2 Data Structures & Algorithms

#### Data Structures Used:

1. DataFrames (Pandas) – For handling and transforming structured network traffic data
2. NumPy Arrays – For efficient numerical computations during model training
3. Dictionaries – For configuration management and parameter storage
4. JSON/YAML Files – For schema validation and pipeline configuration
5. Serialized Objects (Pickle) – For saving trained models and preprocessing pipelines

## **Algorithms Used:**

1. Classification Algorithms – Logistic Regression, Random Forest, etc.
2. KNN Imputer – For handling missing values
3. SMOTE – For handling class imbalance
4. RobustScaler – For feature scaling
5. Hyperparameter Tuning (GridSearchCV / RandomSearchCV) – For model optimization

## **3.3 Testing Correctness & Performance**

### **Correctness Testing:**

1. Unit tests for each pipeline component (ingestion, validation, transformation, training)
2. Schema validation to ensure correct data structure and types
3. Model prediction testing using unseen test data
4. API endpoint testing for valid request/response handling
5. CI/CD checks to ensure safe and error-free deployment

### **Performance Testing:**

1. Model evaluation using Accuracy, Precision, Recall, and F1-score
  2. Cross-validation to ensure consistent performance
  3. Inference time measurement for fast predictions
  4. Basic load testing to handle multiple API requests
  5. Continuous monitoring for model drift and degradation
- 

## **4. ML / Analytics Approach**

### **4.1 Data Cleaning & Preparation:**

1. Handle missing values using KNN Imputer or SimpleImputer
2. Remove duplicate or irrelevant columns
3. Encode categorical features into numerical format

4. Scale numerical features using RobustScaler
5. Handle class imbalance using SMOTE
6. Split dataset into training and testing sets

#### **4.2 Feature Engineering:**

1. Generate derived features from raw network attributes (e.g., byte ratios, connection rates).
2. Encode categorical variables such as protocol type and flags.
3. Create statistical features (mean, count, frequency-based metrics).
4. Remove low-importance or highly correlated features.
5. Normalize and scale features for model stability.
6. Select the most relevant features using feature importance techniques.

#### **4.3 Model Development:**

1. Train multiple classification models.
2. Perform hyperparameter tuning.
3. Use cross-validation for stability.
4. Evaluate using Accuracy, Precision, Recall, and F1-score.
5. Select and save the best-performing model for deployment.

#### **4.4 Explainability & Trust:**

1. Use SHAP to explain model predictions.
2. Identify key features influencing attack detection.
3. Provide feature importance reports.
4. Ensure transparency in model decisions.
5. Enable monitoring to maintain model reliability over time.

#### **4.5 Explainability & Trust:**

1. Use SHAP to interpret individual predictions.
2. Display feature importance for transparency.
3. Provide explanation reports for security teams.

4. Monitor model behavior to detect bias or drift.
5. Ensure consistent and reliable decision-making in production.

#### **4.6 Deployment-Ready Inference Pipeline:**

1. Load trained model and preprocessing artifacts from AWS S3.
2. Expose a FastAPI endpoint for real-time predictions.
3. Accept CSV or JSON input for batch and single inference.
4. Containerize the service using Docker for portability.
5. Deploy on AWS EC2 via CI/CD for scalable production access.

## **5. Timeline & Milestones**

<b>Month</b>	<b>Week</b>	<b>Phase</b>	<b>Key Deliverables &amp; Tasks</b>
Month 1	W1	Project Launch & Scope	Define problem statement, target users, and technical direction. Finalize tech stack (Python, FastAPI, MongoDB, AWS, Docker). Prepare PRD document and confirm project scope.
	W2	Product Design & User Flow	Define system workflow (data ingestion → training → deployment → prediction). Design user interaction flow for CSV upload

			and prediction. Create high-level architecture diagram.
	W3	Technical Design (HLD + LLD) & Feasibility	Prepare High-Level Design (HLD) and Low-Level Design (LLD). Validate feasibility of ETL pipeline and MLOps architecture. Perform initial EDA and baseline model experiment in notebook. Setup experiment tracking and record initial results.
	W4	Development Sprint 1 – Core Foundations	Initialize Git repository and project structure. Setup backend scaffold (FastAPI). Configure environment variables and logging. Connect MongoDB Atlas. Create health-check API endpoint.
Month 2	W5	Development Sprint 2 – Core Feature Implementation	Implement data ingestion component. Build data validation module (schema + drift). Develop data transformation pipeline. Train baseline model and save artifacts.
	W6	Development Sprint 3 – Working Pipeline	Implement model evaluation and comparison logic. Build prediction API endpoint. Validate full pipeline flow (ingestion → training → prediction). Conduct functional testing and review.

	W7	Deployment & Public Access	Dockerize application. Push Docker image to AWS ECR. Deploy container on EC2. Make API publicly accessible. Perform deployment validation.
	W8	Scalability & Optimization	Optimize model performance and inference speed. Fine-tune hyperparameters. Improve pipeline efficiency. Update experiment tracking results. Enhance error handling and logging.
Month 3	W9	Evaluation & Industry-Level Review	Conduct performance evaluation (accuracy, latency, drift checks). Implement stretch features (advanced monitoring or batch prediction improvements). Prepare system design explanation, scaling discussion, and live demo readiness.

## Month 1 – Planning & Architecture

- W1: Problem definition + PRD + Tech stack
- W2: System workflow + User flow + Architecture diagram
- W3: HLD + LLD + EDA + Baseline model
- W4: Backend scaffold (FastAPI) + MongoDB + project setup

## Month 2 – Build & Deploy

- W5: Data ingestion + Validation + Training pipeline
- W6: Model evaluation + Prediction API + End-to-end testing
- W7: Dockerize + Deploy on AWS (ECR + EC2)
- W8: Optimization + Logging + Performance improvements

Month 3 – Production Readiness & Interview Prep

W9:

System performance evaluation (accuracy, latency, drift)

Add stretch features (monitoring / batch prediction)

Prepare live demo

Prepare system design explanation

Prepare scaling & trade-off discussion

---

## 6. Risks & Dependencies

### 6.1 Hardest part technically:

1. Building a fully automated end-to-end MLOps pipeline.
2. Implementing reliable data validation and drift detection.
3. Managing model comparison and safe deployment logic.
4. Integrating Docker, CI/CD, and AWS services smoothly.
5. Ensuring scalability, performance, and system stability in production.

### 6.2 Dependencies / mentor support needed:

1. Guidance on designing a scalable MLOps architecture.
2. Support in optimizing model selection and evaluation strategy.
3. Assistance with AWS deployment best practices (S3, ECR, EC2, IAM).
4. Review of CI/CD pipeline setup and Docker configuration.
5. Feedback on improving model explainability and production readiness.

## 6.3 Explainability & Trust

1. Integrate SHAP to explain why a traffic instance is classified as Normal or Attack.
  2. Highlight top contributing features for each prediction.
  3. Provide global feature importance to understand overall model behavior.
  4. Monitor model drift to ensure consistent performance over time.
  5. Maintain transparent logs for auditing and security review.
- 

## 7. Evaluation Readiness

### 7.1 How will you prove your project “works”?

1. Achieve strong evaluation metrics (Accuracy, Precision, Recall, F1-score) on test data.
2. Demonstrate correct classification of network traffic as **Normal** or **Attack**.
3. Validate model stability using cross-validation.
4. Show successful end-to-end pipeline execution (ingestion → training → deployment).
5. Provide live API demo with real-time or batch predictions.

### 7.2 Success metrics or goals:

1. Model Accuracy  $\geq 90\%$  on test dataset.
  2. High Recall for attack detection to minimize missed threats.
  3. Low False Positive Rate ( $< 5\%$ ) to reduce unnecessary alerts.
  4. Stable cross-validation scores across multiple folds.
  5. Successful automated deployment with zero manual intervention.
-

## 8. Responsibilities

Task	Prince kumar maurya	Mentor Notes
Dataset analysis & validation	<input checked="" type="checkbox"/>	
Data ingestion pipeline development	<input checked="" type="checkbox"/>	
Data transformation & preprocessing	<input checked="" type="checkbox"/>	
Handling class imbalance (SMOTE)	<input checked="" type="checkbox"/>	
Model development & hyperparameter tuning	<input checked="" type="checkbox"/>	
Model evaluation & comparison logic	<input checked="" type="checkbox"/>	
API development for inference	<input checked="" type="checkbox"/>	
Dockerization & CI/CD setup	<input checked="" type="checkbox"/>	

AWS deployment (S3, ECR, EC2)	<input checked="" type="checkbox"/>	
Explainability, monitoring & documentation	<input checked="" type="checkbox"/>	

---

## 9. Deliverables

### 9.1 Data & EDA Deliverables

1. Cleaned and validated network traffic dataset.
2. Exploratory Data Analysis (EDA) report with key insights.
3. Data distribution and class imbalance analysis.
4. Correlation analysis and feature importance overview.
5. Data drift and validation reports (if applicable).

### 9.2 Machine Learning Deliverables

1. Trained and optimized intrusion detection model.
2. Saved model artifact (model.pkl) and preprocessing pipeline (preprocessing.pkl).
3. Model evaluation report (Accuracy, Precision, Recall, F1-score).
4. Hyperparameter tuning results and model comparison summary.
5. Versioned model artifacts stored in AWS S3.

### 9.3 Backend Deliverables

1. FastAPI-based inference API for real-time predictions.
2. Batch prediction endpoint supporting CSV input.
3. Integrated model loading from AWS S3.
4. Dockerized backend application ready for deployment.
5. CI/CD pipeline configured for automated build and deployment.

## **9.4 Frontend Deliverables**

1. User interface for uploading network traffic CSV files.
2. Dashboard displaying prediction results (Normal / Attack).
3. Visualization of key metrics and model insights.
4. Feature importance or SHAP explanation display.
5. Option to download prediction results as a CSV report.

## **9.5 Deployment Deliverables**

1. Dockerized application image ready for production use.
  2. CI/CD pipeline configured via GitHub Actions.
  3. Docker image pushed to AWS ECR repository.
  4. Deployed application running on AWS EC2 instance.
  5. Model artifacts securely stored and versioned in AWS S3.
- 

**Signatures (Student) : Prince kumar maurya**

**Mentor Approval :**

**Date :**