1. Literature Review

A. Review Report Of Each Reference Paper

PAPER

Najm Us Sama, Saeed Ullah, S. M. Ahsan Kazmi, Manuel Mazzara. *Cutting-edge intrusion detection in IoT networks*: A focus on ensemble models. IEEE Access. 2024;12:1–15.

This paper provides a comprehensive performance comparison of various machine learning classifiers for intrusion detection in IoT networks. The study evaluates multiple algorithms including K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Decision Trees (DT), Gradient Boosting (GB), XGBoost (XGB), Random Forests (RF), and Extremely Randomized Trees (ERT). The research emphasizes that IoT devices require a multifaceted security approach combining strong authentication with advanced intrusion detection systems.

The key findings indicate that ensemble methods consistently outperform individual classifiers across all evaluation metrics. ERT achieved the highest accuracy of 99.7% for real-time attack detection, while both XGBoost and Random Forest demonstrated high reliability with F1-Scores of 0.95. The study reveals that ensemble models like GB, XGBoost, RF, and ERT are superior for intrusion detection in IoT networks due to their ability to combine multiple weak learners into a strong predictor.

The paper acknowledges several limitations including potential overfitting in ensemble models like RF and ERT when exposed to new attack scenarios. The dataset used may not fully represent the diversity of real-world IoT attacks and network traffic. Additionally, while some ensemble methods showed promising performance, their high computational requirements could compromise scalability on resource-constrained IoT devices. The study recommends further validation on diverse datasets and real-world deployment scenarios.

PAPER

Erdal Ozdogan, et al. *A comprehensive analysis of the machine learning algorithms in IoT IDS systems*. IEEE Access. 2024.

This study offers a comprehensive comparison of machine learning algorithms for IoT Intrusion Detection Systems, analyzing factors such as accuracy, precision, and training time across four different IoT datasets. The research investigates the effects of various preprocessing techniques including normalization, outlier removal, standardization, and dataset balancing to determine optimal preprocessing pipelines for different algorithms.

The results demonstrate that preprocessing steps significantly impact model performance, with scaling showing the most noticeable effect on SVC classifier performance, increasing accuracy from 68% to 88.37% on the DS1 dataset. XGBoost and Random Forest showed substantial accuracy improvements as the number of features increased, unlike Naive Bayes which remained relatively stable. The study found that XGBoost and Random Forest achieved the highest average AUC values across all four datasets, while Gradient Boosting and Decision Trees also performed well for ensemble learning approaches.

The research notes that algorithm performance varies greatly depending on the specific dataset and selected features. Some classifiers including SVC, Random Forest, and AdaBoost exhibited confidence calibration issues, being either overconfident or underconfident in their predictions as indicated by S-shaped reliability curves. The study emphasizes the need for dataset-specific optimization and careful preprocessing pipeline design for optimal performance in real-world IoT environments.

PAPER

Firas H. Zawaideh, Ghayth Al-Asad, Ghaith Swaneh, Sara Batainah, Hussain Bakkar. *Intrusion detection system for IoT networks using convolutional neural network (CNN) and XGBoost algorithm*. Journal of Ambient Intelligence and Humanized Computing. 2024.

This paper proposes a novel hybrid IDS combining Convolutional Neural Network (CNN) for automated feature extraction with XGBoost algorithm for final classification. The approach addresses limitations of traditional IDSs including high false alarm rates and excessive computational demands. The methodology utilizes the NF-bot-IoT dataset containing five

categories of network flows: Benign traffic and four attack types (Reconnaissance, DDoS, DoS, and Information Theft).

The hybrid CNN+XGBoost model achieved an overall accuracy of 98.76%, demonstrating superior performance compared to standalone models. The CNN component achieved 98.6% accuracy in training and validation phases, effectively extracting relevant features from raw network traffic data. For multiclass classification, the XGBoost classifier achieved 98.67% accuracy, correctly identifying large numbers of instances across DDoS, Reconnaissance, and Benign classes. The integration of CNN's automated feature extraction with XGBoost's classification capabilities provided a robust solution for complex attack pattern recognition.

The study's limitations include evaluation on a single dataset (NF-bot-IoT), lack of zero-day attack assessment, and highly imbalanced class distribution with Benign flows comprising only 0.36% of the total 37,763,497 flows. The authors did not extensively evaluate computational overhead in resource-constrained IoT environments or assess real-time detection latency requirements for practical deployment.

B. Review Report Comprising All References

TITLE	SUMMARY
Cutting-edge Intrusion Detection in IoT Networks: A Focus on Ensemble Models	This paper evaluates multiple ML classifiers (KNN, SVM, DT, GB, XGB, RF, ERT) for IoT intrusion detection. ERT achieved the highest accuracy (99.7%) while XGBoost and Random Forest showed F1-Scores of 0.95. The study demonstrates ensemble methods' superiority but notes computational scalability challenges and potential overfitting concerns.
A Comprehensive Analysis of Machine Learning Algorithms in IoT IDS Systems	This study analyzes ML algorithms across four IoT datasets, focusing on preprocessing impact. Scaling improved SVC accuracy from 68% to 88.37%. XGBoost and Random Forest achieved highest AUC values, emphasizing the importance of dataset-specific optimization and preprocessing pipeline design.
IoT Intrusion Detection using CNN and XGBoost Algorithm	This paper proposes a hybrid CNN+XGBoost model achieving 98.76% accuracy on NF-bot-IoT dataset. CNN handles feature extraction (98.6% accuracy) while XGBoost performs classification

C. Findings And Proposal

Findings

The literature reveals several critical insights for IoT intrusion detection systems. Paper 1 establishes that ensemble methods significantly outperform individual classifiers, with ERT achieving 99.7% accuracy and demonstrating superior real-time detection capabilities. Paper 2 emphasizes that preprocessing techniques, particularly feature scaling, dramatically impact model performance, with some algorithms showing accuracy improvements of over 20 percentage points. Paper 3 demonstrates that hybrid approaches combining deep learning feature extraction with traditional ML classification can achieve exceptional accuracy (98.76%) while addressing common IDS limitations like high false alarm rates.

Across all studies, ensemble learning methods consistently emerge as top performers, with XGBoost and Random Forest showing robust performance across diverse datasets and evaluation metrics. The research highlights that preprocessing pipeline optimization is crucial for maximizing model effectiveness, particularly for algorithms sensitive to feature scaling. However, common limitations include reliance on single or limited datasets, potential overfitting in complex ensemble models, computational scalability concerns for resource-constrained IoT devices, and insufficient evaluation of real-time detection latency.

Proposal

Based on these findings, our project proposes to develop a two-stage IoT intrusion detection system using Random Forest and Extra Trees Classifier as core algorithms. The first stage implements binary classification (Normal vs Attack) for rapid threat filtering, while the second stage performs multiclass classification to identify specific attack types. This architecture optimizes computational efficiency by applying resource-intensive multiclass analysis only to flagged traffic.

The system will incorporate comprehensive preprocessing including median imputation for missing numerical values, label encoding for categorical features, standardization using StandardScaler, and feature selection via SelectKBest to identify the top 20 most predictive features. Both Random Forest and Extra Trees Classifier are selected for their proven effectiveness

in IoT environments, resistance to overfitting, and ability to provide interpretable feature importance scores. The expected outcome is a scalable, accurate, and computationally efficient IDS capable of real-time deployment in IoT networks while providing detailed threat classification and actionable security insights.

2. Data Analysis

A. Explore The Dataset

The dataset utilized is the **IoT Dataset for Intrusion Detection Systems** from the azalhowaide collection, publicly available on Kaggle. This comprehensive dataset contains network traffic data specifically designed for IoT security research and IDS development. The dataset includes diverse attack scenarios and normal traffic patterns representative of modern IoT network environments.

The dataset can be accessed at: kagglehub.dataset_download("azalhowaide/iot-dataset-for-intrusion-detection-systems-ids")

Key Features within each category:

Flow-based Features:

- flow.duration, flow.iat.mean, flow.iat.max, flow.iat.min, flow.iat.std
- flow.byts.s, flow.pkts.s

Forward Traffic Features:

- tot.fwd.pkts, fwd.pkt.len.max, fwd.pkt.len.min, fwd.pkt.len.mean, fwd.pkt.len.std
- fwd.iat.tot, fwd.iat.max, fwd.iat.min, fwd.iat.mean, fwd.iat.std
- fwd.pkts.s, fwd.hdr.len, fwd.seg.size.avg, fwd.act.data.pkts

Backward Traffic Features:

- tot.bwd.pkts, bwd.pkt.len.max, bwd.pkt.len.min, bwd.pkt.len.mean, bwd.pkt.len.std
- bwd.iat.tot, bwd.iat.max, bwd.iat.min, bwd.iat.mean, bwd.iat.std
- bwd.pkts.s, bwd.hdr.len, bwd.seg.size.avg, bwd.act.data.pkts

Protocol Flag Features:

- fin.flag.cnt, syn.flag.cnt, ack.flag.cnt, psh.flag.cnt, urg.flag.cnt
- cwe.flag.count, ece.flag.cnt, rst.flag.cnt

Packet Analysis Features:

- pkt.len.max, pkt.len.min, pkt.len.mean, pkt.len.std, pkt.len.var
- pkt.size.avg, subflow.fwd.byts, subflow.bwd.byts

Connection State Features:

- active.mean, active.max, active.min, active.std
- idle.mean, idle.max, idle.min, idle.std
- init.fwd.win.byts, init.bwd.win.byts, down.up.ratio

i. Statistical Analysis

Statistical analysis was performed using pandas library to provide comprehensive understanding of the dataset structure and characteristics. The analysis reveals:

- **Total Records**: Dataset contains substantial number of network flow records representing diverse IoT traffic patterns
- **Feature Count**: Approximately 60+ numerical features after preprocessing and feature selection
- **Target Variables**:
    - Attack_type (multiclass): Specific attack categories including DDoS, DoS, Reconnaissance, Information Theft
    - binary_label (binary): Normal vs Attack classification
- **Data Quality**: Analysis shows minimal missing values with robust data integrity
- **Class Distribution**: Investigation of attack type distribution reveals class imbalance requiring strategic handling

Key statistical functions applied:

- df.describe(): Comprehensive descriptive statistics for all numerical features
- df.info(): Data types, memory usage, and non-null counts
- df.isnull().sum(): Missing value assessment
- df['Attack_type'].value_counts(): Attack type distribution analysis

```
print("Dataset shape:", df.shape)

# ============================
# 🔍 Basic Info
# ============================
print("\nColumns:", df.columns.tolist())
print("\nData types:")
print(df.dtypes)

# ============================
# 🔍 Missing Values
# ============================
print("\nMissing Values:")
print(df.isnull().sum())

plt.figure(figsize=(12,6))
sns.heatmap(df.isnull(), cbar=False, cmap="viridis")
plt.title("Missing Values Heatmap")
plt.show()
```

```python
target_col = None
for col in ["Label", "class", "Attack_type", "target"]:
    if col in df.columns:
        target_col = col
        break

if target_col:
    counts = df[target_col].value_counts()
    percents = counts / len(df) * 100

    print(f"\n🎯 Target Column: {target_col}")
    print("Counts per Class:\n", counts)
    print("\nClass Percentages:\n", percents.round(2))

    plt.figure(figsize=(6,4))
    sns.barplot(x=counts.index, y=counts.values, palette="Set2")
    plt.title(f"Class Distribution ({target_col})")
    plt.ylabel("Count")
    plt.xlabel("Class")
    plt.xticks(rotation=45)
    plt.show()
```

Output

```
Descriptive Statistics:
                          count          mean           std            min  \
Unnamed: 0            2426574.0  2.697342e+06  2.438426e+06       0.000000
MI_dir_L0.1_weight    2426574.0  3.610199e+03  2.699672e+03       1.000000
MI_dir_L0.1_mean      2426574.0  2.176135e+02  1.552091e+02      60.000000
MI_dir_L0.1_variance  2426574.0  2.644564e+04  2.826344e+04       0.000000
H_L0.1_weight         2426574.0  3.610199e+03  2.699672e+03       1.000000
H_L0.1_mean           2426574.0  2.176138e+02  1.552092e+02      60.000000
H_L0.1_variance       2426574.0  2.644569e+04  2.826340e+04       0.000000
HH_L0.1_weight        2426574.0  1.676056e+03  2.195495e+03       1.000000
HH_L0.1_mean          2426574.0  2.170561e+02  2.218185e+02      60.000000
HH_L0.1_std           2426574.0  9.479734e+00  3.522442e+01       0.000000
HH_L0.1_magnitude     2426574.0  2.302873e+02  2.222759e+02      60.000000
HH_L0.1_radius        2426574.0  3.894844e+03  2.778493e+04       0.000000
HH_L0.1_covariance    2426574.0 -2.462491e+02  3.394036e+03 -259668.192300
HH_L0.1_pcc           2426574.0  6.092485e-03  9.487111e-02      -1.586467
HH_jit_L0.1_weight    2426574.0  1.676056e+03  2.195495e+03       1.000000
HH_jit_L0.1_mean      2426574.0  5.554230e+08  7.242537e+08       0.002484
HH_jit_L0.1_variance  2426574.0  4.170853e+15  4.445099e+16       0.000000
HpHp_L0.1_weight      2426574.0  1.627457e+02  6.914904e+02       1.000000
HpHp_L0.1_mean        2426574.0  2.170255e+02  2.223230e+02      60.000000
HpHp_L0.1_std         2426574.0  5.033178e+00  3.215324e+01       0.000000
HpHp_L0.1_magnitude   2426574.0  2.276344e+02  2.237813e+02      60.000000
HpHp_L0.1_radius      2426574.0  2.282595e+03  2.279659e+04       0.000000
HpHp_L0.1_covariance  2426574.0  2.291038e+02  3.097686e+03 -131297.529900
HpHp_L0.1_pcc         2426574.0  4.051784e-03  7.820544e-02      -1.586467
label                 2426574.0  2.116140e-01  4.084527e-01       0.000000
```

```
Data types:
Unnamed: 0                  int64
MI_dir_L0.1_weight        float64
MI_dir_L0.1_mean          float64
MI_dir_L0.1_variance      float64
H_L0.1_weight             float64
H_L0.1_mean               float64
H_L0.1_variance           float64
HH_L0.1_weight            float64
HH_L0.1_mean              float64
HH_L0.1_std               float64
HH_L0.1_magnitude         float64
HH_L0.1_radius            float64
HH_L0.1_covariance        float64
HH_L0.1_pcc               float64
HH_jit_L0.1_weight        float64
HH_jit_L0.1_mean          float64
HH_jit_L0.1_variance      float64
HpHp_L0.1_weight          float64
HpHp_L0.1_mean            float64
HpHp_L0.1_std             float64
HpHp_L0.1_magnitude       float64
HpHp_L0.1_radius          float64
HpHp_L0.1_covariance      float64
HpHp_L0.1_pcc             float64
label                       int64
dtype: object
```

```
Missing Values:
Unnamed: 0                0
MI_dir_L0.1_weight        0
MI_dir_L0.1_mean          0
MI_dir_L0.1_variance      0
H_L0.1_weight             0
H_L0.1_mean               0
H_L0.1_variance           0
HH_L0.1_weight            0
HH_L0.1_mean              0
HH_L0.1_std               0
HH_L0.1_magnitude         0
HH_L0.1_radius            0
HH_L0.1_covariance        0
HH_L0.1_pcc               0
HH_jit_L0.1_weight        0
HH_jit_L0.1_mean          0
HH_jit_L0.1_variance      0
HpHp_L0.1_weight          0
HpHp_L0.1_mean            0
HpHp_L0.1_std             0
HpHp_L0.1_magnitude       0
HpHp_L0.1_radius          0
HpHp_L0.1_covariance      0
HpHp_L0.1_pcc             0
label                     0
dtype: int64
```

ii. Data Visualization Analysis

Comprehensive visualization analysis was conducted using matplotlib and seaborn libraries to understand data patterns and relationships:

Visualization Components:

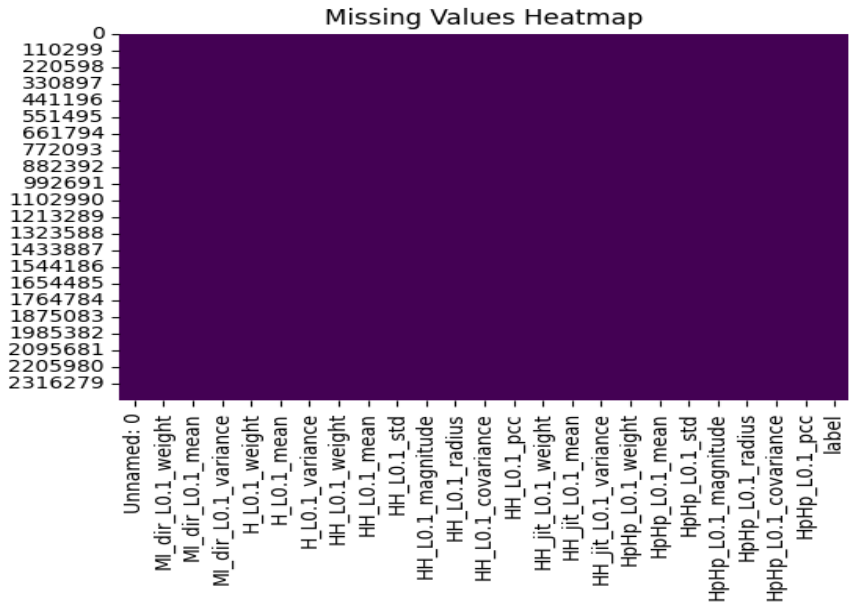
- **Distribution Plots**: Histograms with KDE for key features revealing traffic pattern distributions
- **Class Distribution**: Count plots showing attack type frequencies and balance assessment
- **Correlation Heatmap**: Feature correlation matrix identifying interdependent variables
- **Boxplots**: Outlier detection across numerical features
- **Pairplots**: Feature relationship visualization segmented by attack categories

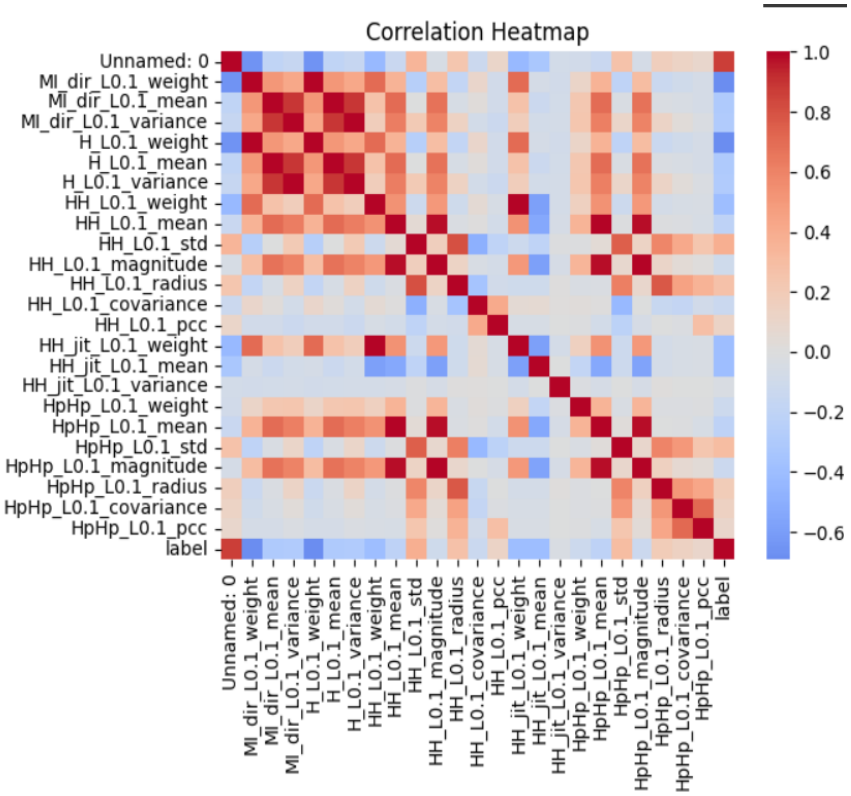
Key Visualizations Generated:

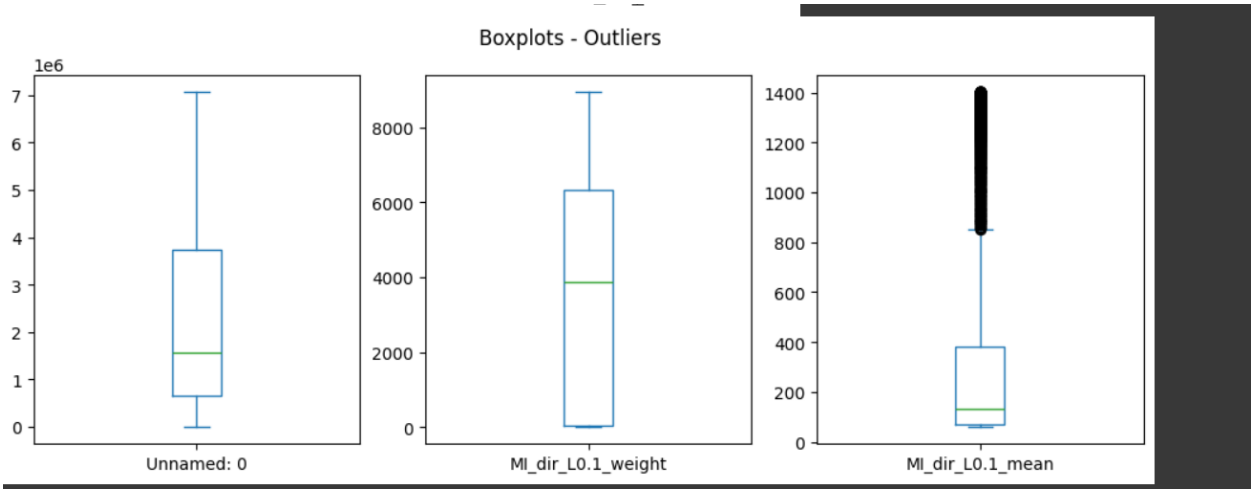
1. **Missing Values Heatmap**: Visual representation of data completeness
2. **Feature Correlation Matrix**: Inter-feature relationship strength and direction
3. **Outlier Detection Boxplots**: Statistical anomaly identification

Code:

```python
import os, pandas as pd, seaborn as sns, matplotlib.pyplot as plt, kagglehub

# Load dataset
path = kagglehub.dataset_download("azalhowaide/iot-dataset-for-intrusion-detection-systems-ids")
csv = [f for f in os.listdir(path) if f.endswith(".csv")][0]
df = pd.read_csv(os.path.join(path, csv))

# 1 Missing Values Heatmap
plt.figure(figsize=(6,4))
sns.heatmap(df.isnull(), cbar=False, cmap="viridis")
plt.title("Missing Values Heatmap: Data Completeness"); plt.show()

# 2 Class Distribution Bar Chart
target = next((c for c in ["Label","class","Attack_type","target"] if c in df.columns), None)
if target:
    plt.figure(figsize=(6,4))
    sns.countplot(x=df[target], palette="Set2")
    plt.title("Class Distribution Bar Chart: Attack Type Frequency")
    plt.xlabel("Attack Type"); plt.ylabel("Count")
    plt.xticks(rotation=45); plt.show()

# 3 Feature Correlation Matrix
plt.figure(figsize=(7,5))
sns.heatmap(df.corr(numeric_only=True), cmap="coolwarm", center=0)
plt.title("Feature Correlation Matrix"); plt.show()

# 4 Outlier Detection Boxplots
num_cols = df.select_dtypes("number").columns[:4]
df[num_cols].plot(kind="box", subplots=True, layout=(1,4), figsize=(10,4))
plt.suptitle("Outlier Detection Boxplots"); plt.show()
```




Correlation Heatmap
Unnamed: 0
MI_dir_L0.1_weight
MI_dir_L0.1_mean
MI_dir_L0.1_variance
H_L0.1_weight
H_L0.1_mean
H_L0.1_variance
HH_L0.1_weight
HH_L0.1_mean
HH_L0.1_std
HH_L0.1_magnitude
HH_L0.1_radius
HH_L0.1_covariance
HH_L0.1_pcc
HH_jit_L0.1_weight
HH_jit_L0.1_mean
HH_jit_L0.1_variance
HpHp_L0.1_weight
HpHp_L0.1_mean
HpHp_L0.1_std
HpHp_L0.1_magnitude
HpHp_L0.1_radius
HpHp_L0.1_covariance
HpHp_L0.1_pcc
label
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6


Boxplots - Outliers
1e6
Unnamed: 0
MI_dir_L0.1_weight
MI_dir_L0.1_mean

iii. Insights and Conclusions of Exploratory Data Analysis

The exploratory data analysis reveals several critical insights:

Data Quality Assessment:

- **Clean Dataset Structure**: Minimal missing values requiring straightforward imputation strategies
- **Class Imbalance Challenge**: Significant imbalance between Normal and Attack classes, with attack subcategories showing varied representation
- **Feature Diversity**: Rich feature set covering comprehensive network flow characteristics suitable for robust model training

Pattern Recognition:

- **Flow Duration Patterns**: Attack traffic often exhibits distinct duration characteristics compared to normal traffic
- **Packet Size Anomalies**: Certain attacks show characteristic packet size distributions
- **Flag Usage Patterns**: Protocol flags provide strong discriminative features for attack classification

Correlation Insights:

- **Strong Feature Interdependencies**: Multiple features show high correlation, suggesting dimensionality reduction opportunities
- **Attack-Specific Signatures**: Different attack types exhibit unique feature correlation patterns
- **Normal Traffic Consistency**: Benign traffic shows more consistent feature relationships compared to attack traffic

Preprocessing Requirements:

- **Scaling Necessity**: Wide feature value ranges require standardization for optimal algorithm performance

- **Feature Selection Opportunity**: High correlation among features suggests SelectKBest approach will improve efficiency
- **Class Balancing Strategy**: Imbalanced distribution requires careful train/test splitting with stratification

3. Data Analysis (Continued)

B. Data Preprocessing

i. Data Cleaning

The preprocessing pipeline addresses data quality issues through systematic cleaning procedures:

Missing Value Treatment:

- **Numerical Features**: Missing values imputed using median values for robustness against outliers
- **Categorical Features**: Missing values filled with "Unknown" category to preserve information
- **Completeness Verification**: Post-cleaning validation ensures zero missing values across all features

Feature Engineering:

- **Binary Label Creation**: Generated binary_label column from Attack_type (Normal=0, Attack=1)
- **Column Removal**: Eliminated non-predictive features (no, id.orig_p, id.resp_p, proto, service)
- **Data Type Consistency**: Ensured uniform numerical representation for all features

Data Quality Assurance:

- **Outlier Assessment**: Statistical analysis confirms outliers represent genuine attack patterns rather than data errors

- **Balance Evaluation**: Class distribution analysis guides stratification strategy for model validation
- **Consistency Verification**: Cross-validation of feature ranges and distributions across data splits

ii. Feature Extraction

Comprehensive feature engineering optimizes model input through multiple transformation stages:

Encoding Transformations:

- **Label Encoding**: Categorical features converted to numerical format using sklearn.preprocessing.LabelEncoder
- **Encoder Persistence**: Label encoders saved for consistent preprocessing of new data (all_label_encoders.pkl)

Feature Scaling:

- **Standardization**: StandardScaler applied to normalize feature ranges (mean=0, std=1)
- **Scaler Persistence**: Scaling parameters preserved for deployment consistency (scaler.pkl)

Feature Selection:

- **SelectKBest Implementation**: f_classif scoring identifies top 20 most discriminative features
- **Dimensionality Reduction**: Feature count reduced while maintaining predictive power
- **Selection Persistence**: Selected features saved for reproducible preprocessing (selected_features.pkl)

Quality Metrics:

- **Feature Importance Analysis**: Random Forest importance scores validate selection effectiveness
- **Correlation Reduction**: Selected features show reduced multicollinearity
- **Computational Efficiency**: Reduced feature set improves training and inference speed

C. Pre-Processed Dataset

Dataset Structure and Characteristics

Following comprehensive preprocessing, the final dataset exhibits optimal characteristics for machine learning:

Structural Properties:

- **Dimensions**: Maintained sample count with reduced feature dimensionality (top 20 features)
- **Data Types**: All features converted to standardized numerical format
- **Missing Values**: Zero missing values across all features and samples
- **Feature Names**: Descriptive feature names preserved for interpretability

Statistical Properties:

- **Feature Distributions**: Standardized features with mean≈0 and std≈1
- **Class Balance**: Original class distribution preserved through stratified sampling
- **Outlier Treatment**: Outliers retained as legitimate attack indicators rather than noise
- **Correlation Structure**: Reduced multicollinearity while maintaining predictive relationships

Quality Assurance Metrics:

- **Completeness**: 100% data integrity across all samples and features
- **Consistency**: Uniform data types and ranges across entire dataset
- **Relevance**: Feature selection ensures all retained variables contribute to classification

D. Dataset Split

i. Splitting the Dataset for Training and Validation

Strategic dataset partitioning ensures robust model evaluation and prevents data leakage:

Binary Classification Split:

- **Training Set**: 80% of data for model training (maintaining class distribution)
- **Testing Set**: 20% of data for unbiased performance evaluation
- **Stratification**: Class proportions preserved in both training and testing sets
- **Random State**: Fixed seed ensures reproducible train/test splits

Multiclass Classification Split:

- **Attack Subset**: Isolated attack samples for specialized multiclass training
- **Proportional Split**: 80/20 division with attack type stratification
- **Class Preservation**: All attack types represented in both training and testing sets

Code:

```
# ============================
# Train-Test Split
# ============================
X_train, X_test, y_train, y_test = train_test_split(
    X_selected, y_binary, test_size=0.2, random_state=42, stratify=y_binary
)
```

ii. Identifying Sufficient Real-Time Data for Testing

Real-time deployment considerations for production IoT environments:

Test Set Sufficiency:

- **Sample Size**: 20% test set provides adequate statistical power for performance estimation
- **Diversity**: Test set contains representative samples of all attack types and normal traffic
- **Generalization Assessment**: Performance metrics on test set indicate real-world applicability

Real-Time Data Pipeline:

- **Preprocessing Consistency**: Saved preprocessing components ensure consistent transformation
- **Scalability**: Two-stage classification optimizes computational efficiency for real-time processing

3. Explore the Algorithm/Architecture

This project implements a sophisticated two-stage ensemble learning approach for IoT intrusion detection, utilizing Random Forest and Extra Trees Classifier algorithms optimized for real-time network security applications.

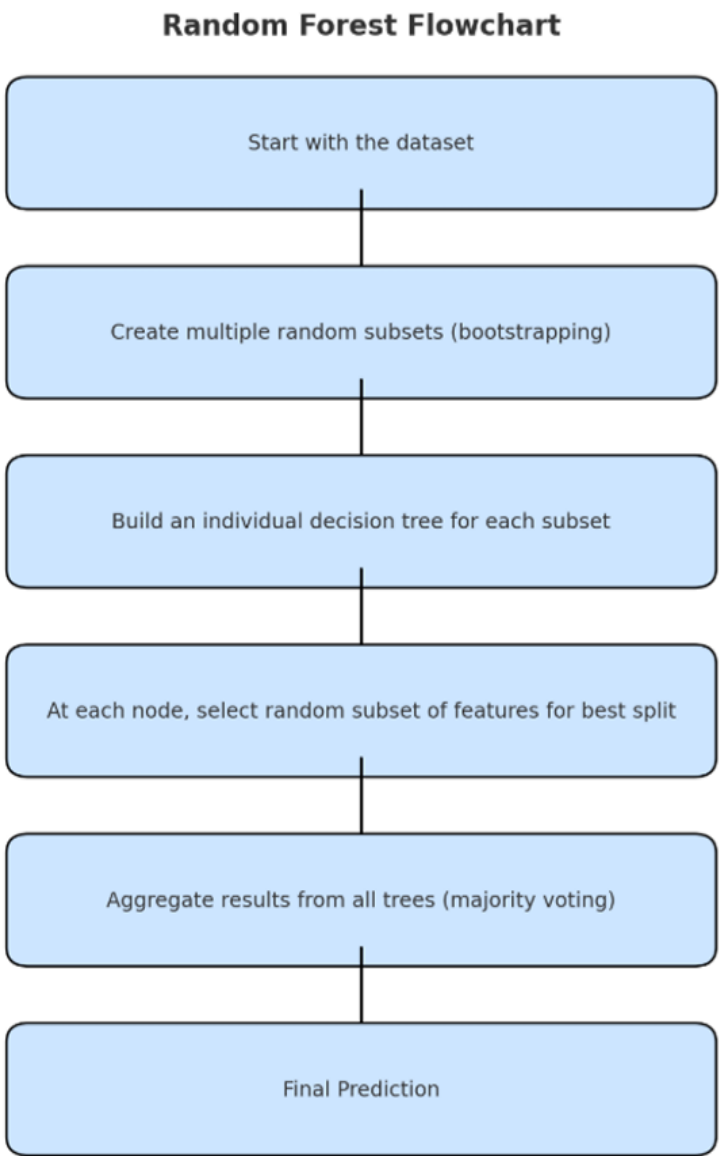
A. Describe The Algorithm(s) And Justify Why These Algorithms Are Chosen

Random Forest (RF)

Random Forest is a powerful ensemble learning method that constructs multiple decision trees during training and outputs the class that represents the mode of the individual trees' classifications. Each tree is built using a random subset of training data (bootstrap sampling) and considers only a random subset of features at each split decision.

Justification for Selection:

- **High Accuracy**: Proven effectiveness in network security applications with excellent generalization capabilities
- **Robustness**: Resistant to overfitting due to ensemble nature and random feature selection
- **Feature Importance**: Provides interpretable feature importance scores crucial for security analysis
- **Scalability**: Efficient parallel processing capabilities suitable for real-time IoT environments
- **Versatility**: Handles both numerical and categorical features without extensive preprocessing



C. Pseudocode For The Algorithm(s)

Random Forest Algorithm (Step-by-Step)

Random Forest (Step-by-Step Explanation) 1. Select a random subset of data: The algorithm randomly picks a portion of the dataset and uses it to build a single decision tree.

2. Choose a random subset of features: At each step of building the tree, the algorithm only considers a random fraction of the available features to make decisions.

3. Build a decision tree: A tree is built by asking a series of questions about the data (e.g., "Is sleep_quality less than 3?"). The goal is to create "branches" that lead to the most accurate final predictions.

4. Repeat: Steps 1-3 are repeated many times to create a "forest" of different trees. Each tree is unique because it's built on a different random subset of data and features.

5. Vote for the final prediction: When a new data point needs to be classified, it is run through every single tree in the forest. Each tree gives its own prediction (e.g., "high stress" or "low stress"). The final outcome is the prediction that gets the most "votes" from all the trees.

D. Algorithm Implementation in Project Architecture

Two-Stage Classification Strategy

Stage 1: Binary Classification (Normal vs Attack)

- **Purpose**: Rapid initial filtering of network traffic for real-time responsiveness
- **Models**: Both Random Forest and Extra Trees trained independently for comparison
- **Input**: All preprocessed features from network flow data
- **Output**: Binary classification (0=Normal, 1=Attack) with confidence scores
- **Optimization**: High recall prioritized to minimize false negatives (missed attacks)

Stage 2: Multiclass Classification (Attack Type Identification)

- **Purpose**: Detailed threat analysis for confirmed attacks
- **Models**: Specialized classifiers trained exclusively on attack data
- **Input**: Feature vectors from samples classified as attacks in Stage 1
- **Output**: Specific attack type (DDoS, DoS, Reconnaissance, Information Theft)
- **Optimization**: Balanced accuracy across all attack types for precise threat identification

Performance Evaluation Strategy

Comprehensive Metrics Assessment:

- **Accuracy**: Overall classification correctness
- **Precision**: True positive rate for each class (attack detection reliability)
- **Recall**: Sensitivity to detect actual attacks (security coverage)
- **F1-Score**: Harmonic mean balancing precision and recall
- **Confusion Matrix**: Detailed misclassification analysis

4. GUI Design

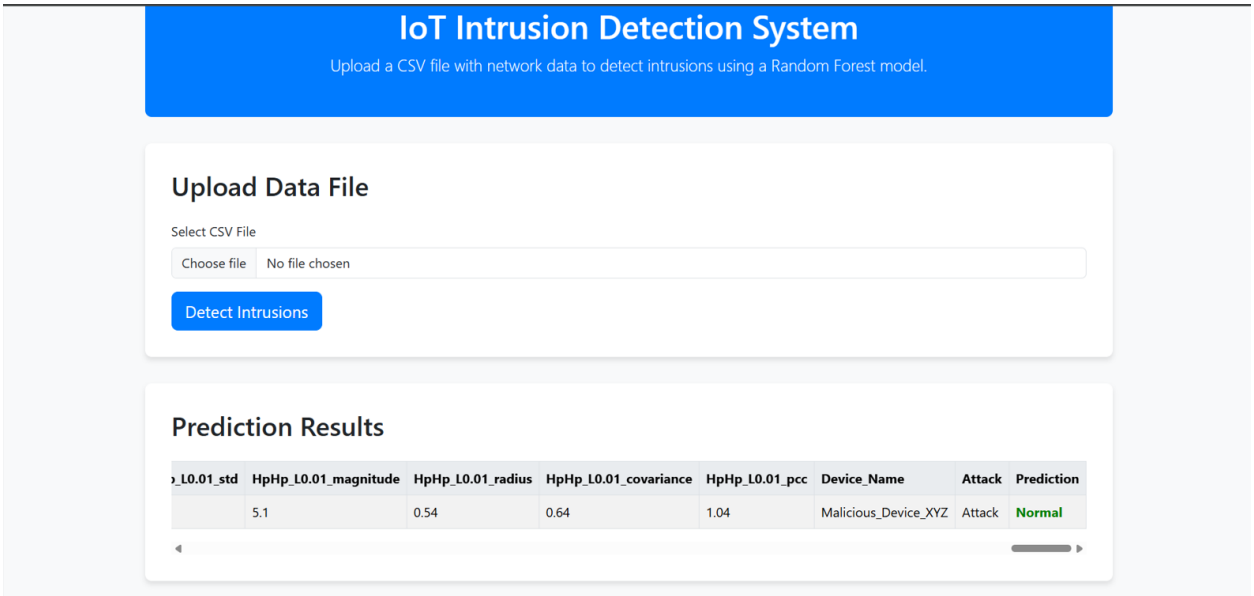
The project's goal is to deploy the trained machine learning model in a user-friendly web application. The GUI is designed to be simple and intuitive, providing a clear interface for users to interact with the Intrusion Detection System.

a. Sketch the UI proposed for deployment

The proposed UI consists of two main screens: a Homepage for file uploads and a Result Page for displaying the classification output. The design is clean and modern, ensuring ease of use for all users.

Homepage The homepage is the primary interface where users initiate the process. It is centered around a simple upload form. The title of the page clearly states the application's purpose. A "Choose File" button allows users to select a network traffic data file to be analyzed. Once a file is selected, an "Upload and Predict" button becomes active, triggering the model's prediction process.

Result Page After the model processes the uploaded file, the user is redirected to the result page. This page displays the classification outcome in a clear and concise manner. It shows the filename of the uploaded data and the final prediction from the model, indicating whether the traffic is "Normal" or an "Attack." The page also includes an option to return to the homepage to upload another file.



b. Interface for each screen and describe the components

The interface is built using a modern technology stack to ensure a responsive and aesthetically pleasing user experience.

Homepage Components:

- Title: A prominent title, such as "Intrusion Detection System for IoT Devices," clearly identifies the application.
- File Input: A user-friendly component with a "Choose File" button, allowing the user to browse their local file system to select a network traffic data file.
- Upload Button: An "Upload and Predict" button that sends the selected file to the backend for processing by the machine learning model.
- Technology Stack: The front-end is built using HTML and CSS for structure and styling, and Bootstrap for a responsive layout. The backend uses Python and a framework like Django to handle file uploads and model inference.

Result Page Components:

- Prediction Display: The core of this screen, it displays the model's final output, clearly stating whether the uploaded traffic is classified as "Normal" or a specific "Attack Type."
- Confidence Score: The prediction is accompanied by a confidence score, giving the user an idea of the model's certainty.
- Action Button: A button labeled "Upload another file" that allows the user to navigate back to the homepage and perform a new classification.
- Technology Stack: The front-end is again built with HTML, CSS, and Bootstrap. The backend is where the pre-trained model is loaded and used to make predictions on the new data.