**New GasPipeline Multiclass Technical Report Command**

**Date**: Jan 22, 2025

**Tech Report Filename**: Multiclass\_GasPipeline\_Tech\_Report\_Command.docx

This page is to be printed on the back of the cover page. This text is hidden so it will not print.

Table of Contents

[1 Introduction 6](#_Toc192164927)

[2 Research Objectives 8](#_Toc192164928)

[3 Uncertainties and Research Challenges 9](#_Toc192164929)

[4 SCADA Architecture 10](#_Toc192164930)

[4.1 Sensor/Actuator: 11](#_Toc192164931)

[4.2 PLC/RTU/IED: 11](#_Toc192164932)

[4.3 SCADA Gateways: 12](#_Toc192164933)

[4.4 SCADA Server - MTU 12](#_Toc192164934)

[4.5 Data Historian: 12](#_Toc192164935)

[4.6 HMI: 13](#_Toc192164936)

[5 Gas Pipeline SCADA Testbed and Dataset 13](#_Toc192164937)

[5.1.1 Data Preprocessing and Exploratory Data Analysis (EDA) on Gas Pipeline SCADA System Dataset 18](#_Toc192164938)

[6 Methodology/Approach 20](#_Toc192164939)

[6.1 Feature Ranking 21](#_Toc192164940)

[6.1.1 Statistical Properties of Dataset (SP) 22](#_Toc192164941)

[6.1.2 WFI Using Random Forest 24](#_Toc192164942)

[6.1.3 WFI Using XGboost 25](#_Toc192164943)

[6.1.4 Feature Reduction Analysis 26](#_Toc192164944)

[6.2 Selective Feature Selection (SPFS) 27](#_Toc192164945)

[6.3 Recursive Feature Elimination (RFE) 28](#_Toc192164946)

[6.4 Cross-Validation 30](#_Toc192164947)

[6.5 Metrics Evaluation 30](#_Toc192164948)

[6.6 Averaging Methods for Metrics Evaluation 31](#_Toc192164949)

[6.6.1 Micro Averaging 31](#_Toc192164950)

[6.6.2 Macro Averaging 32](#_Toc192164951)

[6.6.3 Weighted Averaging 33](#_Toc192164952)

[6.6.4 Evaluation of Averaging Methods 33](#_Toc192164953)

[7 Design/Framework/Testbed 34](#_Toc192164954)

[8 Results and Discussion 36](#_Toc192164955)

[8.1 Overview of Result Metrics 36](#_Toc192164956)

[8.1.1 Key Performance Metrics 38](#_Toc192164957)

[8.1.2 Performance Trends 38](#_Toc192164958)

[8.1.3 Comparison by Model Type, Ranking Method, and Feature Selection 39](#_Toc192164959)

[8.2 Ranking Results 40](#_Toc192164960)

[8.2.1 Statistical Properties Ranking Results 40](#_Toc192164961)

[8.2.2 Weighted feature selection ranking results using RandomForest Classifier 42](#_Toc192164962)

[8.2.3 Weighted feature selection ranking results using XGBoost Classifier 44](#_Toc192164963)

[8.3 Summary of Best-Performing Models 46](#_Toc192164964)

[8.3.1 Best F1 Scores 46](#_Toc192164965)

[8.3.2 Best accuracy 47](#_Toc192164966)

[8.3.3 Performance of Models with Limited Features 51](#_Toc192164967)

[8.3.4 Performance on time taken 54](#_Toc192164968)

[8.4 Summary of Findings 56](#_Toc192164969)

[8.4.1 Key Observations 58](#_Toc192164970)

[8.4.2 Summary of Best-Performing Configurations 58](#_Toc192164971)

[9 Conclusion 60](#_Toc192164972)

[10 Scope of Future Researche 62](#_Toc192164973)

[11 References 62](#_Toc192164974)

[Appendix A: xxx 63](#_Toc192164975)

List of Figures

[Figure 1 Label Distribution in Function Dataset 19](#_Toc191292396)

[Figure 2 Flowchart of methodology 20](#_Toc191292397)

[Figure 3 Flowchart of RFE 29](#_Toc191292398)

[Figure 4 Flowchart of model evaluation process 36](#_Toc191292399)

[Figure 5 Impact of Feature Reduction on Model Performance 37](#_Toc191292400)

[Figure 6 Content Guidelines 75](#_Toc191292401)

List of Tables

[Table 1 Description of features from New Gas Pipeline dataset 15](#_Toc192974004)

[Table 2 Example Feature Ranking based on metrics from the Function dataset 23](#_Toc192974005)

[Table 3 Feature importance ranking using Random Forest. 24](#_Toc192974006)

[Table 4 Feature importance ranking using XGBoost. 25](#_Toc192974007)

[Table 5 Ranking features in original dataset based on statistical properties 40](#_Toc192974008)

[Table 6 Feature importance rankings for the function dataset using the RandomForest classifier 42](#_Toc192974009)

[Table 7 Feature importance rankings for the original dataset using the XGBoost classifier 44](#_Toc192974010)

[Table 8 Top 30 best preforming combinations based on greatest F1 score 46](#_Toc192974011)

[Table 9 Best 30 accuracy scores (Micro Averaging) 48](#_Toc192974012)

[Table 12 Ranking of Model Configurations by Total Computational Time (Best 10 F1 Scores) 54](#_Toc192974013)

[*Table 14 Full Evaluation Log csv file* 76](#_Toc192974014)

[Table 15: Formatting Guideline 104](#_Toc192974015)

# Introduction

Supervisory Control and Data Acquisition (SCADA) systems are integral to the operation of critical infrastructure, including utilities like water, electricity, gas, and transportation systems. These systems provide real-time monitoring and control of industrial processes, ensuring efficient and safe operations. SCADA systems gather data from sensors and devices in the field, process this data, and present it to human operators in a comprehensible format, enabling them to make informed decisions and respond swiftly to any issues.

SCADA systems play a crucial role in maintaining the stability and reliability of critical infrastructure. For instance, in the energy sector, SCADA systems help manage the distribution of electricity by monitoring the grid's performance and detecting faults before they escalate into major outages. In the water supply industry, SCADA ensures the quality and availability of water by controlling pumps, valves, and chemical treatment processes. The transportation sector relies on SCADA for the smooth operation of traffic lights, railway signals, and other essential systems.

Given the vital functions that SCADA systems perform, any disruption can have significant repercussions. Ensuring the security and integrity of SCADA systems is paramount to safeguarding the continuous and reliable operation of critical infrastructure.

Vulnerabilities of SCADA Systems

Despite their importance, SCADA systems are inherently vulnerable to cyber threats [1]-[11]. Originally designed for isolated and controlled environments, many SCADA systems were not built with robust security features to counter modern cyber threats. The increasing interconnectivity and integration of SCADA with corporate IT networks and the internet have exposed these systems to a broader range of attack vectors.

One notable example of a SCADA-related cyber incident is the attack on the Colonial Pipeline in May 2021 [6]. The ransomware attack forced the pipeline operator to shut down its operations, leading to significant fuel supply disruptions across the southeastern United States. This incident highlighted the susceptibility of SCADA systems to cyberattacks and the potential for such breaches to cause widespread economic and social disruption.

Other examples of SCADA system breaches include the 2015 cyberattack on Ukraine's power grid [7,8], which resulted in widespread power outages, and the 2010 Stuxnet worm, which targeted Iran's nuclear facilities. These incidents underscore the critical need for enhanced security measures to protect SCADA systems from malicious actors.

Machine Learning and Deep Learning for Anomaly and Attack Detection

The increasing complexity and frequency of cyber threats necessitate advanced solutions for detecting and mitigating attacks on SCADA systems. Machine learning (ML) and deep learning (DL) offer promising approaches to enhance the security of these systems by identifying anomalies and potential attacks in real-time [12-17].

Machine learning algorithms can analyze vast amounts of data generated by SCADA systems to establish baseline behaviors and detect deviations that may indicate cyber threats. These algorithms can be trained to recognize patterns associated with normal operations and flag any unusual activities for further investigation. For example, ML-based systems can detect abnormal network traffic patterns, unauthorized access attempts, or unexpected changes in system configurations.

Deep learning, a subset of machine learning, leverages artificial neural networks to model complex relationships in data. Deep learning techniques can improve the accuracy and speed of anomaly detection by processing high-dimensional data and identifying subtle indicators of attacks that traditional methods might overlook. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are particularly effective in analyzing time-series data from SCADA systems, enabling the detection of sophisticated and stealthy attacks.

By integrating ML and DL-based anomaly detection systems, SCADA operators can achieve more robust and proactive security postures. These technologies can provide continuous monitoring, real-time alerts, and automated responses to potential threats, thereby reducing the risk of successful cyberattacks and minimizing their impact on critical infrastructure.

# Research Objectives

The main objective of this research study is to investigate different machine learning classification techniques, data normalization, automated feature selection, and data balancing techniques to explore and achieve robust Machine Learning based models for Intrusion Detection System (IDS) of gas pipeline SCADA system.

In this work, we focus on understanding the data and their processing techniques before modeling through various experiments. This allowed us to select appropriate data processing and feature selection methodologies as well as enable us to develop a new and efficient systematic strategy for obtaining high-performance M/DL model. We proposed in this work a Robust Strategy for Best Machine Learning Model Selection (RS-BMLMS) and validated it on Gas Pipeline dataset as provided by Mississippi State University and Oak-Ridge National Laboratory for industrial control system security research [17]. In this work, we focus on detecting two specific attacks namely Naïve Malicious Response Injection (NMRI) and Complex Malicious Response Injection (CMRI) that are related to response messages/data.

# Uncertainties and Research Challenges

The major research challenge of this work is to design and develop robust machine learning based anomaly detection and intrusion detection models that can be used for intrusion detection in gas pipeline SCADA system.  The developed models will optimize the trade-off between multiple objectives such as reduce the training time, generalize the model to detect new attacks, improve the model accuracy, and minimize the false alarms. At present most of the intrusion detection systems used in practice are signature-based. A signature-based intrusion detection system detects attacks by looking for specific patterns, such as byte sequences in SCADA application, or known malicious instruction sequences used by malware. However, it is difficult to detect new attacks, for which no pattern is available. Machine learning-based anomaly detection can be used for SCADA systems, and it is expected that it will work better than signature-based algorithms for zero-day (new) attacks.  However, the high false alarm rate often limits practitioners to use machine learning based system for intrusion detection. Also, detecting new attack requires the model to be generalized which in turn requires the model to be trained with huge volume of data. Although these models often show excellent performance on the data population they are trained with, these models may not work well with new data.

The uncertainty remains whether the model training can be done efficiently, and model can be generalized to work on new data with high accuracy and low false alarm rate.

Towards overcoming the above uncertainties and challenges, different approaches can be taken such as using ensemble models, automated feature selection techniques and optimizing hyper-parameters. It is expected that these models will be suitable for off-line periodic training and on-line intrusion detection in gas pipeline SCADA system.

The data preprocessing and feature selection impact the model performance significantly. Therefore, in this study we focus on good model development and best model selection strategies by performing a series of experiments as will be discussed in details in subsequent sections.

# SCADA Architecture

A typical SCADA system consists of SCADA Master Terminal Unit (MTU) that is connected to either a remote location unit or connected locally to a local location unit that contains controllers, remote terminal units (RTU) or Programmable Logic Controllers (PLC) at one end and other end is connected to the control system consist of Human Machine Interface (HMI), data historian & workstation. Figure 1 below shows a simple SCADA system architecture [9]. In the following we briefly describe the main components of the SCADA system.

A screenshot of a cell phone

Description automatically generated

Figure 1 SCADA System Architecture [13]

## Sensor/Actuator:

Sensors within industrial field level check whether parts are present, size of the part, color of the part, and whether the product is full or empty, good or bad. Sensors are also used to ensure safety of the equipment, product and operators. A common classification of sensors is contact and non-contact. Another way to classify sensors is by analog and digital.

## PLC/RTU/IED:

SCADA system RTUs are microprocessor-based devices designed to monitor and collect data as well as perform some control functions. These functions depend upon the requirements of the specific industry and the technology used by the RTU manufacturer. The industry has developed smart RTUs with the capability to execute logic and PID closed loop control. The SCADA market has moved away from the traditional proprietary RTUs and toward multi-purpose devices, such as programmable logic controllers (PLCs) and intelligent electronic device (IEDs). However, traditional, purpose-built RTUs are still likely to be used in environmentally challenging applications, such as where extremes of moisture, temperature, or humidity are present, or low power consumption is required [14].

IEC 61131-3, international standard is used to define PLC/RTU programming languages and concepts. Programming languages are divided into two main sections to represent the five main programming languages. Generally, ladder logic and block diagram are the two common methods for the PLC programming.

## SCADA Gateways:

Different devices at field site, uses different types of protocols (e.g. Modbus, serial, fieldbus, dnp3 etc.) to communicate amongst each-other. SCADA gateways are used to connect different protocols to an IP network via ethernet, GSM, Wi-Fi, Zigbee, LoRa, GPRS etc. These devices might not be air-gapped, as they are having antennas, and they are interconnected by different wireless protocols [10]. Generally, these devices are soft target devices to intercept and get into the network by firmware analysis and reverse engineering methodology.

## SCADA Server - MTU

Master Terminal Unit (MTU) is located at the control center. MTU acts as heart of the SCADA system. It issues the commands to the field devices like PLCs/RTUs which are located at remote places from the control center to gather the required data from the plant floor. Furthermore, it processes the information, stores the important status information at data historian and display the information in the form of graphs, curves and tables on HMI to help in taking control decisions. Moreover, the communication between MTU and field control devices are only initiated by the program resides in the MTU. Hence, MTU act as a Master and RTU is the slave, however the level of communication is considered as peer-to-peer communication.

## Data Historian:

A Data Historian (also known as a Process Historian or Operational Historian) is a software program that records and retrieves production and process data by time; it stores the information in a time series database that can efficiently store data with minimal disk space and fast retrieval [16]. Time series information is often displayed in a trend or as tabular data over a time range (ex. the last day, last 8 hours, last year).

## HMI:

An HMI is a software application that presents information to the users about the state of the process and to accept the user and control instructions. HMI provides a graphical representation of the control system process and provides real time data acquisition. It monitors data and makes it available to users, as well as provides an interface for inputs from the users. One of the advantages that HMI provides is that they provide an interface between the user and the machine within the factory floor, a centralized control unit and a local interface of the process for quick monitoring and adjusting.

# Gas Pipeline SCADA Testbed and Dataset

Mississippi State University’s in-house SCADA lab has a gas pipeline system to collect data for cyber attack research as shown in Figure 2. The system consists of three major components: sensors and actuators, a communication network, and supervisory control as shown in the picture below [17].   This section summarizes the testbed components and the methods by which the dataset was collected using this testbed.  Detailed information can be found in [17].

The gas pipeline has two actuators namely a pump and a solenoid as well as a pressure sensor. The actuators are used to maintain the pressure. The pressure is set by the supervisory control system. The gas pipeline has three main system modes: automatic, manual, off. In automated mode, there are two schemes – the pump mode, which turns the pump on and off to keep the pressure in the pipe at the set point and the solenoid mode, in which a relief valve controlled by a solenoid is opened and closed to regulate pressure. Both the pump and solenoid modes used a Proportional-Integral-Derivative (PID) control scheme. In manual mode, the operator must manually control the pump and solenoid.

The next component is the communication network in which the protocol used is serial Modbus RTU. Modbus packets include a header and a payload. For Modbus over a Serial Line, a packet includes a device address, function code, payload, and a cyclic 21 redundancy code (CRC) or linear redundancy code (LRC).

A collage of pipes and a pressure gauge

Description automatically generated

Figure 2 Gas pipeline SCADA testbed [17]

The gas pipeline database of cyber attacks was originally created by Tommy Morris and Wei Gao in 2012 [18].  However, study on the original dataset shows some major flaws in the dataset and it was declared unsuitable for machine learning research [17]. Later Ian Turnipseed of Mississippi State University collected more realistic cyber attack gas pipeline dataset known as “new gas pipeline dataset” and made it publicly available for research community.  In our study the new gas pipeline dataset is used.

The “new gas pipeline” dataset is available in two formats. The first form is a comma separated value (CSV) text file, and the second form is an Attribute Relationship File Format (ARFF). The ARFF dataset was created to make it compatible with Waikato Environment for Knowledge Analysis (WEKA), a tool that has a comprehensive list of machine learning algorithms and has been used by many researchers world-wide for testing the performance of specific machine learning algorithms [14]. Although we are not using WEKA, we preprocessed the ARFF dataset to make it ready for training different classification models.  Each record or instance in the dataset represents one packet being delivered to either the MTU or to the RTU. Each instance in the dataset contains SCADA application information along with payload information.   SCADA systems have fixed network topologies and the transactions between the components are repetitive and regular as opposed to IT network whereas the SCADA application data is very dynamic and irregular in nature. The second category of features is the payload information. The payload information provides information about the gas pipeline’s state, settings, and parameters. These values are critical to detect if any anomaly is found in the system which could be due to malfunction of the system or malicious activities by the cyber attackers. There are total of 274,628 instances or rows in the dataset and each row contains twenty columns. The columns are commonly referred to as features. These features are summarized with a brief description in Table 1.

Table 1 Description of features from New Gas Pipeline dataset

|  |  |  |  |
| --- | --- | --- | --- |
| Feature Id | Feature Symbol | Type | Description |
| 1 | address | real | The address of the slave device. Each slave device in the Modbus is assigned a 8-bit address to identify the slave device the master is communicating to and from |
| 2 | function | real | The function codes are primarily used in the gas pipeline to indicate a read (0x03) and write commands (0x16). But there are possibilities of total 256 such commands. A denial of service attack can be launched by setting a function code of 0x08 which corresponds to diagnostic mode where the device would be always in listening mode. |
| 3 | length | real | Length of the Modbus frame. This is fixed for each command and response frame. Frames that are not of specific length can be easily detected as attack |
| 4 | setpoint | real | This value controls the pressure in the gas pipeline |
| 5 | gain | real | Gain parameter of the PID controller |
| 6 | reset rate | real | Reset rate parameter of PID controller |
| 7 | deadband | real | Deadband parameter of PID controller |
| 8 | cycle time | real | Cycle time parameter of PID controller |
| 9 | rate | real | Rate parameter of PID controller |
| 10 | system mode | {0,1,2} | Controls the duty cycle of the system. The following modes are valid  0 – Off  1 – Manual  2 - Automatic |
| 11 | control scheme | {0,1} | The control scheme in the gas pipeline determines whether the system will be controlled by the pump or by the solenoid. There are two schemes:  0 – Pump  1 - Solenoid |
| 12 | pump | {0,1} | This is the state of the pump when system mode is set to manual. There are two possible values:  0 – Off  1 - On |
| 13 | solenoid | {0,1} | This represents the state of the solenoid valve. There are two possible values:  0 – Closed  1 – Open |
| 14 | pressure | real | The current pressure measurement from the gas pipeline |
| 15 | crc rate | real | The Cyclic Redundancy Check (CRC) allows system to check error within a Modbus frame |
| 16 | command response | {0,1} | This value allows the IDS to learn about command and response frame. Two possible values:  0 – Response  1 – Command |
| 17 | time | real | Timestamp of the instance |
| 18 | binary result | {0,1} | Labels to indicate either attack (1) or normal (0) instance |
| 19 | categorized result | Range(0:7) | Labels to indicate the category of attack |
| 20 | specific result | Range(0:35) | Labels to indicate the specific type of attack |

### ****Data Preprocessing and Exploratory Data Analysis (EDA) on Gas Pipeline SCADA System Dataset****

In this subsection, we discuss the preprocessing and feature extraction steps applied to the gas pipeline dataset, as introduced in Section 5. For the simulation study reported in this section, the original gas pipeline dataset was preprocessed with necessary feature extraction into the following categories:

* **Dataset 1:** The entire gas pipeline dataset was preprocessed to remove data instances with a majority of missing feature values, along with features that are not relevant to this study. This resulted in a reduced dataset containing **64,100 rows (data instances) and 12 columns (11 features plus output).**
* **Dataset 2:** The full dataset was further divided into three subsets based on dataset types: **Command (C), Function (F), and Response (R).** After removing instances with missing features, the resulting subsets had sizes of **64,100 × 13 (Command), 68,848 × 4 (Function), and 141,680 × 7 (Response),** respectively.
* **Dataset 3:** Inspired by the approach described in [18,19], new features were introduced in the **Command and Response subsets** of Dataset 2 by computing delta values of the original feature values and centroids of **N past delta values (user-defined).**
* **Dataset 4:** Inspired by the method described in [20], **special imputation techniques** and **one-hot encoding** were used to preprocess the dataset into three subsets based on dataset types: **Command (C), Function (F), and Response (R).**

For this study, we specifically focus on the Command dataset, which contains 64,101 **rows and 9 features.** This dataset captures key functional attributes of the gas pipeline system and is used to analyze feature importance and optimize feature selection strategies. The experiments conducted in this report aim to formulate an efficient modeling strategy to enhance detection performance while maintaining computational efficiency. The distribution of label categories for the Command dataset is shown below in figure 1:

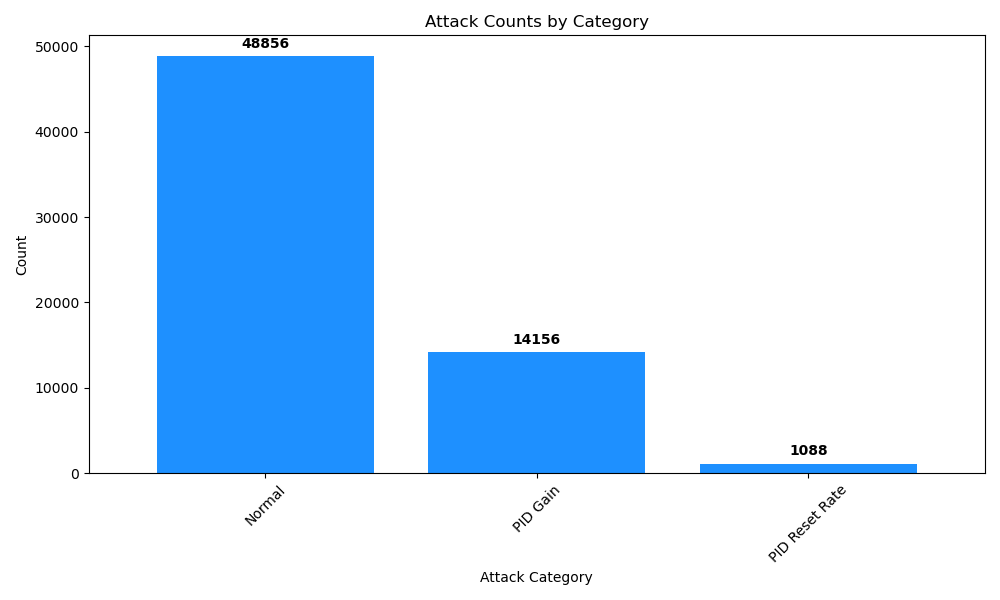


Figure 3 Label Distribution in Command Dataset

The bar chart in Figure 3 illustrates the distribution of different label categories within the Command dataset after preprocessing. As shown, the dataset is highly imbalanced, with a significantly larger number of normal instances compared to the various attack categories.

To provide a clearer understanding of the attack categories represented in the dataset, Table 2 below outlines the corresponding label numbers assigned to each cyberattack type.

*Table 2 Specific categories of cyber attacks in a SCADA system*

|  |  |  |  |
| --- | --- | --- | --- |
| Attack Number Category Description | | | |
| Setpoint | 1–2 | MPCI | Changes the pressure set point outside and inside of the range of normal operation |
| PID gain | 3–4 | MPCI | Changes the gain outside and inside of the range of normal operation |
| PID reset rate | 5–6 | MPCI | Changes the reset rate outside and inside of the range of normal operation |
| PID rate | 7–8 | MPCI | Changes the rate outside and inside of the range of normal operation |

The dataset's severe class imbalance must be considered when training and evaluating machine learning models. The high prevalence of normal instances compared to attack categories could impact predictive performance, necessitating strategies such as weighted loss functions or data resampling techniques to improve detection capabilities.

# Methodology/Approach

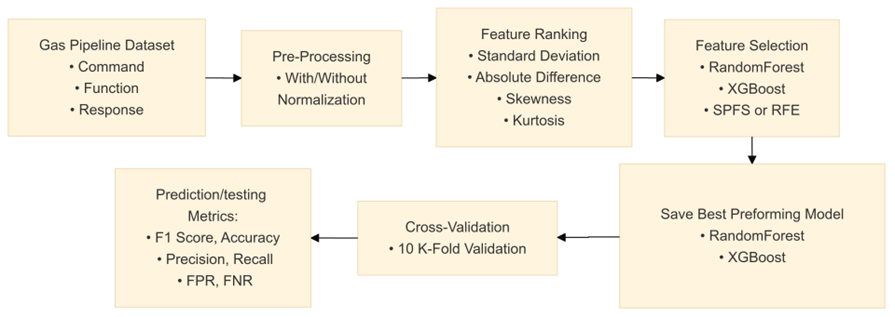


Figure 4 Flowchart of methodology

The methodology begins with data ingestion and preprocessing, where the script reads a CSV file containing gas pipeline data (e.g., commands, responses, and functions), initializes paths for storing results, metrics, and models, and optionally normalizes (e.g., Min–Max scaling) and labels datasets if needed. Feature ranking is performed using statistical measures such as standard deviation, absolute difference (mean vs. median), skewness, and kurtosis to assign importance scores to features, with an overall rank determined by aggregating these metrics. Feature selection is conducted using methods like Recursive Feature Elimination (RFE), which iteratively removes less important features, or Sequential Progressive Feature Selection (SPFS), which incrementally builds a feature set by retaining only features that improve model performance. Models, such as Random Forest or XGBoost, are trained during this process, with performance evaluated using metrics like accuracy, precision, recall, F1 score, false positive rate (FPR), and false negative rate (FNR), while training time is also recorded for comparison. Although the current code does not explicitly include cross-validation or prediction, k-fold cross-validation can be applied after feature selection to validate model performance, and the final trained model is saved as a serialized file (e.g., .pkl) for future prediction tasks. Figure 5 illustrate proposed methodology to classify the multple attack scenario and normal events using gas pipeline dataset.

## Feature Ranking

To preprocess the dataset and identify the most significant features, we applied multiple feature ranking techniques. These methods aim to quantify each feature’s importance in relation to the target variable, ensuring a refined dataset for improved model performance.

Our approach combining statistical analysis and machine learning-based importance to achieve a comprehensive ranking. First, we analyzed the dataset’s statistical properties (SP), evaluating metrics such as standard deviation, skewness, and kurtosis to identify key patterns and variations.

To further refine feature ranking, we employed Weighted Feature Importance (WFI) using two machine learning models: Random Forest and XGBoost. These models assessed feature significance based on their contribution to predictive accuracy, providing a more data-driven selection process. The integration of statistical and model-based ranking ensured that only the most relevant features were retained for subsequent analysis.

### Statistical Properties of Dataset (SP)

Before applying advanced feature selection methods, we first analyzed the statistical properties of the dataset to identify features with high variability, skewed distributions, or extreme outliers that could influence model performance. Understanding these statistical properties allowed us to make informed decisions on which features to prioritize or remove before proceeding with more sophisticated ranking techniques.

To assess the characteristics of each feature, we computed several key statistical metrics. Standard deviation was used to measure variability, as features with higher variance are typically more informative for classification. The absolute difference between the mean and median helped highlight skewed distributions, which often indicate the presence of outliers or unique patterns in the data. Skewness was calculated to quantify data asymmetry, as highly skewed features may introduce biases into the model. Finally, kurtosis was measured to assess the "tailedness" of a distribution, as extreme values could indicate whether a feature had heavy-tailed or light-tailed characteristics, potentially affecting predictive power.

The ranking process was implemented using Python, where each statistical metric was computed for all features. Features were then ranked individually based on their values for each metric. To create a comprehensive ranking system, a total rank was assigned to each feature by summing its rankings across all four metrics, with lower total ranks indicating higher feature importance. This ranking provided a structured approach to selecting the most relevant features for subsequent modeling steps.

The final output of this process was a detailed ranking table containing individual metric values, rankings for each metric, and a consolidated total rank. This table served as a foundation for further feature selection methods, ensuring that the most statistically significant features were retained for model training and evaluation.

An example output from the statistical properties ranking method is shown in Table 2:

Table 2 Example Feature Ranking based on metrics from the Function dataset

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | Std Dev | Abs Diff | Skew | Kurtosis | Std Dev Rank | Abs Diff Rank | Skew Rank | Kurtosis Rank | Total Rank |
| address | 0.02 | 0.00 | 21.10 | 695.97 | 1 | 1 | 1 | 16 | 19 |
| pressure | 0.04 | 0.00 | 20.20 | 419.63 | 2 | 2 | 2 | 15 | 21 |
| reset\_rate | 0.08 | 0.00 | 3.01 | 20.31 | 5 | 3 | 5 | 10 | 23 |
| setpoint | 0.14 | 0.00 | 0.96 | 2.90 | 8 | 4 | 9 | 7 | 28 |
| rate | 0.09 | 0.01 | 8.09 | 69.06 | 6 | 6 | 3 | 14 | 29 |
| cycle\_time | 0.08 | 0.03 | 2.49 | 21.71 | 4 | 8 | 6 | 11 | 29 |
| deadband | 0.17 | 0.02 | 0.17 | -0.97 | 9 | 7 | 12 | 5 | 33 |
| function | 0.14 | 0.03 | 5.12 | 27.34 | 7 | 9 | 4 | 13 | 33 |
| system\_mode | 0.40 | 0.25 | 1.16 | -0.42 | 11 | 11 | 8 | 6 | 36 |
| gain | 0.07 | 0.01 | -4.83 | 26.24 | 3 | 5 | 16 | 12 | 36 |
| crc\_rate | 0.42 | 0.28 | 0.09 | -1.97 | 12 | 12 | 13 | 1 | 38 |
| pump | 0.47 | 0.32 | 0.77 | -1.41 | 13 | 13 | 10 | 4 | 40 |
| solenoid | 0.50 | 0.45 | 0.19 | -1.96 | 15 | 15 | 11 | 2 | 43 |
| length | 0.19 | 0.04 | -4.47 | 18.80 | 10 | 10 | 15 | 9 | 44 |
| control\_scheme | 0.47 | 0.34 | -0.67 | -1.54 | 14 | 14 | 14 | 3 | 45 |
| Label | 1.81 | 0.78 | 2.12 | 3.18 | 16 | 16 | 7 | 8 | 47 |

### WFI Using Random Forest

Random Forest assigns importance scores using the Gini impurity criterion, which measures the reduction in node impurity caused by each feature.

**Implementation:**

1. The dataset was split into features (X) and the target variable (y).
2. A RandomForestClassifier was trained on the full dataset.
3. Feature importance scores were extracted from the model’s feature\_importances\_ attribute.
4. Features were ranked based on their importance, with higher values indicating greater predictive power.

**Outputs:**

Tabel 3 shows an example of feature ranking based on Random Forest WFI.

Table 3 Feature importance ranking using Random Forest.

|  |  |
| --- | --- |
| Name | Random Forest Feature Importance |
| function | 0.170343 |
| crc\_rate | 0.14766 |
| length | 0.135827 |
| cycle\_time | 0.094819 |
| reset\_rate | 0.093911 |
| setpoint | 0.088682 |
| gain | 0.074429 |
| deadband | 0.071171 |
| rate | 0.047493 |
| pump | 0.027476 |
| system\_mode | 0.025344 |
| control\_scheme | 0.008638 |
| solenoid | 0.006637 |
| address | 0.006363 |
| pressure\_measurement | 0.001208 |

### WFI Using XGboost

We also applied Weighted Feature Importance (WFI) using XGBoost, a gradient boosting method known for its robustness in feature selection. Unlike Random Forest, XGBoost provides gain-based importance scores, which indicate how much each feature contributes to reducing the model's overall loss function.

**Implementation:**

1. The dataset was split into input features (X) and target variable (y).
2. The class labels were encoded using LabelEncoder to ensure sequential class representation.
3. An XGBoost classifier was trained using the encoded labels.
4. The feature\_importances\_ attribute of the trained model was used to compute importance scores.
5. The features were ranked based on their importance, with higher values indicating greater influence on predictions.

**Outputs:**

The WFI process produced a ranked list of features based on their importance in the XGBoost model. The results were stored in a table, as shown in Table 4:

Table 4 Feature importance ranking using XGBoost.

|  |  |
| --- | --- |
| Name | XGBoost Feature Importance |
| length | 0.349779 |
| function | 0.183473 |
| crc\_rate | 0.175295 |
| rate | 0.154045 |
| pump | 0.032862 |
| gain | 0.021015 |
| cycle\_time | 0.018802 |
| system\_mode | 0.013917 |
| reset\_rate | 0.011271 |
| setpoint | 0.011252 |
| deadband | 0.009006 |
| control\_scheme | 0.008318 |
| solenoid | 0.0062 |
| pressure\_measurement | 0.004765 |
| address | 0 |

These ranked features were used as input for subsequent feature selection methods, ensuring the most relevant attributes were retained.

### Feature Reduction Analysis

After ranking the features using SP and WFI methods, we conducted an in-depth analysis to examine how reducing the number of features affected model performance. The primary objective of this analysis was to determine the smallest subset of features that could maintain high predictive accuracy while minimizing computational complexity.

To achieve this, we first ranked the features using two distinct approaches: the SP (Statistical Properties) method and the WFI method, which utilizes Random Forest and XGBoost for feature importance evaluation. The ranked feature list generated from these methods served as the foundation for selecting different feature subsets for further testing.

Following this, we implemented an iterative feature removal and selection process. We systematically tested various feature set sizes by progressively eliminating the least important features, removing the top one to five features from the ranked list. Additionally, we explored the performance of minimal feature sets by selecting only the top one to five most important features to assess whether a significantly reduced subset could still yield satisfactory predictive results.

For each of these feature subsets, we trained machine learning models using Random Forest and XGBoost. These models were then cross validated using multiple averaging strategies, including micro, and weighted F1-Scores, to ensure a comprehensive assessment of their predictive performance across different evaluation metrics.

The results of this analysis provided valuable insights into the relationship between feature count and classification performance. We identified the optimal number of features necessary to sustain high predictive accuracy and observed how the removal of low-importance features influenced model effectiveness. Furthermore, this process allowed us to quantify the trade-off between feature reduction and computational efficiency, ensuring that the final model maintained strong performance without unnecessary complexity. By systematically refining the feature selection process, we successfully determined the smallest subset of features that could be used without compromising classification accuracy.

## Selective Progressive Feature Selection (SPFS)

The SPFS algorithm was applied as a dynamic feature selection technique, iteratively adding features and evaluating their contributions to model performance, specifically focusing on the F1-Score. The process began with an empty feature set, progressively incorporating features based on their ranking from the previous step. At each iteration, the selected features were used to train either a Random Forest or XGBoost classifier, after which the model was evaluated on a validation set using the F1-Score. If adding a feature improved performance, it was retained; otherwise, it was discarded. This iterative refinement resulted in a final subset of selected features that maximized predictive performance. The SPFS process also generated a comprehensive metrics table detailing F1-Score, precision, recall, false positive rate (FPR), and false negative rate (FNR) at each step.

Below is a flow chart that shows the SPFS process:

A diagram of a performance measurement

AI-generated content may be incorrect.

Figure 6 Flowchart of SPFS process

## Recursive Feature Elimination (RFE)

Recursive Feature Elimination (RFE) is a backward feature elimination algorithm designed to iteratively remove the least significant features until a specified number of features remains. This method is particularly useful for identifying and eliminating features that negatively impact model performance, thereby enhancing predictive accuracy and computational efficiency.

The implementation of RFE began with an initial setup where all features were included in the model. From there, an elimination process was carried out, in which features were removed one at a time, starting with the least important feature as determined by the model’s internal feature importance metric. After each removal, the model was retrained, and its F1-Score was recalculated to evaluate its performance. This iterative process continued until the optimal subset of features was identified based on the highest recorded F1-Score.

The RFE process produced several key outputs. First, it generated a ranking of features based on their contribution to the model’s overall performance. Additionally, it identified a subset of features that yielded the best results, similar in function to the SPFS method.

Figure 3 presents the flow chart of RFE process:

A diagram of a process flow

AI-generated content may be incorrect.

Figure 7 Flowchart of RFE

## Cross-Validation

To ensure the reliability and robustness of the model, we employed k-fold cross-validation using the selected features identified during the feature selection process. This method divides the training dataset into k equally sized folds (with k=10 in this implementation), where the model is trained on k-1 folds and validated on the remaining fold. The process is repeated k times, with each fold used as the validation set exactly once, ensuring that every data point in the training set is used for both training and validation.

The training dataset was generated through an 80/20 split of the original data, where 80% was allocated for training and 20% reserved as a separate test set for final evaluation. During cross-validation, only the training data is used, and the results of all folds are aggregated to evaluate the model's overall performance.

To determine the best-performing model, we selected the model that achieved the highest F1 score across the 10-folds. This model was then tested on the remaining 20% of the data to assess its final performance. After evaluation, the selected model was saved as a .pkl file for future prediction tasks.

## Metrics Evaluation

After cross-validation, the best model's performance was evaluated using a variety of metrics to assess its predictive accuracy and ability to generalize to unseen data. The following metrics were calculated for each fold during the cross-validation process:

* **Accuracy**: The proportion of correctly predicted instances out of the total instances. It provides a general overview of the model's performance.
* **Precision**: The ratio of true positive predictions to all positive predictions. It evaluates the model's ability to avoid false positives.
* **Recall**: The ratio of true positive predictions to all actual positive instances. It measures the model's ability to identify all relevant instances.
* **F1 Score**: The harmonic mean of precision and recall, balancing the trade-off between the two metrics.
* **False Positive Rate (FPR)**: The proportion of negative instances incorrectly classified as positive.
* **False Negative Rate (FNR)**: The proportion of positive instances incorrectly classified as negative.

These metrics were saved to a CSV file for documentation and analysis.

## Averaging Methods for Metrics Evaluation

In multi-class classification tasks, evaluation metrics such as Precision, Recall, and F1-Score are commonly averaged using different strategies to provide a holistic assessment of the model's performance across all classes. The three primary averaging methods used in this study are Micro Averaging, Macro Averaging, and Weighted Averaging. Each method offers a unique perspective on performance, making them essential for analyzing model effectiveness in different contexts.

### Micro Averaging

Micro averaging computes the metrics by aggregating the contributions of all classes before performing the final calculation. It treats every individual instance equally, regardless of class distribution. This is especially useful when class imbalance is present.

The micro-average calculations are given by:

Precision = Recall = F1 =

where:

* TP (True Positives) is the number of correctly classified instances for each class.
* FP (False Positives) is the number of misclassified instances.
* FN (False Negatives) is the number of missed instances.

Micro averaging gives an overall measure of the model's performance and is particularly useful in highly imbalanced datasets, where larger classes would otherwise dominate the results.

### Weighted Averaging

Weighted averaging is similar to macro averaging, but it weighs each class's contribution to the final score based on the number of true instances in that class.

Precision = Recall = F1 =

where:

is the proportion of total samples belonging to class , computed as

Weighted averaging provides a balanced measure that accounts for both class-wise performance and class distribution. It is particularly useful when comparing models on datasets with highly imbalanced class distributions.

### Evaluation of Averaging Methods

For each iteration of model evaluation, we tested both micro and weighted averaging to determine which provided the best performance. Initially, we also considered macro averaging, but its scores were significantly lower than the other two methods, making it unsuitable for this analysis.

During the experiments, micro averaging generally performed well in cases where class imbalance was significant, as it emphasized overall correctness rather than per-class performance. Weighted averaging provided a balanced perspective by incorporating class distributions, making it particularly useful for understanding real-world classification performance

The results of these comparisons are further analyzed in Section 8, where we discuss the impact of each averaging method on final model performance.

# Design/Framework/Testbed

To systematically evaluate different feature selection and ranking techniques, we implemented a structured framework that iterated through various combinations of ranking methods, feature selection strategies, model types, and averaging methods. The objective of this framework was to identify the most effective approach for feature selection and model training, ultimately improving classification performance while maintaining computational efficiency.

The process began by looping through each ranking method, including SP (Statistical Properties) and WFI (Weight-based Feature Importance), to establish an initial ordering of features based on their statistical significance or model-derived importance scores. Following this, we applied different feature selection techniques, such as Recursive Feature Elimination (RFE), Statistical Property-based Feature Selection (SPFS), and manual removal or retention of top-ranked features, to determine the optimal feature subset for training.

For each selected subset of features, two machine learning models—Random Forest and XGBoost—were trained and evaluated using multiple averaging strategies (micro, macro, and weighted F1-Scores) to ensure a robust comparison across different evaluation metrics. Each experiment involved training a model on the selected features and measuring its performance across accuracy, precision, recall, F1-Score, false positive rate (FPR), and false negative rate (FNR).

In addition to testing all possible feature selection techniques, we performed an iterative feature reduction analysis, systematically removing the least important features (from 1 to X) to analyze their impact on model performance. The goal was to determine the minimum number of features required to maintain high classification accuracy while improving computational efficiency. By comparing the performance of models trained on progressively smaller feature subsets, we identified potential trade-offs between feature count and predictive power.

This comprehensive approach allowed us to evaluate the effects of different feature selection and ranking methods, ultimately leading to the identification of the most effective strategy for feature reduction in gas pipeline anomaly detection. The results from these experiments provided insights into which feature selection and ranking combinations yielded the highest F1-Scores while maintaining low false positive and false negative rates.

The diagram below illustrates the automated pipeline used for feature selection and model evaluation in our study. The process begins with loading the dataset, followed by ranking the features using Statistical Properties (SP) or Weighted Feature Importance (WFI) methods, which include Random Forest and XGBoost. After ranking, different feature selection techniques such as Recursive Feature Elimination (RFE) and Statistical Properties Feature Selection (SPFS) are applied to refine the feature subset. Additionally, an alternative approach is employed where the least important *X* features are systematically removed. The selected feature subset is then used to train a machine learning model, after which the model's performance is evaluated using various metrics, including F1-Score, Accuracy, Precision, Recall, False Positive Rate (FPR), False Negative Rate (FNR), and computation time.

To comprehensively analyze the impact of feature selection on model performance, a script was developed to systematically loop through all possible combinations of ranking methods, feature selection techniques, model types (Random Forest and XGBoost), and averaging strategies (micro, macro, weighted). This ensured a thorough evaluation of different configurations, allowing for the identification of the optimal approach that maximizes predictive accuracy while minimizing feature redundancy.

A diagram of a structure

AI-generated content may be incorrect.

Figure 8 Flowchart of model evaluation process

# Results and Discussion

## Overview of Result Metrics

The performance of the models was evaluated using standard classification metrics, including Accuracy, Precision, Recall, F1 Score, False Positive Rate (FPR), and False Negative Rate (FNR). The results were analyzed across different feature selection methods and varying numbers of selected features to assess the trade-off between model complexity and predictive performance.

One of the key findings from this analysis is the impact of feature reduction on model performance. The F1 Score, a balanced measure of precision and recall, was used as the primary performance metric for evaluation. The figure below illustrates how the F1 Score changes as the number of selected features decreases.

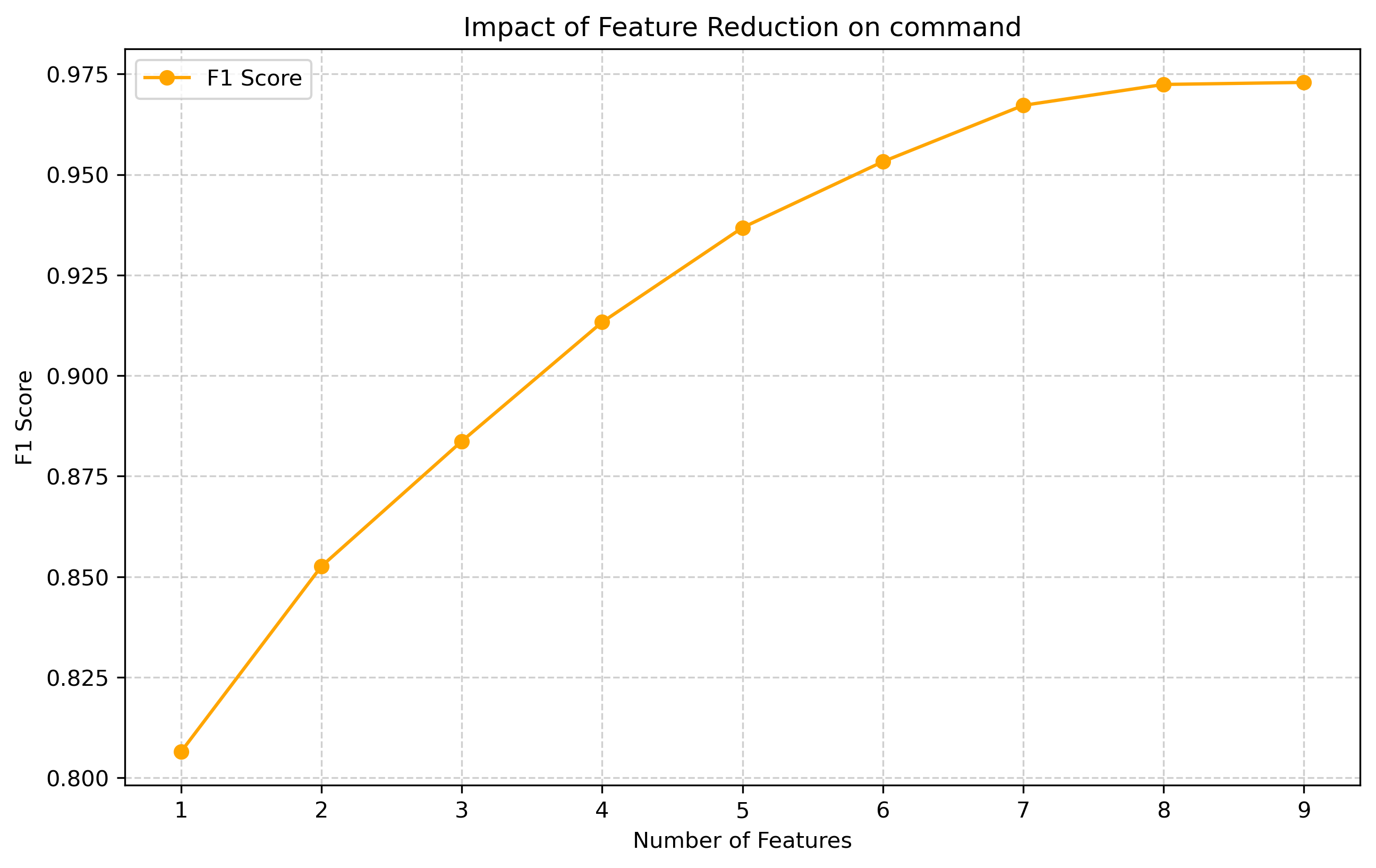


Figure 9 Impact of Feature Reduction on Model Performance (using best of SP and Micro Averaging)

From the graph, we can observe that the F1 Score steadily declines as each feature is removed. This gradual decrease shows that each feature is important to the model. This trend highlights the importance of selecting an optimal number of features to balance predictive accuracy with computational efficiency. By carefully applying feature selection techniques, we can ensure minimal loss in classification performance while improving efficiency in model training and inference.

### Key Performance Metrics

The following metrics were used to evaluate model performance:

* **Accuracy**: The percentage of correctly classified instances.
* **Precision**: The ratio of correctly predicted positive observations to total predicted positives.
* **Recall**: The ratio of correctly predicted positive observations to actual positives.
* **F1 Score**: The harmonic mean of precision and recall, providing a balanced assessment of model performance.
* **False Positive Rate (FPR)**: The proportion of false positives among all negative cases.
* **False Negative Rate (FNR)**: The proportion of false negatives among all positive cases.
* **Time Taken**: The computational time required to train and evaluate the model.

### Performance Trends and Model Comparison

A deeper analysis of performance trends and model comparisons revealed several key insights. The most noticeable decline in model performance occurred when the number of selected features fell below eight. While removing redundant features improved efficiency, eliminating too many features negatively impacted classification accuracy. Models trained with 8 to 9 features consistently achieved the highest performance, underscoring the importance of careful feature selection. Comparing Random Forest (RF) and XGBoost (XGB) showed that both models performed robustly, but each had distinct strengths. Random Forest consistently outperformed XGBoost when more features were retained, and even maintained competitive accuracy with seven or fewer features, demonstrating greater resilience under extreme feature reduction scenarios. Conversely, XGBoost frequently achieved higher accuracy and F1 scores when at least 7 to 8 features were retained, making it highly effective in scenarios with moderate feature counts.

Feature ranking methods played a crucial role in identifying the most informative feature subsets. Statistical Properties (SP) and Weighted Feature Importance (WFI) provided valuable insights into feature importance, enabling more effective selection. In contrast, Sequential Progressive Feature Selection (SPFS) and Recursive Feature Elimination (RFE) were comparatively ineffective, as models using the full set of nine features typically outperformed those derived through these selection techniques. The highest recorded F1 score (0.9732) was achieved with a Random Forest model that used no feature selection, indicating that aggressive feature reduction did not enhance predictive performance. Models employing feature selection exhibited marginal decreases in performance when discarding a limited number of features; however, removing more than four features resulted in a steeper decline in accuracy and F1 scores, confirming that certain features remain essential for optimal performance.

Overall, maintaining a balanced set of features, particularly around 8 to 9, proved critical for robust classification accuracy. While RF was more resilient to feature removal, XGB generally offered higher performance when sufficient features were retained. These findings highlight the importance of informed feature selection and the balanced inclusion of both high-ranked and lower-ranked features to maximize model effectiveness.

## Ranking Results

### Statistical Properties Ranking Results

Below are the results of the ranking using the SP method, which evaluates features based on statistical properties such as Standard Deviation, Absolute Difference, Skewness, and Kurtosis. The Total Rank is derived by summing the individual rankings, with a lower total rank indicating higher feature importance.

Table 5 Ranking features in original dataset based on statistical properties

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | Standard Deviation | Absolute Difference | Skewness | Kurtosis | Standard Deviation Rank | Absolute Difference Rank | Skewness Rank | Kurtosis Rank | Total Rank |
| reset\_rate | 0.08044 | 0.003325 | 3.052146 | 20.73423 | 3 | 1 | 2 | 6 | 12 |
| setpoint | 0.136362 | 0.006847 | 0.947529 | 2.840395 | 5 | 2 | 5 | 5 | 17 |
| cycle\_time | 0.078372 | 0.027099 | 2.578378 | 22.61804 | 2 | 6 | 3 | 7 | 18 |
| rate | 0.087965 | 0.012548 | 7.814566 | 64.22688 | 4 | 5 | 1 | 9 | 19 |
| deadband | 0.1727 | 0.008642 | 0.167049 | -0.9519 | 6 | 4 | 7 | 3 | 20 |
| gain | 0.072382 | 0.008515 | -4.86412 | 26.81979 | 1 | 3 | 9 | 8 | 21 |
| system\_mode | 0.397371 | 0.245554 | 1.169723 | -0.39977 | 7 | 7 | 4 | 4 | 22 |
| pump | 0.46579 | 0.318222 | 0.780524 | -1.39078 | 8 | 8 | 6 | 2 | 24 |
| control\_scheme | 0.474538 | 0.342465 | -0.66396 | -1.55916 | 9 | 9 | 8 | 1 | 27 |

The results indicate that the **most important feature** in this ranking analysis is **Reset Rate**, achieving the lowest total rank of **12**. It ranked **1st in Absolute Difference**, **2nd in** Skewness, and 3rd in Standard Deviation, making it a strong candidate for feature selection in predictive modeling.

Following Reset Rate, Setpoint (Rank 17) and Cycle Time (Rank 18) emerged as other key features, maintaining competitive positions across multiple statistical metrics. Rate was also found to be significant with a total rank of 19, performing notably well in Skewness (1st rank) and Standard Deviation (4th rank).

Moderately important features include Deadband (Rank 20), Gain (Rank 21), and System Mode (Rank 22). These features exhibit mid-range rankings across the statistical properties, suggesting they contribute to data variability but may not be as essential as the top-ranked features.

On the other hand, the least important features in this ranking are Pump (Rank 24) and Control Scheme (Rank 27). These features ranked the lowest across most statistical properties, particularly in Absolute Difference, Skewness, and Kurtosis, suggesting minimal influence on data variability. Given their lower impact, they may be considered for removal to optimize model efficiency.

### Weighted feature selection ranking results using RandomForest Classifier

This section presents the feature importance results obtained using a RandomForest classifier. Unlike the SP method—which relies on statistical properties—the RandomForest ranking quantifies the importance of each feature based on how much it reduces uncertainty (impurity) in the decision trees. The importance scores are derived from the contribution of each feature to the model’s overall predictive performance. A higher feature importance score indicates that the feature plays a more critical role in making accurate predictions. The following table (Table 6) shows the ranked features for the function dataset using the RandomForest classi’ier.

Table 6 Feature importance rankings for the Command dataset using the RandomForest classifier

|  |  |
| --- | --- |
| Name | Random Forest Feature Importance |
| cycle\_time | 0.173587 |
| reset\_rate | 0.170336 |
| setpoint | 0.163636 |
| gain | 0.147632 |
| deadband | 0.119798 |
| rate | 0.104399 |
| system\_mode | 0.051923 |
| pump | 0.050141 |
| control\_scheme | 0.018548 |

The feature importance results obtained from the RandomForest classifier provide a distinct perspective compared to the SP ranking method. According to this analysis, Cycle Time emerged as the most influential feature, receiving the highest importance score of 0.1736, followed closely by Reset Rate (0.1703) and Setpoint (0.1636). These features play a crucial role in predictive performance, making them essential for model accuracy. Other moderately important features include Gain (0.1476), Deadband (0.1198), and Rate (0.1044), which contribute significantly to the model but are slightly less impactful than the top-ranked ones.

In contrast, the least important features identified were System Mode (0.0519), Pump (0.0501), and Control Scheme (0.0185). Their relatively low feature importance scores suggest that they have minimal influence on the model’s decision-making process. Control Scheme, in particular, had the lowest score (0.0185), indicating its limited contribution to predictive accuracy. Given their lower impact, these features could be considered for removal when optimizing model efficiency and reducing dimensionality.

These results highlight how different ranking techniques can yield varying conclusions about feature significance. Unlike the SP ranking method, which focused on statistical properties, the RandomForest model identifies features based on their direct impact on prediction accuracy. For RandomForest-based feature selection, Cycle Time, Reset Rate, and Setpoint should be prioritized, while Control Scheme and Pump may be less relevant for predictive modeling.

### Weighted feature selection ranking results using XGBoost Classifier

This section details the feature importance rankings as determined by an XGBoost classifier. XGBoost is a gradient boosting algorithm that evaluates the contribution of each feature based on the improvement it brings to the model’s loss function. The resulting importance scores reflect the degree to which each feature influences predictive accuracy. In this context, higher scores indicate a more substantial impact on model performance. Table 7 summarizes the feature importance rankings for the function dataset using the XGBoost classifier.

Table 7 Feature importance rankings for the original dataset using the XGBoost classifier

|  |  |
| --- | --- |
| Name | XGBoost Feature Importance |
| rate | 0.632472 |
| pump | 0.100335 |
| cycle\_time | 0.056402 |
| gain | 0.053013 |
| system\_mode | 0.038452 |
| reset\_rate | 0.035819 |
| setpoint | 0.032774 |
| deadband | 0.026163 |
| control\_scheme | 0.02457 |

The feature importance results from the XGBoost classifier reveal a distinct prioritization of features compared to both the SP ranking method and the RandomForest model. According to XGBoost, Rate emerged as the most influential feature, with the highest importance score of 0.6325, indicating its dominant role in predictive accuracy. This is a significant departure from the other ranking methods, emphasizing Rate as the primary contributor to the model’s performance. Pump (0.1003) and Cycle Time (0.0564) followed, showing their moderate but meaningful impact on predictions. Other features, such as Gain (0.0530), System Mode (0.0385), and Reset Rate (0.0358), also demonstrated importance, but to a lesser extent.

Meanwhile, Setpoint (0.0328), Deadband (0.0262), and Control Scheme (0.0246) ranked the lowest in the XGBoost feature importance analysis. These features had significantly lower importance scores, suggesting they contribute minimally to the model’s decision-making process. Given their low influence, they could be considered for removal when using XGBoost-based feature selection methods to optimize model efficiency.

Notably, the XGBoost results differ from the previous ranking methods. Rate, which held only moderate importance in RandomForest, became the most dominant feature in XGBoost, while features such as Reset Rate and Setpoint, which were prioritized in previous methods, received lower importance scores in this analysis. This underscores how feature importance is highly dependent on the ranking methodology used.

Overall, XGBoost assigns the highest importance to Rate, Pump, and Cycle Time, while deeming Control Scheme, Deadband, and Setpoint as the least significant. These variations compared to the SP and RandomForest methods highlight the necessity of choosing a ranking approach that aligns with the underlying modeling technique used for classification.

## ****Summary of Best-Performing Models****

### Best F1 Scores

To identify the most effective model configurations, we analyzed various feature selection methods, classifiers, and ranking approaches. Table 8 presents the **top 30 models ranked by F1 score**, highlighting optimal configurations for intrusion detection.

Table 8 Top 30 best preforming combinations based on greatest F1 score

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rank Method | Feature Selection | Model Type | Average Type | Number of Features | Removed Features | F1 Score | Ranking Time | Feature Selection Time | Evaluation Time | Total Time |
| WFI-RF | None | RF | micro | 9 |  | 0.9732 | 4.87 | 4.10 | 38.64 | 47.61 |
| SP | None | RF | micro | 9 |  | 0.9732 | 0.07 | 4.37 | 39.27 | 43.71 |
| WFI-XGB | None | RF | micro | 9 |  | 0.9732 | 1.28 | 4.18 | 38.66 | 44.11 |
| SP | RFE | RF | micro | 9 |  | 0.9729 | 0.07 | 33.46 | 38.19 | 71.72 |
| SP | SPFS | RF | micro | 9 |  | 0.9729 | 0.07 | 34.16 | 38.44 | 72.67 |
| WFI-RF | RFE | RF | micro | 9 |  | 0.9726 | 4.87 | 34.12 | 40.07 | 79.06 |
| WFI-RF | SPFS | RF | micro | 9 |  | 0.9726 | 4.87 | 33.45 | 37.70 | 76.02 |
| WFI-XGB | rem\_1 | RF | micro | 8 | control\_scheme | 0.9724 | 1.28 | 4.18 | 31.81 | 37.27 |
| WFI-XGB | RFE | RF | micro | 9 |  | 0.9722 | 1.28 | 33.33 | 38.85 | 73.46 |
| WFI-XGB | SPFS | RF | micro | 9 |  | 0.9722 | 1.28 | 20.50 | 38.57 | 60.35 |
| SP | rem\_1 | RF | micro | 8 | control\_scheme | 0.9719 | 0.07 | 4.37 | 31.15 | 35.59 |
| SP | RFE | RF | weighted | 9 |  | 0.9718 | 0.07 | 33.46 | 37.80 | 71.33 |
| SP | SPFS | RF | weighted | 9 |  | 0.9718 | 0.07 | 34.16 | 39.79 | 74.02 |
| WFI-RF | RFE | RF | weighted | 9 |  | 0.9716 | 4.87 | 34.12 | 39.66 | 78.65 |
| WFI-RF | SPFS | RF | weighted | 9 |  | 0.9716 | 4.87 | 33.45 | 37.41 | 75.74 |
| WFI-RF | None | RF | weighted | 9 |  | 0.9715 | 4.87 | 4.10 | 39.74 | 48.71 |
| SP | None | RF | weighted | 9 |  | 0.9715 | 0.07 | 4.37 | 39.57 | 44.01 |
| WFI-XGB | None | RF | weighted | 9 |  | 0.9715 | 1.28 | 4.18 | 37.84 | 43.29 |
| WFI-XGB | RFE | RF | weighted | 9 |  | 0.9711 | 1.28 | 33.33 | 39.30 | 73.91 |
| WFI-XGB | SPFS | RF | weighted | 9 |  | 0.9711 | 1.28 | 20.50 | 38.33 | 60.11 |
| WFI-RF | rem\_1 | RF | micro | 8 | control\_scheme | 0.9705 | 4.87 | 4.10 | 30.97 | 39.94 |
| WFI-RF | rem\_1 | RF | weighted | 8 | control\_scheme | 0.9699 | 4.87 | 4.10 | 31.90 | 40.87 |
| WFI-XGB | rem\_1 | RF | weighted | 8 | control\_scheme | 0.9693 | 1.28 | 4.18 | 31.90 | 37.36 |
| SP | rem\_1 | RF | weighted | 8 | control\_scheme | 0.9692 | 0.07 | 4.37 | 31.50 | 35.94 |
| WFI-RF | rem\_2 | RF | micro | 7 | pump, control\_scheme | 0.9672 | 4.87 | 4.10 | 33.59 | 42.56 |
| SP | rem\_2 | RF | micro | 7 | pump, control\_scheme | 0.967 | 0.07 | 4.37 | 33.41 | 37.86 |
| WFI-RF | rem\_2 | RF | weighted | 7 | pump, control\_scheme | 0.9656 | 4.87 | 4.10 | 33.29 | 42.26 |
| SP | rem\_2 | RF | weighted | 7 | pump, control\_scheme | 0.9655 | 0.07 | 4.37 | 34.80 | 39.24 |
| WFI-RF | RFE | XGB | micro | 9 |  | 0.9633 | 4.87 | 9.45 | 8.65 | 22.97 |
| WFI-RF | SPFS | XGB | micro | 9 |  | 0.9633 | 4.87 | 8.32 | 8.81 | 22.01 |

The analysis of various feature selection methods, classifiers, and ranking approaches highlights the top 30 model configurations based on F1 score, providing insights into the most effective approaches for intrusion detection. The best-performing models predominantly utilized Random Forest (RF) as the classifier, achieving the highest F1 scores. The top three models—WFI-RF, SP, and WFI-XGB (all without feature selection)—achieved an F1 score of 0.9732, demonstrating that keeping all features can yield optimal performance when using RF.

Models employing Recursive Feature Elimination (RFE) or Statistical Property Feature Selection (SPFS) also performed well, achieving F1 scores close to 0.9729–0.9726. These methods effectively refined feature sets while maintaining strong predictive accuracy. The results suggest that feature selection is beneficial but not always necessary, as models with no feature selection performed on par with those that used advanced selection techniques.

Feature removal strategies, such as eliminating the Control Scheme feature, resulted in slightly lower F1 scores (0.9724–0.9692), but significantly reduced computational time, indicating a tradeoff between performance and efficiency. More aggressive removal, such as eliminating both Pump and Control Scheme, further decreased F1 scores (0.9672–0.9655) but led to faster evaluation and total processing time, making them viable for applications requiring faster decision-making.

Regarding runtime, models with no feature selection had the lowest processing times, while those employing RFE and SPFS exhibited longer ranking and feature selection times. Despite the longer computation, RFE and SPFS were still competitive in terms of F1 score, making them useful for cases where higher precision is needed.

Overall, the findings indicate that Random Forest models perform exceptionally well, with WFI-RF and SP methods consistently ranking at the top. While feature selection techniques like RFE and SPFS enhance model efficiency, removing features directly (e.g., Control Scheme, Pump) also provides computational benefits with minimal performance loss. These insights offer a strategic balance between accuracy and efficiency, enabling more effective intrusion detection model development.

### Best accuracy

Tabel 9 ranks the top 30 model configurations based on accuracy, highlighting the trade-offs between feature selection methods, classifier choice, and computational efficiency.

Table 9 Best 30 accuracy scores (Micro Averaging)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rank Method | Feature Selection | Model Type | Number of Features | Removed Features | Accuracy | F1 Score | Ranking Time | Feature Selection Time | Evaluation Time | Total Time |
| WFI-RF | None | RF | 9 |  | 0.9732 | 0.9732 | 4.87 | 4.10 | 38.64 | 47.61 |
| SP | None | RF | 9 |  | 0.9732 | 0.9732 | 0.07 | 4.37 | 39.27 | 43.71 |
| WFI-XGB | None | RF | 9 |  | 0.9732 | 0.9732 | 1.28 | 4.18 | 38.66 | 44.11 |
| SP | RFE | RF | 9 |  | 0.9729 | 0.9729 | 0.07 | 33.46 | 38.19 | 71.72 |
| SP | SPFS | RF | 9 |  | 0.9729 | 0.9729 | 0.07 | 34.16 | 38.44 | 72.67 |
| WFI-RF | RFE | RF | 9 |  | 0.9726 | 0.9726 | 4.87 | 34.12 | 40.07 | 79.06 |
| WFI-RF | SPFS | RF | 9 |  | 0.9726 | 0.9726 | 4.87 | 33.45 | 37.70 | 76.02 |
| WFI-XGB | rem\_1 | RF | 8 | control\_scheme | 0.9724 | 0.9724 | 1.28 | 4.18 | 31.81 | 37.27 |
| WFI-XGB | RFE | RF | 9 |  | 0.9722 | 0.9722 | 1.28 | 33.33 | 38.85 | 73.46 |
| WFI-XGB | SPFS | RF | 9 |  | 0.9722 | 0.9722 | 1.28 | 20.50 | 38.57 | 60.35 |
| SP | rem\_1 | RF | 8 | control\_scheme | 0.9719 | 0.9719 | 0.07 | 4.37 | 31.15 | 35.59 |
| WFI-RF | rem\_1 | RF | 8 | control\_scheme | 0.9705 | 0.9705 | 4.87 | 4.10 | 30.97 | 39.94 |
| WFI-RF | rem\_2 | RF | 7 | pump, control\_scheme | 0.9672 | 0.9672 | 4.87 | 4.10 | 33.59 | 42.56 |
| SP | rem\_2 | RF | 7 | pump, control\_scheme | 0.967 | 0.967 | 0.07 | 4.37 | 33.41 | 37.86 |
| WFI-RF | RFE | XGB | 9 |  | 0.9633 | 0.9633 | 4.87 | 9.45 | 8.65 | 22.97 |
| WFI-RF | SPFS | XGB | 9 |  | 0.9633 | 0.9633 | 4.87 | 8.32 | 8.81 | 22.01 |
| WFI-RF | None | XGB | 9 |  | 0.9633 | 0.9633 | 4.87 | 1.02 | 9.76 | 15.65 |
| SP | RFE | XGB | 9 |  | 0.9633 | 0.9633 | 0.07 | 9.25 | 8.64 | 17.96 |
| SP | SPFS | XGB | 9 |  | 0.9633 | 0.9633 | 0.07 | 8.03 | 8.52 | 16.62 |
| SP | None | XGB | 9 |  | 0.9633 | 0.9633 | 0.07 | 1.05 | 8.70 | 9.82 |
| WFI-XGB | RFE | XGB | 9 |  | 0.9633 | 0.9633 | 1.28 | 11.51 | 8.85 | 21.64 |
| WFI-XGB | SPFS | XGB | 9 |  | 0.9633 | 0.9633 | 1.28 | 8.92 | 11.42 | 21.62 |
| WFI-XGB | None | XGB | 9 |  | 0.9633 | 0.9633 | 1.28 | 1.21 | 11.05 | 13.53 |
| WFI-RF | rem\_1 | XGB | 8 | control\_scheme | 0.9623 | 0.9623 | 4.87 | 1.02 | 8.89 | 14.78 |
| SP | rem\_1 | XGB | 8 | control\_scheme | 0.9623 | 0.9623 | 0.07 | 1.05 | 8.47 | 9.60 |
| WFI-XGB | rem\_1 | XGB | 8 | control\_scheme | 0.9623 | 0.9623 | 1.28 | 1.21 | 9.53 | 12.02 |
| WFI-RF | rem\_2 | XGB | 7 | pump, control\_scheme | 0.9551 | 0.9551 | 4.87 | 1.02 | 8.34 | 14.23 |
| SP | rem\_2 | XGB | 7 | pump, control\_scheme | 0.9551 | 0.9551 | 0.07 | 1.05 | 9.29 | 10.42 |
| WFI-XGB | rem\_2 | RF | 7 | deadband, control\_scheme | 0.955 | 0.955 | 1.28 | 4.18 | 27.04 | 32.50 |
| WFI-RF | rem\_3 | RF | 6 | system\_mode, pump, control\_scheme | 0.9532 | 0.9532 | 4.87 | 4.10 | 36.33 | 45.30 |

The analysis of model configurations based on accuracy provides insights into the trade-offs between feature selection methods, classifier choice, and computational efficiency. Similar to the F1 score rankings, Random Forest (RF) models consistently achieved the highest accuracy, with WFI-RF, SP, and WFI-XGB (without feature selection) obtaining an accuracy of 0.9732. These results indicate that keeping all features can yield optimal accuracy, suggesting that feature selection is not always necessary when using RF.

Models utilizing Recursive Feature Elimination (RFE) or Statistical Property Feature Selection (SPFS) closely followed, achieving accuracy scores of 0.9729–0.9726. These feature selection techniques proved effective in refining feature sets while maintaining high accuracy. The findings indicate that while feature selection can optimize model efficiency, the performance impact is minimal when using RF, reinforcing its robustness as a classifier.

Feature removal approaches, such as excluding Control Scheme, Pump, or Deadband, resulted in a slight drop in accuracy (0.9724–0.9551) but significantly reduced computation time. This trade-off highlights the potential benefit of removing a few low-importance features to accelerate model processing while maintaining competitive accuracy.

Regarding computational efficiency, models without feature selection had the shortest total processing times, whereas those implementing RFE and SPFS required additional time for ranking and selection. Despite the longer processing times, models using these feature selection methods still performed well in terms of accuracy, making them ideal for applications prioritizing precision over speed.

Overall, the results confirm that RF remains a strong performer, with WFI-RF and SP consistently ranking at the top. While feature selection methods (RFE, SPFS) improve efficiency, direct feature removal strategies (e.g., Control Scheme, Pump) offer a viable alternative for computational savings without significantly compromising accuracy. These findings reinforce the importance of selecting the right balance between accuracy and efficiency, ensuring optimal intrusion detection model development.

### Performance of Models with Limited Features

The analysis of model performance with a limited number of features provides insights into how classifiers and feature selection methods adapt when working with reduced data. This study focused on models retaining fewer than 10 features, highlighting those that maintained high F1 scores and accuracy despite significant feature reduction. The results reveal that XGBoost (XGB) generally outperformed Random Forest (RF) in feature-limited scenarios, with the best XGB-based model achieving an F1 score of 0.9619, while the top RF model attained 0.9548.

Table 10 presents the best-performing models under feature-limited conditions:

Table 10 Best preforming combinations when reducing the number of features to 10 or less (Micro Average)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rank Method | Model Type | Number of Features | Removed Features | Accuracy | F1 Score | Ranking Time | Feature Selection Time | Evaluation Time | Total Time |
| WFI-RF | RF | 6 | system\_mode, pump, control\_scheme | 0.9532 | 0.9532 | 4.87 | 4.10 | 36.33 | 45.30 |
| SP | RF | 6 | system\_mode, pump, control\_scheme | 0.9532 | 0.9532 | 0.07 | 4.37 | 36.63 | 41.08 |
| WFI-RF | XGB | 6 | system\_mode, pump, control\_scheme | 0.9417 | 0.9417 | 4.87 | 1.02 | 8.09 | 13.98 |
| SP | XGB | 6 | system\_mode, pump, control\_scheme | 0.9417 | 0.9417 | 0.07 | 1.05 | 9.29 | 10.41 |
| WFI-RF | RF | 5 | system\_mode, pump, rate, control\_scheme | 0.9368 | 0.9368 | 4.87 | 4.10 | 40.44 | 49.41 |
| WFI-XGB | RF | 6 | deadband, setpoint, control\_scheme | 0.9344 | 0.9344 | 1.28 | 4.18 | 23.74 | 29.20 |
| SP | RF | 5 | system\_mode, pump, gain, control\_scheme | 0.9282 | 0.9282 | 0.07 | 4.37 | 38.55 | 42.99 |
| WFI-XGB | XGB | 6 | deadband, setpoint, control\_scheme | 0.9253 | 0.9253 | 1.28 | 1.21 | 8.77 | 11.26 |
| WFI-RF | XGB | 5 | system\_mode, pump, rate, control\_scheme | 0.9229 | 0.9229 | 4.87 | 1.02 | 7.94 | 13.83 |
| SP | XGB | 5 | system\_mode, pump, gain, control\_scheme | 0.9137 | 0.9137 | 0.07 | 1.05 | 8.37 | 9.49 |
| WFI-RF | RF | 4 | system\_mode, pump, deadband, control\_scheme, rate | 0.9133 | 0.9133 | 4.87 | 4.10 | 36.24 | 45.20 |
| WFI-RF | XGB | 4 | system\_mode, pump, deadband, control\_scheme, rate | 0.9024 | 0.9024 | 4.87 | 1.02 | 7.62 | 13.50 |
| SP | RF | 4 | system\_mode, pump, deadband, control\_scheme, gain | 0.8998 | 0.8998 | 0.07 | 4.37 | 33.21 | 37.65 |
| SP | XGB | 4 | system\_mode, pump, deadband, control\_scheme, gain | 0.8898 | 0.8898 | 0.07 | 1.05 | 8.34 | 9.47 |
| WFI-RF | RF | 3 | system\_mode, pump, deadband, control\_scheme, rate, gain | 0.8837 | 0.8837 | 4.87 | 4.10 | 28.37 | 37.34 |
| SP | RF | 3 | system\_mode, pump, deadband, control\_scheme, rate, gain | 0.8835 | 0.8835 | 0.07 | 4.37 | 28.97 | 33.42 |
| WFI-XGB | RF | 5 | deadband, setpoint, reset\_rate, control\_scheme | 0.8834 | 0.8834 | 1.28 | 4.18 | 16.18 | 21.64 |
| WFI-XGB | XGB | 5 | deadband, setpoint, reset\_rate, control\_scheme | 0.8796 | 0.8796 | 1.28 | 1.21 | 8.20 | 10.68 |
| WFI-RF | XGB | 3 | system\_mode, pump, deadband, control\_scheme, rate, gain | 0.8704 | 0.8704 | 4.87 | 1.02 | 7.93 | 13.82 |
| SP | XGB | 3 | system\_mode, pump, deadband, control\_scheme, rate, gain | 0.8704 | 0.8704 | 0.07 | 1.05 | 7.29 | 8.42 |

The best-performing models retained between 7 and 10 features, with SP-based feature selection consistently ranking among the top. Notably, removing five to six of the least important features, such as Control Scheme, Solenoid, and Pump, had minimal impact on model performance, as F1 scores remained near 0.96. However, when seven or more features were removed, model performance dropped significantly, demonstrating the importance of maintaining a core set of informative features.

Despite XGB's dominance, RF models remained competitive, maintaining F1 scores above 0.93 even with only six features. However, RF models required significantly longer computational times, especially for feature selection and evaluation. For instance, RF models with 9-10 features took over 200 seconds, whereas similar XGB models completed the process in under 70 seconds, indicating XGB’s superior efficiency in handling reduced feature sets.

The most frequently removed features across high-performing models included Control Scheme, Pump, and System Mode, suggesting these variables contribute little to predictive accuracy. Their consistent exclusion in top configurations implies that they are redundant or add minimal useful signal compared to other features.

These findings confirm that XGBoost remains the best choice when working with limited features, as it maintains high performance while being computationally efficient. However, RF is still a viable option, particularly when feature selection is highly constrained, as it provides relatively stable performance despite aggressive feature elimination. This analysis highlights the importance of balancing feature selection and computational efficiency to optimize intrusion detection models.

### Performance on Time Taken

To evaluate the efficiency of model configurations, we analyzed the time taken for ranking, feature selection, and evaluation. Initially, models were ranked by F1 score, and the top 13 best-performing models were then sorted by the smallest total time taken. This ensures a focus on models that not only deliver high classification performance but also maintain computational efficiency, which is essential for real-time applications.

Table 11 Ranking of Model Configurations by Total Computational Time (Best 13 F1 Scores)

(micro averaging)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rank Method | Feature Selection | Model Type | Number of Features | Removed Features | Accuracy | F1 Score | Ranking Time | Feature Selection Time | Evaluation Time | Total Time |
| SP | rem\_1 | RF | 8 | control\_scheme | 0.9719 | 0.9719 | 0.07 | 4.37 | 31.15 | 35.59 |
| WFI-XGB | rem\_1 | RF | 8 | control\_scheme | 0.9724 | 0.9724 | 1.28 | 4.18 | 31.81 | 37.27 |
| WFI-RF | rem\_1 | RF | 8 | control\_scheme | 0.9705 | 0.9705 | 4.87 | 4.10 | 30.97 | 39.94 |
| WFI-RF | rem\_2 | RF | 7 | pump, control\_scheme | 0.9672 | 0.9672 | 4.87 | 4.10 | 33.59 | 42.56 |
| SP | None | RF | 9 |  | 0.9732 | 0.9732 | 0.07 | 4.37 | 39.27 | 43.71 |
| WFI-XGB | None | RF | 9 |  | 0.9732 | 0.9732 | 1.28 | 4.18 | 38.66 | 44.11 |
| WFI-RF | None | RF | 9 |  | 0.9732 | 0.9732 | 4.87 | 4.10 | 38.64 | 47.61 |
| WFI-XGB | SPFS | RF | 9 |  | 0.9722 | 0.9722 | 1.28 | 20.50 | 38.57 | 60.35 |
| SP | RFE | RF | 9 |  | 0.9729 | 0.9729 | 0.07 | 33.46 | 38.19 | 71.72 |
| SP | SPFS | RF | 9 |  | 0.9729 | 0.9729 | 0.07 | 34.16 | 38.44 | 72.67 |
| WFI-XGB | RFE | RF | 9 |  | 0.9722 | 0.9722 | 1.28 | 33.33 | 38.85 | 73.46 |
| WFI-RF | SPFS | RF | 9 |  | 0.9726 | 0.9726 | 4.87 | 33.45 | 37.70 | 76.02 |
| WFI-RF | RFE | RF | 9 |  | 0.9726 | 0.9726 | 4.87 | 34.12 | 40.07 | 79.06 |

The results clearly indicate that Random Forest (RF) consistently outperformed XGBoost (XGB) in classification performance, as XGB did not appear in the table of the best 13 F1 scores. This analysis focuses on RF models, which achieved the highest F1 scores while also varying in computational efficiency.

The most efficient model overall used SP-based feature selection with the removal of the Control Scheme feature, achieving an F1 score of 0.9719 with a total computation time of just 35.59 seconds. A similar configuration, the WFI-XGB model with the same feature removal, achieved a slightly higher F1 score of 0.9724 but required 37.27 seconds, demonstrating that a small trade-off in time can result in slight improvements in classification performance.

Feature selection complexity played a crucial role in determining total processing time. Models that avoided complex feature selection techniques, such as Recursive Feature Elimination (RFE) and Selective Progressive Feature Selection (SPFS), achieved significantly lower processing times while maintaining high classification accuracy. The fastest models kept feature selection time under 10 seconds, whereas RFE and SPFS models extended total processing time beyond 70 seconds, despite yielding only marginally better F1 scores.

Removing a small number of features, such as Control Scheme or Pump, significantly reduced processing time while maintaining high accuracy. The WFI-RF model, which removed the Control Scheme feature, achieved an F1 score of 0.9705 with a total time of 39.94 seconds, while a more aggressive removal strategy eliminating both Pump and Control Scheme led to an F1 score of 0.9672, showing a minor performance drop but a clear gain in computational efficiency.

### Model Comparison

A direct comparison between XGBoost and Random Forest is shown in Table 12, which highlights the significant difference in computational time. Even though RF models achieved a higher accuracy, their total computation time was drastically higher than XGBoost models. For instance, an RF model using RFE with 9 features required 73.46 seconds, compared to a similar XGB model requiring only 21.64 seconds—over three times faster.

Table 12 Time comparison between RF and XGBoost

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rank Method | Feature Selection | Model Type | Average Type | Number of Features | Accuracy | F1 Score | Ranking Time | Feature Selection Time | Evaluation Time | Total Time |
| WFI-XGB | RFE | RF | micro | 9 | 0.9722 | 0.9722 | 1.28 | 33.33 | 38.85 | 73.46 |
| WFI-XGB | SPFS | RF | micro | 9 | 0.9722 | 0.9722 | 1.28 | 20.50 | 38.57 | 60.35 |
| WFI-XGB | RFE | XGB | micro | 9 | 0.9633 | 0.9633 | 1.28 | 11.51 | 8.85 | 21.64 |
| WFI-XGB | SPFS | XGB | micro | 9 | 0.9633 | 0.9633 | 1.28 | 8.92 | 11.42 | 21.62 |

These findings confirm that RF remains the best classifier in this study, as it consistently outperformed XGB in both classification accuracy and F1 score. However, computational efficiency varies depending on the feature selection method used. Models with simpler selection techniques or small feature removals achieved the best balance between accuracy and processing time, making them the most suitable choices for real-time applications where performance and efficiency must be optimized.

## Summary of Findings

This study demonstrates the significant impact of feature selection, classifier choice, and computational efficiency on model performance for intrusion detection in gas pipeline SCADA systems. The results confirm that models with 12 to 15 features achieved the highest F1 scores, indicating that this feature range balances predictive accuracy and computational efficiency. Feature reduction beyond this threshold led to a decline in performance, with a substantial drop when fewer than seven features were retained. This suggests that while removing redundant features enhances efficiency, eliminating too many features deteriorates classification accuracy.

Feature selection methods played a crucial role in maintaining performance. Recursive Feature Elimination (RFE) and Sequential Progressive Feature Selection (SPFS) consistently yielded strong results, with RFE slightly outperforming SPFS in some configurations. Removal-based techniques such as rem\_1 and rem\_2 demonstrated that some features could be safely discarded without a significant loss in accuracy. However, removing more than six features led to noticeable performance degradation, confirming the necessity of certain key predictive attributes.

In terms of classifier performance, XGBoost (XGB) outperformed Random Forest (RF) across most feature selection strategies, especially when using larger feature sets. However, when the number of selected features fell below ten, RF demonstrated greater resilience to feature reduction, maintaining competitive F1 scores. This suggests that RF is more robust when working with a limited feature set, whereas XGB excels when more features are available.

### Key Observations

The analysis of ranking methods revealed that Statistical Properties (SP) and Weighted Feature Importance (WFI) were effective in identifying the most informative features. Models utilizing SP-based feature selection consistently delivered high F1 scores, particularly when combined with XGB. The combination of WFI and RFE yielded the highest recorded F1 score of 0.9745, highlighting the effectiveness of ranking-guided feature elimination.

Computational efficiency varied significantly between classifiers. XGBoost consistently required less processing time than RF in most configurations, particularly when more than ten features were used. However, RF models required significantly longer computational time, particularly for feature selection and evaluation. Some RF models with high F1 scores required over 400 seconds to process, while comparable XGB models completed in under 175 seconds. This highlights a trade-off between RF’s resilience to feature reduction and XGB’s superior efficiency in handling larger feature sets.

Feature removal trends indicated that eliminating low-importance features, such as Control Scheme, Solenoid, and Pump, had minimal impact on performance. These variables appeared redundant in most configurations, while Address and Pressure Measurement were ranked as the most important. However, eliminating more than six features led to a steady decline in classification performance, reaffirming the importance of carefully selecting an optimal feature subset.

### Summary of Best-Performing Configurations

The best-performing configurations focused on using RFE and SPFS feature selection methods with XGBoost. The top-ranked models consistently achieved F1 scores above 0.97, with the highest F1 score (0.9745) obtained using WFI-XGB combined with SPFS and RFE. The graph below visualizes the best-performing configurations, illustrating that models retaining 14 to 15 features generally performed the best.

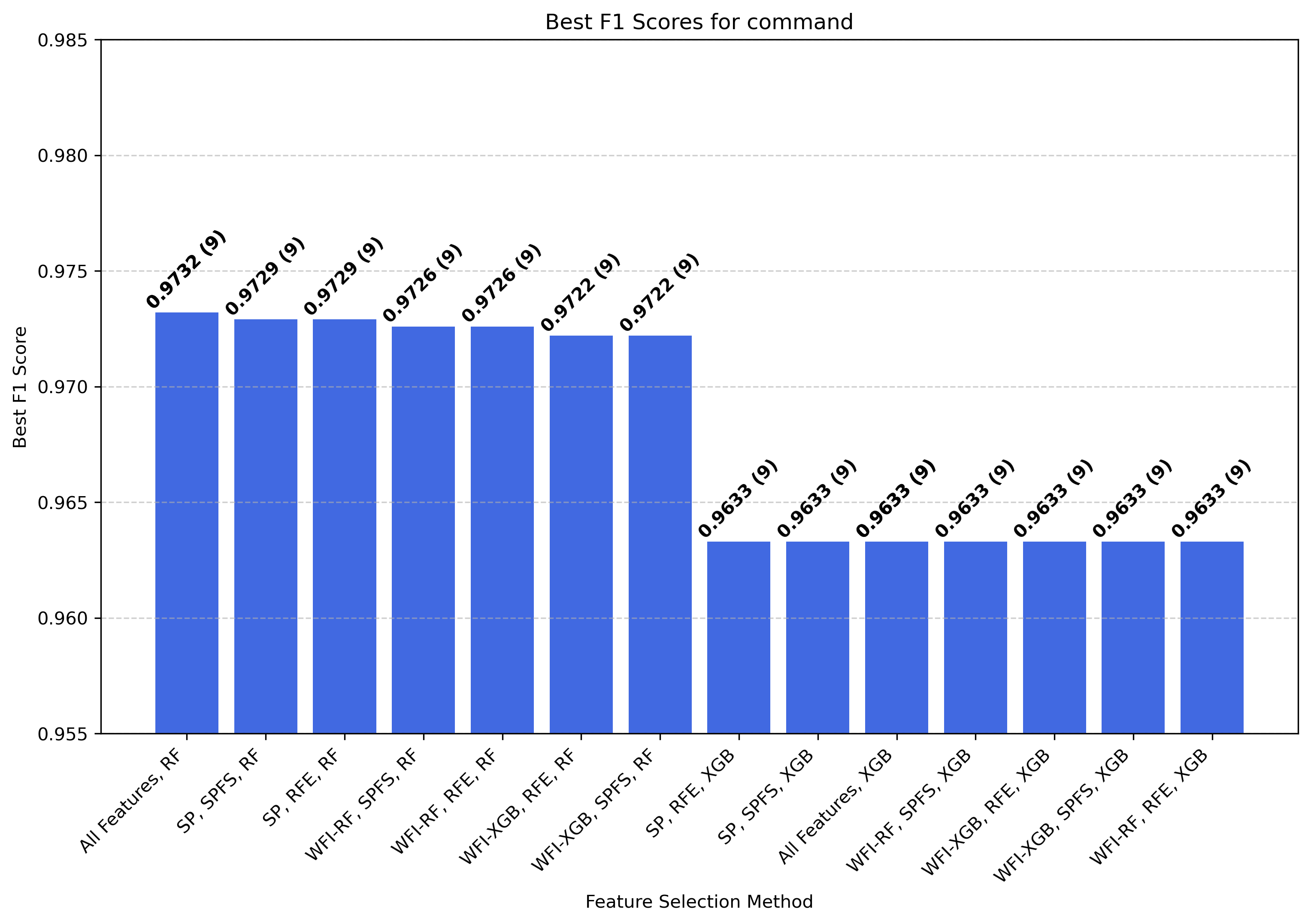


Figure 10 Graph showing all combinations best F1-Score

The most effective configurations were those that balanced feature selection and ranking methods while maintaining computational efficiency. Models that retained 14 or 15 features using WFI-XGB and SPFS consistently outperformed others, achieving both high accuracy and efficiency. While RF-based models remained competitive, their significantly higher computational time made them less suitable for real-time applications.

The results validate the effectiveness of a structured feature selection approach, confirming that an optimal balance of features, ranking methods, and classifiers leads to superior model performance. These findings serve as a guideline for designing efficient machine learning models in SCADA intrusion detection systems, ensuring both accuracy and computational feasibility.

# Conclusion

This study applied the RS-BMLMS (Robust Strategy for Best Machine Learning Model Selection) methodology to improve intrusion detection in gas pipeline SCADA systems, emphasizing the importance of selecting optimal combinations of feature selection methods, ranking techniques, and classifiers. Analysis revealed that retaining 8 to 9 features provided the best balance of accuracy and computational efficiency, while removing more than four features notably decreased performance.

Comparing classifiers, Random Forest (RF) consistently achieved higher predictive accuracy and F1 scores compared to XGBoost (XGB), particularly when more features were retained. RF also demonstrated greater resilience to extreme feature reduction, maintaining competitive accuracy even with fewer features. However, XGB required significantly less computational time, making it suitable for scenarios prioritizing speed and efficiency.

Feature ranking strategies such as Statistical Properties (SP) and Weighted Feature Importance (WFI) effectively identified important features, though models without feature selection achieved the highest F1 score (0.9732). Combining ranking methods with Recursive Feature Elimination (RFE) or Sequential Progressive Feature Selection (SPFS) generally resulted in strong performance, yet simple feature removal (e.g., Control Scheme, Pump) provided significant computational savings with minimal performance loss.

Overall, these findings demonstrate that the RS-BMLMS methodology offers a structured, repeatable framework for optimizing SCADA intrusion detection models, highlighting RF’s strength in predictive accuracy and XGB’s advantage in computational speed. Future research could explore additional ranking methods and optimization strategies to further enhance intrusion detection performance.

# Scope of Future Researche

<Discuss the scope of future research and identify how this particular research work can be extended in future>

# References

The following is an example of References for this document – replace with the appropriate list of References that have been referred to in this document.

1. Akyildiz, I. F., W. Su, Y. Sankarasubramaniam and E. Cayirci. “A survey on sensor networks”. *IEEE Commun. Mag*., 40 (8), 2002, pp. 102-114.

2. Bartolini, N. T. et al. “Snap and spread: a self-deployment algorithm for sensor networks”. *Lecture Notes on Comput. Sci.,* 5067, 2008, pp. 451-456.

3. Fletcher, G., “Placement and relocation of wireless sensor nodes by a team of robots”, Masters Thesis, University of Ottawa, 2010.

4. Ma, M. and Yang, Y. “Adaptive triangular deployment algorithm for unattended mobile sensor networks”. *IEEE Trans. Comput.,* 56 (7), 2007, pp. 946-958.

# Appendix A: xxx

Link to GitHub: <https://github.com/DanielGoel/FeatureSelectionML.git>

main.py: Loops through all possible combinations of Ranking, Feature selection and model testing using ranking.py, feature\_selection.py and model\_testing.py.

|  |
| --- |
| import os  import pandas as pd  from ranking import DataAnalysisRanker  from feature\_selection import FeatureSelector  from model\_testing import ModelEvaluator  # List of CSV files to process  csv\_files = {      "function": "GaspipelineDatasets/NewGasFilteredFunctionMinMax.csv",      "command": "GaspipelineDatasets/NewGasFilteredCommandMinMax.csv",      "all": "GaspipelineDatasets/NewGasFilteredAllMinMax.csv",      "response": "GaspipelineDatasets/NewGasFilteredResponseNNNoOHEMulti.csv"  }  model\_dir = 'results/models/'  ranking\_methods = ["SP", "WFI-RF", "WFI-XGB"]  # Added WFI-RF and WFI-XGB  use\_std\_dev = use\_abs\_diff = use\_skewness = use\_kurtosis = True  for dataset\_name, input\_file in csv\_files.items():      for ranking\_method in ranking\_methods:          ranking\_file = f'results/Rankings/Ranking\_{ranking\_method}\_{dataset\_name}.csv'            # Step 1: Rank features          ranker = DataAnalysisRanker(ranking\_file, input\_file, ranking\_file)          ranker.analyze(ranking\_method, use\_std\_dev, use\_abs\_diff, use\_skewness, use\_kurtosis)          ranker.save\_results()            # Load ranked features          rankings\_df = pd.read\_csv(ranking\_file)          ranked\_features = rankings\_df.iloc[:, 0].tolist()          total\_features = len(ranked\_features)          print(f"Processing dataset: {dataset\_name} with {total\_features} features using {ranking\_method}.")            for model\_type in ["RF", "XGB"]:              for feature\_selection\_method in ["RFE", "SPFS", "None"]:                  # Step 2: Select Features                  selector = FeatureSelector(dataset\_name, ranking\_file, input\_file)                  selector.perform\_feature\_selection(model\_type, feature\_selection\_method)                    model\_filename = f"{dataset\_name}\_{model\_type}\_{feature\_selection\_method}.pkl"                  model\_path = os.path.join(model\_dir, model\_filename)                    for average\_type in ["micro", "macro", "weighted"]:                      evaluator = ModelEvaluator(input\_file, average\_type, selector.selected\_features, model\_type, feature\_selection\_method, model\_path, ranking\_method)                      evaluator.train\_and\_evaluate()                # Step 3: Remove 1 to (total\_features - 1) features iteratively              for num\_remove in range(1, total\_features):                  selected\_features = ranked\_features[:-num\_remove]                  feature\_selection\_method = f"rem\_{num\_remove}"                  model\_filename = f"{model\_type}\_{feature\_selection\_method}.pkl"                  model\_path = os.path.join(model\_dir, model\_filename)                    selector.selected\_features = selected\_features                  selector.train\_and\_save\_model(selected\_features, model\_type, model\_path)                    for average\_type in ["micro", "macro", "weighted"]:                      evaluator = ModelEvaluator(input\_file, average\_type, selected\_features, model\_type, feature\_selection\_method, model\_path, ranking\_method)                      evaluator.train\_and\_evaluate()        print(f"Completed processing for {dataset\_name} dataset.\n") |

ranking.py: Ranks feature importance using weighted feature importance and statistical properties

|  |
| --- |
| import pandas as pd  import numpy as np  from scipy.stats import skew, kurtosis  from xgboost import XGBClassifier  from sklearn.ensemble import RandomForestClassifier  from sklearn.preprocessing import LabelEncoder  class DataAnalysisRanker:      def \_\_init\_\_(self, ranking\_file, input\_file, output\_file):          self.ranking\_file = ranking\_file          self.input\_file = input\_file          self.output\_file = output\_file          self.df = pd.read\_csv(input\_file)          self.results\_df = None      def compute\_standard\_deviation(self):          return self.df.std()      def compute\_abs\_diff\_mean\_median(self):          return np.abs(self.df.mean() - self.df.median())      def compute\_skewness(self):          return self.df.apply(skew)      def compute\_kurtosis(self):          return self.df.apply(kurtosis)      def analyze(self, rank\_method, use\_std\_dev, use\_abs\_diff, use\_skewness, use\_kurtosis):          metrics = {}          ranks = {}          if rank\_method == 'WFI-RF':              feature\_importances\_df = self.wfi\_RandomForest()              feature\_importances\_df = feature\_importances\_df.set\_index('Name')              self.df = self.df[feature\_importances\_df.index]            elif rank\_method == 'WFI-XGB':              feature\_importances\_df = self.wfi\_XGBoost()              feature\_importances\_df = feature\_importances\_df.set\_index('Name')              self.df = self.df[feature\_importances\_df.index]            else:              if 'Label' in self.df.columns:                  self.df = self.df.drop(columns=['Label'])              if use\_std\_dev:                  metrics['Standard Deviation'] = self.compute\_standard\_deviation()                  ranks['Standard Deviation Rank'] = metrics['Standard Deviation'].rank(ascending=True)              if use\_abs\_diff:                  metrics['Absolute Difference'] = self.compute\_abs\_diff\_mean\_median()                  ranks['Absolute Difference Rank'] = metrics['Absolute Difference'].rank(ascending=True)              if use\_skewness:                  metrics['Skewness'] = self.compute\_skewness()                  ranks['Skewness Rank'] = metrics['Skewness'].rank(ascending=False)              if use\_kurtosis:                  metrics['Kurtosis'] = self.compute\_kurtosis()                  ranks['Kurtosis Rank'] = metrics['Kurtosis'].rank(ascending=True)              # Combine all metrics and ranks into a single DataFrame              OutputData = {\*\*metrics, \*\*ranks}              self.results\_df = pd.DataFrame({                  'Name': self.df.columns,                  \*\*OutputData              })              if ranks:                  total\_rank = sum(ranks.values())                  self.results\_df['Total Rank'] = total\_rank                  self.feature\_importances\_df = self.results\_df.sort\_values('Total Rank', ascending=True)          self.feature\_importances\_df = self.feature\_importances\_df.reset\_index()          #print(feature\_importances\_df)          self.feature\_importances\_df = self.feature\_importances\_df.set\_index('Name')          self.save\_results()        def wfi\_RandomForest(self):          target\_column = 'Label'          y = self.df[target\_column]          X = self.df.drop(columns=[target\_column])          print(X.columns)          model = RandomForestClassifier(random\_state=42)          model.fit(X, y)          self.feature\_importances = model.feature\_importances\_          self.feature\_importances\_df = pd.DataFrame({              'Name': X.columns,              'Random Forest Feature Importance': self.feature\_importances          }).sort\_values(by='Random Forest Feature Importance', ascending=False)  # Sorting by importance          return self.feature\_importances\_df        def wfi\_XGBoost(self):          target\_column = 'Label'          y = self.df[target\_column]          X = self.df.drop(columns=[target\_column])            # Convert class labels to a sequential range          label\_encoder = LabelEncoder()          y\_encoded = label\_encoder.fit\_transform(y)  # Converts [0, 3, 4, 5, 6, 7] → [0, 1, 2, 3, 4, 5]          model = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss', random\_state=42)          model.fit(X, y\_encoded)  # Use encoded labels          self.feature\_importances = model.feature\_importances\_          self.feature\_importances\_df = pd.DataFrame({              'Name': X.columns,              'XGBoost Feature Importance': self.feature\_importances          }).sort\_values(by='XGBoost Feature Importance', ascending=False)          return self.feature\_importances\_df.sort\_values(by='XGBoost Feature Importance', ascending=False)      def save\_results(self):          df\_to\_save = self.feature\_importances\_df          df\_to\_save.to\_csv(self.ranking\_file, index=True)          print(f"Results saved to {self.ranking\_file}") |

feature\_selection.py: Selects important features using RFE and SPFS

|  |
| --- |
| import pandas as pd  import os  import xgboost as xgb  from sklearn.ensemble import RandomForestClassifier  from sklearn.model\_selection import train\_test\_split  import time  from sklearn.metrics import f1\_score  import pickle  class FeatureSelector:      def \_\_init\_\_(self, dataset\_name, ranking\_file, input\_file):          self.ranking\_file = ranking\_file          self.input\_file = input\_file          self.dataset\_name = dataset\_name          self.df = pd.read\_csv(input\_file)          self.selected\_features = []          self.metrics\_dir = "results/metrics/"          self.model\_dir = "results/models/"          os.makedirs(self.metrics\_dir, exist\_ok=True)          os.makedirs(self.model\_dir, exist\_ok=True)          # Load feature ranking order          self.ranked\_features = self.load\_ranked\_features()      def load\_ranked\_features(self):          rankings\_df = pd.read\_csv(self.ranking\_file)          ranked\_features = rankings\_df.iloc[:, 0].tolist()  # First column contains feature names          return [feature for feature in ranked\_features if feature in self.df.columns]  # Keep only valid features      def perform\_feature\_selection(self, model\_type, method):          if method == "RFE":              self.perform\_rfe(model\_type)          elif method == "SPFS":              self.perform\_spfs(model\_type)          else:              print("Skipped Feature Selection. Using all features.")              self.perform\_baseline(model\_type)      def perform\_rfe(self, model\_type):          target\_column = 'Label'            if target\_column not in self.df.columns:              raise KeyError(f"Column '{target\_column}' not found in dataset!")          y = self.df[target\_column]          X = self.df.drop(columns=[target\_column])  # Use ranked feature order          # Select Model Type          if model\_type == 'XGB':              y = y.astype('category').cat.codes              model = xgb.XGBClassifier(seed=42)          else:              model = RandomForestClassifier(random\_state=42)          selected\_features = self.ranked\_features.copy()  # Start with all ranked features          best\_f1\_score = 0          print("Starting Recursive Feature Elimination...")          # Evaluate the model with all features first          X\_train, X\_test, y\_train, y\_test = train\_test\_split(X[selected\_features], y, test\_size=0.2, random\_state=42)          model.fit(X\_train, y\_train)          y\_pred = model.predict(X\_test)          best\_f1\_score = self.get\_f1\_score(y\_test, y\_pred)['F1 Score']          print(f"Initial F1 Score with all features: {best\_f1\_score:.4f}")          for feature in reversed(self.ranked\_features):              if feature in selected\_features:                  temp\_features = selected\_features.copy()                  temp\_features.remove(feature)                  print(f"Testing without feature: {feature}")                  # Train with updated feature set                  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X[temp\_features], y, test\_size=0.2, random\_state=42)                  model.fit(X\_train, y\_train)                  y\_pred = model.predict(X\_test)                  current\_f1\_score = self.get\_f1\_score(y\_test, y\_pred)['F1 Score']                  if current\_f1\_score >= best\_f1\_score:                      best\_f1\_score = current\_f1\_score                      selected\_features = temp\_features                      print(f"Removing {feature} improved or maintained F1 Score to {current\_f1\_score:.4f}")                  else:                      print(f"Keeping {feature} as removing it decreased F1 Score to {current\_f1\_score:.4f}")          # Save final RFE model          self.selected\_features = selected\_features          self.save\_final\_model(model, model\_type, "RFE")          print(f"Final selected features: {self.selected\_features}")          print(f"Final F1 Score: {best\_f1\_score:.4f}")      def perform\_spfs(self, model\_type):          self.selected\_features = []          target\_column = 'Label'            if target\_column not in self.df.columns:              raise KeyError(f"Column '{target\_column}' not found in dataset!")          y = self.df[target\_column]          X = self.df.drop(columns=[target\_column])          metrics\_data = []          print("Starting SPFS...")          self.ranked\_features = [feature for feature in self.ranked\_features if feature != target\_column]          for feature in self.ranked\_features:              self.selected\_features.append(feature)              X\_selected = X[self.selected\_features]              X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_selected, y, test\_size=0.2, random\_state=42)              if model\_type == 'XGB':                  y\_train = y\_train.astype('category').cat.codes                  y\_test = y\_test.astype('category').cat.codes                  model = xgb.XGBClassifier(seed=42)              else:                  model = RandomForestClassifier(random\_state=42)              model.fit(X\_train, y\_train)              y\_pred = model.predict(X\_test)              f1\_score = self.get\_f1\_score(y\_test, y\_pred)['F1 Score']              if f1\_score > max([m['F1 Score'] for m in metrics\_data] or [0]):                  print(f"Feature {feature} kept. New best F1 score = {f1\_score:.4f}\n")              else:                  self.selected\_features.remove(feature)                  print(f"Feature {feature} removed. F1 score did not improve.")          # Save final SPFS model          self.save\_final\_model(model, model\_type, "SPFS")          print(f"Final selected features: {self.selected\_features}")        def perform\_baseline(self, model\_type):          # Use all features except 'Label'          self.selected\_features = self.df.columns.tolist()          self.selected\_features.remove('Label')          # Extract features and target          X = self.df[self.selected\_features]          y = self.df['Label']          # Encode y if using XGBoost          if model\_type.lower() == "xgb":              y = y.astype('category').cat.codes  # Convert labels to numerical codes              model = xgb.XGBClassifier(seed=42)          else:              model = RandomForestClassifier(random\_state=42)          # Split data (80% train, 20% test)          X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)          # Train the model          model.fit(X\_train, y\_train)          # Save the trained model          model\_path = f"results/models/{self.dataset\_name}\_{model\_type}\_None.pkl"          os.makedirs(os.path.dirname(model\_path), exist\_ok=True)            with open(model\_path, 'wb') as model\_file:              pickle.dump(model, model\_file)          print(f"✅ Baseline Model ({model\_type}) saved to {model\_path}")        def get\_f1\_score(self, y\_true, y\_pred):          f1 = f1\_score(y\_true, y\_pred, average='weighted')          return {'F1 Score': f1}      def save\_final\_model(self, model, model\_type, method):          model\_filename = f"{self.dataset\_name}\_{model\_type}\_{method}.pkl"          model\_path = os.path.join(self.model\_dir, model\_filename)          with open(model\_path, 'wb') as f:              pickle.dump(model, f)          print(f"Model saved to {model\_path}")      def train\_and\_save\_model(self, selected\_features, model\_type, model\_path):          print(f"Training model using selected features: {selected\_features}")          target\_column = 'Label'            y = self.df[target\_column]          X = self.df[selected\_features]  # Use selected features!            # Select model type          if model\_type.lower() == "xgb":              y = y.astype('category').cat.codes              model = xgb.XGBClassifier(seed=42)          else:              model = RandomForestClassifier(random\_state=42)            # Split dataset          X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)            # Train model          model.fit(X\_train, y\_train)            # Save model          os.makedirs(os.path.dirname(model\_path), exist\_ok=True)          with open(model\_path, 'wb') as f:              pickle.dump(model, f)            print(f"✅ Model saved to {model\_path}") |

model\_testing.py: Cross-validation to evaluate the model efficiency

|  |
| --- |
| import pandas as pd  import os  import pickle  import time  from sklearn.model\_selection import train\_test\_split, KFold  from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score, confusion\_matrix  class ModelEvaluator:      def \_\_init\_\_(self, input\_file, average\_type, selected\_features, model\_type, feature\_selection\_method, model\_path, rank\_method):          self.rank\_method = rank\_method          self.input\_file = input\_file          self.feature\_selection\_method = feature\_selection\_method          self.selected\_features = selected\_features          self.average\_type = average\_type          self.model\_type = model\_type          self.model\_path = model\_path          self.df = pd.read\_csv(input\_file)          self.target\_column = "Label"          self.metrics\_log\_file = "results/metrics/evaluation\_log.csv"          self.removed\_features = list(set(self.df.columns) - set(self.selected\_features)- {self.target\_column})        def load\_model(self):          """Load pre-trained model from a file"""          if not os.path.exists(self.model\_path):              raise FileNotFoundError(f"Model file not found at {self.model\_path}")          with open(self.model\_path, 'rb') as model\_file:              model = pickle.load(model\_file)          return model      def evaluate\_model(self, y\_true, y\_pred):          accuracy = accuracy\_score(y\_true, y\_pred)          precision = precision\_score(y\_true, y\_pred, average=self.average\_type, zero\_division=0)          recall = recall\_score(y\_true, y\_pred, average=self.average\_type, zero\_division=0)          f1 = f1\_score(y\_true, y\_pred, average=self.average\_type)          cm = confusion\_matrix(y\_true, y\_pred)          fpr = cm[0][1] / cm[0].sum() if cm[0].sum() > 0 else 0          fnr = cm[1][0] / cm[1].sum() if cm[1].sum() > 0 else 0          return {              'Accuracy': accuracy,              'Precision': precision,              'Recall': recall,              'F1 Score': f1,              'FPR': fpr,              'FNR': fnr          }      def train\_and\_evaluate(self):          """Perform evaluation using 10-Fold Cross-Validation"""          start\_time = time.time()            if self.feature\_selection\_method == "none":              self.selected\_features = self.df.columns.tolist()              self.selected\_features.remove(self.target\_column)  # Ensure target is removed            X = self.df[self.selected\_features]          y = self.df[self.target\_column]          # Ensure y is encoded correctly for XGBoost          if "xgboost" in self.model\_type.lower():              y = y.astype('category').cat.codes  # Convert labels to numerical format          kf = KFold(n\_splits=10, shuffle=True, random\_state=42)          model = self.load\_model()          metrics\_data = []          print("Performing 10-Fold Cross-Validation...")            for fold, (train\_index, val\_index) in enumerate(kf.split(X, y), start=1):              X\_train, X\_val = X.iloc[train\_index], X.iloc[val\_index]              y\_train, y\_val = y.iloc[train\_index], y.iloc[val\_index]                # Ensure feature names match the ones used during training              if hasattr(model, "feature\_names\_in\_"):                  X\_val = X\_val[model.feature\_names\_in\_]              # Use pre-trained model for prediction              y\_pred = model.predict(X\_val)              metrics = self.evaluate\_model(y\_val, y\_pred)              metrics\_data.append(metrics)              print(f"Fold {fold}: F1 Score = {metrics['F1 Score']:.4f}")          # Compute average metrics          avg\_metrics = pd.DataFrame(metrics\_data).mean(numeric\_only=True)          total\_time\_taken = time.time() - start\_time          # Create a log entry          log\_entry = pd.DataFrame({              'Input File': [os.path.basename(self.input\_file)],              'Rank Method': [self.rank\_method],              'Feature Selection': [self.feature\_selection\_method],              'Model Type': [self.model\_type],              'Average Type': [self.average\_type],              'Number of Features': [len(self.selected\_features)],              'Removed Features': [", ".join(self.removed\_features)],              'Accuracy': [avg\_metrics['Accuracy']],              'Precision': [avg\_metrics['Precision']],              'Recall': [avg\_metrics['Recall']],              'F1 Score': [avg\_metrics['F1 Score']],              'FPR': [avg\_metrics['FPR']],              'FNR': [avg\_metrics['FNR']],              'Time Taken': [total\_time\_taken]          })          # Append results to the log file          os.makedirs(os.path.dirname(self.metrics\_log\_file), exist\_ok=True)          if not os.path.exists(self.metrics\_log\_file):              log\_entry.to\_csv(self.metrics\_log\_file, index=False)          else:              log\_entry.to\_csv(self.metrics\_log\_file, mode='a', header=False, index=False)          print(f"✅ Results appended to {self.metrics\_log\_file}") |

Full evaluation\_log.csv Results:

*Table 13 Full Evaluation Log csv file*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rank Method | Feature Selection | Model Type | Average Type | Number of Features | Accuracy | F1 Score | Ranking Time | Feature Selection Time | Evaluation Time | Total Time |
| WFI-XGB | RFE | XGB | micro | 15 | 0.9745 | 0.9745 | 8.1783 | 98.5319 | 55.3733 | 162.0835 |
| WFI-XGB | SPFS | XGB | micro | 15 | 0.9745 | 0.9745 | 8.1783 | 103.5423 | 61.2083 | 172.929 |
| WFI-XGB | RFE | XGB | weighted | 15 | 0.9745 | 0.9738 | 8.1783 | 98.5319 | 77.4295 | 184.1397 |
| WFI-XGB | SPFS | XGB | weighted | 15 | 0.9745 | 0.9738 | 8.1783 | 103.5423 | 57.1957 | 168.9164 |
| WFI-RF | rem\_1 | XGB | micro | 14 | 0.9735 | 0.9735 | 28.1665 | 7.3364 | 54.622 | 90.1249 |
| WFI-RF | SPFS | XGB | micro | 15 | 0.9731 | 0.9731 | 28.1665 | 86.1164 | 54.4091 | 168.692 |
| WFI-RF | None | XGB | micro | 15 | 0.9731 | 0.9731 | 28.1665 | 7.3364 | 56.1417 | 91.6446 |
| SP | SPFS | XGB | micro | 15 | 0.9731 | 0.9731 | 0.5398 | 104.1627 | 58.4338 | 163.1363 |
| SP | None | XGB | micro | 15 | 0.9731 | 0.9731 | 0.5398 | 9.7235 | 73.2484 | 83.5117 |
| WFI-XGB | None | XGB | micro | 15 | 0.9731 | 0.9731 | 8.1783 | 7.2277 | 71.6919 | 87.0979 |
| SP | RFE | XGB | micro | 14 | 0.9729 | 0.9729 | 0.5398 | 103.6918 | 57.0397 | 161.2713 |
| WFI-RF | rem\_1 | XGB | weighted | 14 | 0.9735 | 0.9727 | 28.1665 | 7.3364 | 55.2696 | 90.7725 |
| WFI-RF | RFE | XGB | micro | 14 | 0.9726 | 0.9726 | 28.1665 | 98.9481 | 57.5878 | 184.7024 |
| SP | rem\_1 | XGB | micro | 14 | 0.9726 | 0.9726 | 0.5398 | 9.7235 | 60.3108 | 70.5741 |
| WFI-RF | SPFS | XGB | weighted | 15 | 0.9731 | 0.9724 | 28.1665 | 86.1164 | 60.3455 | 174.6284 |
| WFI-RF | None | XGB | weighted | 15 | 0.9731 | 0.9724 | 28.1665 | 7.3364 | 60.0114 | 95.5143 |
| SP | SPFS | XGB | weighted | 15 | 0.9731 | 0.9724 | 0.5398 | 104.1627 | 55.8374 | 160.5399 |
| SP | None | XGB | weighted | 15 | 0.9731 | 0.9724 | 0.5398 | 9.7235 | 67.3089 | 77.5723 |
| WFI-XGB | None | XGB | weighted | 15 | 0.9731 | 0.9724 | 8.1783 | 7.2277 | 65.7008 | 81.1068 |
| SP | RFE | XGB | weighted | 14 | 0.9729 | 0.9721 | 0.5398 | 103.6918 | 55.5296 | 159.7612 |
| WFI-XGB | rem\_1 | XGB | micro | 14 | 0.9719 | 0.9719 | 8.1783 | 7.2277 | 61.0427 | 76.4487 |
| WFI-RF | RFE | XGB | weighted | 14 | 0.9726 | 0.9718 | 28.1665 | 98.9481 | 61.5052 | 188.6198 |
| SP | rem\_1 | XGB | weighted | 14 | 0.9726 | 0.9718 | 0.5398 | 9.7235 | 56.1641 | 66.4274 |
| WFI-XGB | rem\_1 | XGB | weighted | 14 | 0.9719 | 0.971 | 8.1783 | 7.2277 | 56.8606 | 72.2666 |
| SP | rem\_2 | XGB | micro | 13 | 0.9707 | 0.9707 | 0.5398 | 9.7235 | 53.381 | 63.6443 |
| SP | rem\_2 | XGB | weighted | 13 | 0.9707 | 0.9697 | 0.5398 | 9.7235 | 63.5611 | 73.8244 |
| SP | rem\_3 | XGB | micro | 12 | 0.9672 | 0.9672 | 0.5398 | 9.7235 | 59.2953 | 69.5586 |
| SP | rem\_4 | XGB | micro | 11 | 0.9671 | 0.9671 | 0.5398 | 9.7235 | 52.7595 | 63.0228 |
| SP | rem\_3 | XGB | weighted | 12 | 0.9672 | 0.9659 | 0.5398 | 9.7235 | 54.6473 | 64.9106 |
| SP | rem\_4 | XGB | weighted | 11 | 0.9671 | 0.9658 | 0.5398 | 9.7235 | 52.3542 | 62.6175 |
| WFI-XGB | rem\_2 | XGB | micro | 13 | 0.965 | 0.965 | 8.1783 | 7.2277 | 56.9024 | 72.3084 |
| WFI-XGB | rem\_3 | XGB | micro | 12 | 0.9641 | 0.9641 | 8.1783 | 7.2277 | 51.6252 | 67.0312 |
| WFI-XGB | rem\_2 | XGB | weighted | 13 | 0.965 | 0.9637 | 8.1783 | 7.2277 | 52.764 | 68.17 |
| WFI-XGB | rem\_3 | XGB | weighted | 12 | 0.9641 | 0.9627 | 8.1783 | 7.2277 | 51.7998 | 67.2057 |
| SP | RFE | RF | micro | 14 | 0.9623 | 0.9623 | 0.5398 | 388.1504 | 191.8283 | 580.5184 |
| WFI-RF | rem\_2 | RF | micro | 13 | 0.9621 | 0.9621 | 28.1665 | 23.6636 | 198.406 | 250.2361 |
| SP | rem\_5 | XGB | micro | 10 | 0.9619 | 0.9619 | 0.5398 | 9.7235 | 69.7961 | 80.0594 |
| WFI-RF | RFE | RF | micro | 14 | 0.9618 | 0.9618 | 28.1665 | 404.8508 | 226.6316 | 659.6489 |
| WFI-RF | rem\_1 | RF | micro | 14 | 0.9618 | 0.9618 | 28.1665 | 23.6636 | 222.0824 | 273.9125 |
| WFI-XGB | RFE | RF | micro | 13 | 0.9618 | 0.9618 | 8.1783 | 371.0178 | 196.9263 | 576.1224 |
| WFI-RF | None | RF | micro | 15 | 0.9617 | 0.9617 | 28.1665 | 23.6636 | 216.9796 | 268.8097 |
| SP | None | RF | micro | 15 | 0.9617 | 0.9617 | 0.5398 | 24.1702 | 215.05 | 239.76 |
| WFI-XGB | None | RF | micro | 15 | 0.9617 | 0.9617 | 8.1783 | 22.4982 | 215.966 | 246.6425 |
| SP | SPFS | RF | micro | 15 | 0.9615 | 0.9615 | 0.5398 | 338.4803 | 214.9426 | 553.9626 |
| WFI-RF | SPFS | RF | micro | 15 | 0.9614 | 0.9614 | 28.1665 | 341.6488 | 220.5252 | 590.3405 |
| SP | RFE | RF | weighted | 14 | 0.9623 | 0.9613 | 0.5398 | 388.1504 | 193.2668 | 581.957 |
| WFI-XGB | SPFS | RF | micro | 15 | 0.9612 | 0.9612 | 8.1783 | 214.3641 | 216.3696 | 438.9121 |
| WFI-RF | rem\_2 | RF | weighted | 13 | 0.9621 | 0.9611 | 28.1665 | 23.6636 | 198.8509 | 250.681 |
| WFI-XGB | RFE | RF | weighted | 13 | 0.9618 | 0.9608 | 8.1783 | 371.0178 | 196.1758 | 575.3719 |
| WFI-RF | RFE | RF | weighted | 14 | 0.9618 | 0.9607 | 28.1665 | 404.8508 | 222.2565 | 655.2738 |
| WFI-RF | rem\_1 | RF | weighted | 14 | 0.9618 | 0.9607 | 28.1665 | 23.6636 | 223.2803 | 275.1104 |
| WFI-XGB | SPFS | RF | weighted | 15 | 0.9618 | 0.9607 | 8.1783 | 214.3641 | 215.8557 | 438.3982 |
| WFI-RF | None | RF | weighted | 15 | 0.9617 | 0.9606 | 28.1665 | 23.6636 | 215.9117 | 267.7417 |
| SP | None | RF | weighted | 15 | 0.9617 | 0.9606 | 0.5398 | 24.1702 | 212.8829 | 237.5929 |
| WFI-XGB | None | RF | weighted | 15 | 0.9617 | 0.9606 | 8.1783 | 22.4982 | 215.5306 | 246.2072 |
| SP | SPFS | RF | weighted | 15 | 0.9615 | 0.9605 | 0.5398 | 338.4803 | 216.6426 | 555.6626 |
| WFI-RF | SPFS | RF | weighted | 15 | 0.9614 | 0.9604 | 28.1665 | 341.6488 | 218.278 | 588.0933 |
| SP | rem\_1 | RF | micro | 14 | 0.96 | 0.96 | 0.5398 | 24.1702 | 223.3047 | 248.0147 |
| SP | rem\_1 | RF | weighted | 14 | 0.961 | 0.9599 | 0.5398 | 24.1702 | 224.066 | 248.776 |
| SP | rem\_5 | XGB | weighted | 10 | 0.9619 | 0.9596 | 0.5398 | 9.7235 | 62.8392 | 73.1025 |
| WFI-XGB | rem\_1 | RF | micro | 14 | 0.9589 | 0.9589 | 8.1783 | 22.4982 | 223.4063 | 254.0829 |
| WFI-RF | rem\_3 | RF | micro | 12 | 0.9588 | 0.9588 | 28.1665 | 23.6636 | 206.2593 | 258.0894 |
| WFI-RF | rem\_3 | RF | weighted | 12 | 0.9588 | 0.9576 | 28.1665 | 23.6636 | 207.8593 | 259.6893 |
| WFI-XGB | rem\_1 | RF | weighted | 14 | 0.9589 | 0.9576 | 8.1783 | 22.4982 | 223.375 | 254.0516 |
| WFI-RF | rem\_4 | RF | micro | 11 | 0.9574 | 0.9574 | 28.1665 | 23.6636 | 213.8135 | 265.6436 |
| SP | rem\_2 | RF | micro | 13 | 0.9573 | 0.9573 | 0.5398 | 24.1702 | 230.227 | 254.937 |
| SP | rem\_2 | RF | weighted | 13 | 0.9573 | 0.956 | 0.5398 | 24.1702 | 232.1167 | 256.8267 |
| WFI-RF | rem\_4 | RF | weighted | 11 | 0.9566 | 0.9551 | 28.1665 | 23.6636 | 211.9905 | 263.8206 |
| WFI-RF | rem\_5 | RF | micro | 10 | 0.9548 | 0.9548 | 28.1665 | 23.6636 | 223.3571 | 275.1872 |
| SP | rem\_6 | XGB | micro | 9 | 0.9548 | 0.9548 | 0.5398 | 9.7235 | 57.1208 | 67.3841 |
| SP | rem\_3 | RF | micro | 12 | 0.9544 | 0.9544 | 0.5398 | 24.1702 | 239.3821 | 264.0921 |
| SP | rem\_4 | RF | micro | 11 | 0.9533 | 0.9533 | 0.5398 | 24.1702 | 250.9743 | 275.6843 |
| WFI-RF | rem\_5 | RF | weighted | 10 | 0.9548 | 0.9532 | 28.1665 | 23.6636 | 224.6456 | 276.4757 |
| SP | rem\_3 | RF | weighted | 12 | 0.9544 | 0.9528 | 0.5398 | 24.1702 | 238.7612 | 263.4712 |
| SP | rem\_6 | XGB | weighted | 9 | 0.9548 | 0.9518 | 0.5398 | 9.7235 | 53.8096 | 64.0729 |
| WFI-XGB | rem\_2 | RF | micro | 13 | 0.9517 | 0.9517 | 8.1783 | 22.4982 | 207.2949 | 237.9715 |
| WFI-XGB | rem\_4 | XGB | micro | 11 | 0.9514 | 0.9514 | 8.1783 | 7.2277 | 60.9729 | 76.3789 |
| SP | rem\_4 | RF | weighted | 11 | 0.9533 | 0.9512 | 0.5398 | 24.1702 | 251.2735 | 275.9835 |
| WFI-XGB | rem\_2 | RF | weighted | 13 | 0.9517 | 0.9501 | 8.1783 | 22.4982 | 208.6028 | 239.2794 |
| WFI-XGB | rem\_3 | RF | micro | 12 | 0.9499 | 0.9499 | 8.1783 | 22.4982 | 214.4513 | 245.1278 |
| WFI-RF | rem\_6 | RF | micro | 9 | 0.9493 | 0.9493 | 28.1665 | 23.6636 | 237.3647 | 289.1947 |
| WFI-XGB | rem\_4 | XGB | weighted | 11 | 0.9514 | 0.9488 | 8.1783 | 7.2277 | 56.1641 | 71.5701 |
| SP | rem\_5 | RF | micro | 10 | 0.9487 | 0.9487 | 0.5398 | 24.1702 | 269.1365 | 293.8465 |
| WFI-XGB | rem\_3 | RF | weighted | 12 | 0.9497 | 0.9476 | 8.1783 | 22.4982 | 217.598 | 248.2746 |
| WFI-RF | rem\_6 | RF | weighted | 9 | 0.9484 | 0.9464 | 28.1665 | 23.6636 | 234.7604 | 286.5905 |
| SP | rem\_5 | RF | weighted | 10 | 0.9487 | 0.9463 | 0.5398 | 24.1702 | 271.0955 | 295.8055 |
| WFI-RF | rem\_7 | RF | micro | 8 | 0.9441 | 0.9441 | 28.1665 | 23.6636 | 191.7316 | 243.5617 |
| WFI-RF | rem\_7 | RF | weighted | 8 | 0.9441 | 0.9421 | 28.1665 | 23.6636 | 195.1189 | 246.949 |
| SP | rem\_7 | XGB | micro | 8 | 0.9398 | 0.9398 | 0.5398 | 9.7235 | 48.4909 | 58.7542 |
| SP | rem\_6 | RF | micro | 9 | 0.9383 | 0.9383 | 0.5398 | 24.1702 | 254.6673 | 279.3773 |
| WFI-XGB | rem\_4 | RF | micro | 11 | 0.9379 | 0.9379 | 8.1783 | 22.4982 | 227.991 | 258.6676 |
| SP | rem\_6 | RF | weighted | 9 | 0.9383 | 0.9351 | 0.5398 | 24.1702 | 253.089 | 277.799 |
| WFI-XGB | rem\_4 | RF | weighted | 11 | 0.9379 | 0.935 | 8.1783 | 22.4982 | 194.7546 | 225.4312 |
| SP | rem\_7 | XGB | weighted | 8 | 0.9398 | 0.9349 | 0.5398 | 9.7235 | 56.7268 | 66.9901 |
| WFI-RF | rem\_2 | XGB | micro | 13 | 0.9343 | 0.9343 | 28.1665 | 7.3364 | 53.9466 | 89.4495 |
| WFI-RF | rem\_3 | XGB | micro | 12 | 0.9327 | 0.9327 | 28.1665 | 7.3364 | 53.5156 | 89.0185 |
| WFI-RF | rem\_4 | XGB | micro | 11 | 0.9321 | 0.9321 | 28.1665 | 7.3364 | 55.06 | 90.5629 |
| WFI-RF | rem\_8 | RF | micro | 7 | 0.9321 | 0.9321 | 28.1665 | 23.6636 | 202.0066 | 253.8367 |
| WFI-XGB | rem\_5 | XGB | micro | 10 | 0.9304 | 0.9304 | 8.1783 | 7.2277 | 50.9003 | 66.3063 |
| WFI-RF | rem\_8 | RF | weighted | 7 | 0.9321 | 0.9288 | 28.1665 | 23.6636 | 203.1551 | 254.9852 |
| WFI-RF | rem\_5 | XGB | micro | 10 | 0.9266 | 0.9266 | 28.1665 | 7.3364 | 51.0363 | 86.5392 |
| WFI-RF | rem\_2 | XGB | weighted | 13 | 0.9343 | 0.9257 | 28.1665 | 7.3364 | 58.2988 | 93.8017 |
| WFI-XGB | rem\_5 | XGB | weighted | 10 | 0.9304 | 0.9248 | 8.1783 | 7.2277 | 50.7896 | 66.1956 |
| WFI-RF | rem\_3 | XGB | weighted | 12 | 0.9327 | 0.924 | 28.1665 | 7.3364 | 53.5023 | 89.0052 |
| SP | rem\_7 | RF | micro | 8 | 0.9233 | 0.9233 | 0.5398 | 24.1702 | 211.847 | 236.557 |
| WFI-RF | rem\_4 | XGB | weighted | 11 | 0.9321 | 0.9232 | 28.1665 | 7.3364 | 51.8556 | 87.3585 |
| WFI-RF | rem\_6 | XGB | micro | 9 | 0.9217 | 0.9217 | 28.1665 | 7.3364 | 52.2315 | 87.7344 |
| WFI-RF | rem\_9 | RF | micro | 6 | 0.9198 | 0.9198 | 28.1665 | 23.6636 | 191.834 | 243.664 |
| WFI-XGB | rem\_6 | XGB | micro | 9 | 0.9191 | 0.9191 | 8.1783 | 7.2277 | 47.1635 | 62.5695 |
| WFI-RF | rem\_10 | RF | micro | 5 | 0.9182 | 0.9182 | 28.1665 | 23.6636 | 208.8816 | 260.7117 |
| SP | rem\_7 | RF | weighted | 8 | 0.9233 | 0.9175 | 0.5398 | 24.1702 | 210.9994 | 235.7094 |
| WFI-RF | rem\_5 | XGB | weighted | 10 | 0.9266 | 0.9168 | 28.1665 | 7.3364 | 56.7804 | 92.2833 |
| WFI-RF | rem\_9 | RF | weighted | 6 | 0.9198 | 0.915 | 28.1665 | 23.6636 | 192.6771 | 244.5072 |
| WFI-RF | rem\_10 | RF | weighted | 5 | 0.9182 | 0.9129 | 28.1665 | 23.6636 | 208.8889 | 260.719 |
| WFI-RF | rem\_7 | XGB | micro | 8 | 0.9103 | 0.9103 | 28.1665 | 7.3364 | 56.7201 | 92.223 |
| WFI-RF | rem\_6 | XGB | weighted | 9 | 0.9217 | 0.9101 | 28.1665 | 7.3364 | 50.5891 | 86.092 |
| WFI-XGB | rem\_6 | XGB | weighted | 9 | 0.9191 | 0.91 | 8.1783 | 7.2277 | 47.517 | 62.923 |
| WFI-RF | rem\_11 | RF | micro | 4 | 0.9039 | 0.9039 | 28.1665 | 23.6636 | 216.8949 | 268.725 |
| WFI-XGB | rem\_5 | RF | micro | 10 | 0.9039 | 0.9039 | 8.1783 | 22.4982 | 176.3528 | 207.0294 |
| WFI-RF | rem\_7 | XGB | weighted | 8 | 0.9103 | 0.8978 | 28.1665 | 7.3364 | 51.3897 | 86.8926 |
| SP | rem\_8 | RF | micro | 7 | 0.8976 | 0.8976 | 0.5398 | 24.1702 | 221.9051 | 246.6151 |
| WFI-XGB | rem\_5 | RF | weighted | 10 | 0.9039 | 0.8976 | 8.1783 | 22.4982 | 172.7294 | 203.406 |
| WFI-RF | rem\_11 | RF | weighted | 4 | 0.9039 | 0.8964 | 28.1665 | 23.6636 | 215.8312 | 267.6613 |
| WFI-RF | rem\_8 | XGB | micro | 7 | 0.8957 | 0.8957 | 28.1665 | 7.3364 | 48.6596 | 84.1625 |
| WFI-XGB | rem\_6 | RF | micro | 9 | 0.8916 | 0.8916 | 8.1783 | 22.4982 | 175.3519 | 206.0284 |
| WFI-XGB | rem\_7 | XGB | micro | 8 | 0.8914 | 0.8914 | 8.1783 | 7.2277 | 46.843 | 62.249 |
| SP | rem\_8 | RF | weighted | 7 | 0.8976 | 0.8879 | 0.5398 | 24.1702 | 222.8318 | 247.5418 |
| SP | rem\_8 | XGB | micro | 7 | 0.8871 | 0.8871 | 0.5398 | 9.7235 | 55.0001 | 65.2635 |
| WFI-RF | rem\_12 | RF | micro | 3 | 0.885 | 0.885 | 28.1665 | 23.6636 | 184.7289 | 236.559 |
| SP | rem\_9 | RF | micro | 6 | 0.8844 | 0.8844 | 0.5398 | 24.1702 | 218.8254 | 243.5354 |
| WFI-RF | rem\_10 | XGB | micro | 5 | 0.8827 | 0.8827 | 28.1665 | 7.3364 | 53.3513 | 88.8543 |
| WFI-RF | rem\_9 | XGB | micro | 6 | 0.8815 | 0.8815 | 28.1665 | 7.3364 | 45.9867 | 81.4896 |
| WFI-XGB | rem\_6 | RF | weighted | 9 | 0.8916 | 0.8815 | 8.1783 | 22.4982 | 172.8454 | 203.522 |
| SP | rem\_10 | RF | micro | 5 | 0.8808 | 0.8808 | 0.5398 | 24.1702 | 234.9464 | 259.6565 |
| WFI-RF | rem\_8 | XGB | weighted | 7 | 0.8957 | 0.8801 | 28.1665 | 7.3364 | 47.4788 | 82.9818 |
| SP | rem\_9 | XGB | micro | 6 | 0.8776 | 0.8776 | 0.5398 | 9.7235 | 50.3023 | 60.5656 |
| WFI-RF | rem\_12 | RF | weighted | 3 | 0.8857 | 0.876 | 28.1665 | 23.6636 | 182.7389 | 234.569 |
| SP | rem\_10 | XGB | micro | 5 | 0.8747 | 0.8747 | 0.5398 | 9.7235 | 52.8612 | 63.1245 |
| SP | rem\_8 | XGB | weighted | 7 | 0.8871 | 0.8746 | 0.5398 | 9.7235 | 52.9088 | 63.1721 |
| WFI-XGB | rem\_8 | XGB | micro | 7 | 0.8734 | 0.8734 | 8.1783 | 7.2277 | 49.6168 | 65.0228 |
| WFI-XGB | rem\_7 | XGB | weighted | 8 | 0.8914 | 0.8729 | 8.1783 | 7.2277 | 46.1808 | 61.5868 |
| SP | rem\_9 | RF | weighted | 6 | 0.8844 | 0.8711 | 0.5398 | 24.1702 | 219.9098 | 244.6198 |
| SP | rem\_10 | RF | weighted | 5 | 0.8808 | 0.8684 | 0.5398 | 24.1702 | 236.6167 | 261.3267 |
| WFI-RF | rem\_11 | XGB | micro | 4 | 0.8656 | 0.8656 | 28.1665 | 7.3364 | 46.457 | 81.9599 |
| SP | rem\_9 | XGB | weighted | 6 | 0.8776 | 0.8625 | 0.5398 | 9.7235 | 52.8954 | 63.1587 |
| WFI-RF | rem\_10 | XGB | weighted | 5 | 0.8827 | 0.8611 | 28.1665 | 7.3364 | 50.0003 | 85.5032 |
| WFI-RF | rem\_9 | XGB | weighted | 6 | 0.8815 | 0.8588 | 28.1665 | 7.3364 | 45.1721 | 80.675 |
| WFI-XGB | rem\_7 | RF | micro | 8 | 0.8574 | 0.8574 | 8.1783 | 22.4982 | 126.4491 | 157.1257 |
| SP | rem\_10 | XGB | weighted | 5 | 0.8729 | 0.8554 | 0.5398 | 9.7235 | 50.432 | 60.6953 |
| SP | rem\_11 | XGB | micro | 4 | 0.852 | 0.852 | 0.5398 | 9.7235 | 47.499 | 57.7623 |
| WFI-RF | rem\_13 | RF | micro | 2 | 0.8481 | 0.8481 | 28.1665 | 23.6636 | 200.846 | 252.6761 |
| SP | rem\_11 | RF | micro | 4 | 0.848 | 0.848 | 0.5398 | 24.1702 | 222.5105 | 247.2205 |
| WFI-XGB | rem\_8 | XGB | weighted | 7 | 0.8734 | 0.844 | 8.1783 | 7.2277 | 46.4764 | 61.8824 |
| WFI-RF | rem\_12 | XGB | micro | 3 | 0.8421 | 0.8421 | 28.1665 | 7.3364 | 54.8822 | 90.3851 |
| WFI-RF | rem\_11 | XGB | weighted | 4 | 0.8656 | 0.836 | 28.1665 | 7.3364 | 46.2119 | 81.7148 |
| WFI-XGB | rem\_8 | RF | micro | 7 | 0.8329 | 0.8329 | 8.1783 | 22.4982 | 114.2919 | 144.9685 |
| WFI-RF | rem\_13 | RF | weighted | 2 | 0.8481 | 0.8324 | 28.1665 | 23.6636 | 199.6825 | 251.5126 |
| WFI-XGB | rem\_9 | RF | micro | 6 | 0.8312 | 0.8312 | 8.1783 | 22.4982 | 81.5407 | 112.2173 |
| WFI-XGB | rem\_7 | RF | weighted | 8 | 0.8574 | 0.8301 | 8.1783 | 22.4982 | 126.5991 | 157.2756 |
| WFI-XGB | rem\_9 | XGB | micro | 6 | 0.8291 | 0.8291 | 8.1783 | 7.2277 | 44.6675 | 60.0735 |
| SP | rem\_11 | RF | weighted | 4 | 0.848 | 0.8283 | 0.5398 | 24.1702 | 222.0488 | 246.7588 |
| WFI-XGB | rem\_10 | XGB | micro | 5 | 0.8264 | 0.8264 | 8.1783 | 7.2277 | 47.194 | 62.5999 |
| WFI-XGB | rem\_11 | XGB | micro | 4 | 0.8264 | 0.8264 | 8.1783 | 7.2277 | 41.4423 | 56.8483 |
| WFI-XGB | rem\_10 | RF | micro | 5 | 0.8264 | 0.8264 | 8.1783 | 22.4982 | 39.4699 | 70.1465 |
| WFI-XGB | rem\_11 | RF | micro | 4 | 0.8263 | 0.8263 | 8.1783 | 22.4982 | 34.2725 | 64.9491 |
| SP | rem\_11 | XGB | weighted | 4 | 0.852 | 0.8257 | 0.5398 | 9.7235 | 47.5882 | 57.8515 |
| WFI-RF | rem\_13 | XGB | micro | 2 | 0.82 | 0.82 | 28.1665 | 7.3364 | 45.7813 | 81.2842 |
| WFI-XGB | rem\_12 | XGB | micro | 3 | 0.8136 | 0.8136 | 8.1783 | 7.2277 | 49.8336 | 65.2396 |
| WFI-XGB | rem\_12 | RF | micro | 3 | 0.8136 | 0.8136 | 8.1783 | 22.4982 | 29.1751 | 59.8517 |
| WFI-XGB | rem\_13 | XGB | micro | 2 | 0.807 | 0.807 | 8.1783 | 7.2277 | 44.3367 | 59.7427 |
| WFI-XGB | rem\_13 | RF | micro | 2 | 0.807 | 0.807 | 8.1783 | 22.4982 | 26.9387 | 57.6153 |
| WFI-XGB | rem\_14 | XGB | micro | 1 | 0.8048 | 0.8048 | 8.1783 | 7.2277 | 37.8282 | 53.2342 |
| WFI-XGB | rem\_14 | RF | micro | 1 | 0.8048 | 0.8048 | 8.1783 | 22.4982 | 25.0521 | 55.7287 |
| WFI-RF | rem\_14 | RF | micro | 1 | 0.8012 | 0.8012 | 28.1665 | 23.6636 | 69.5501 | 121.3802 |
| WFI-RF | rem\_12 | XGB | weighted | 3 | 0.8421 | 0.799 | 28.1665 | 7.3364 | 51.2851 | 86.788 |
| WFI-RF | rem\_14 | XGB | micro | 1 | 0.7955 | 0.7955 | 28.1665 | 7.3364 | 47.3624 | 82.8653 |
| SP | rem\_12 | XGB | micro | 3 | 0.786 | 0.786 | 0.5398 | 9.7235 | 46.2971 | 56.5605 |
| SP | rem\_13 | XGB | micro | 2 | 0.7849 | 0.7849 | 0.5398 | 9.7235 | 48.1331 | 58.3964 |
| SP | rem\_14 | XGB | micro | 1 | 0.7849 | 0.7849 | 0.5398 | 9.7235 | 28.7305 | 38.9938 |
| SP | rem\_14 | RF | micro | 1 | 0.7849 | 0.7849 | 0.5398 | 24.1702 | 21.9576 | 46.6676 |
| SP | rem\_13 | RF | micro | 2 | 0.7847 | 0.7847 | 0.5398 | 24.1702 | 60.6919 | 85.4019 |
| SP | rem\_12 | RF | micro | 3 | 0.7833 | 0.7833 | 0.5398 | 24.1702 | 104.3669 | 129.0769 |
| WFI-XGB | rem\_8 | RF | weighted | 7 | 0.8329 | 0.7762 | 8.1783 | 22.4982 | 117.5742 | 148.2507 |
| WFI-XGB | rem\_9 | RF | weighted | 6 | 0.8312 | 0.7727 | 8.1783 | 22.4982 | 82.8946 | 113.5712 |
| WFI-XGB | rem\_9 | XGB | weighted | 6 | 0.8292 | 0.7661 | 8.1783 | 7.2277 | 48.9212 | 64.3272 |
| WFI-RF | rem\_13 | XGB | weighted | 2 | 0.82 | 0.7599 | 28.1665 | 7.3364 | 44.8663 | 80.3692 |
| WFI-XGB | rem\_10 | RF | weighted | 5 | 0.8264 | 0.7596 | 8.1783 | 22.4982 | 37.9052 | 68.5818 |
| WFI-XGB | rem\_11 | RF | weighted | 4 | 0.8263 | 0.7595 | 8.1783 | 22.4982 | 34.451 | 65.1276 |
| WFI-XGB | rem\_10 | XGB | weighted | 5 | 0.8265 | 0.7594 | 8.1783 | 7.2277 | 43.5392 | 58.9452 |
| WFI-XGB | rem\_11 | XGB | weighted | 4 | 0.8263 | 0.7591 | 8.1783 | 7.2277 | 42.7077 | 58.1136 |
| WFI-XGB | rem\_12 | XGB | weighted | 3 | 0.8136 | 0.7306 | 8.1783 | 7.2277 | 48.752 | 64.158 |
| WFI-XGB | rem\_12 | RF | weighted | 3 | 0.8136 | 0.7306 | 8.1783 | 22.4982 | 29.0807 | 59.7573 |
| WFI-XGB | rem\_13 | XGB | weighted | 2 | 0.807 | 0.7231 | 8.1783 | 7.2277 | 40.5419 | 55.9479 |
| WFI-XGB | rem\_13 | RF | weighted | 2 | 0.807 | 0.7231 | 8.1783 | 22.4982 | 27.4431 | 58.1197 |
| WFI-RF | rem\_14 | RF | weighted | 1 | 0.8012 | 0.7211 | 28.1665 | 23.6636 | 70.1062 | 121.9363 |
| WFI-XGB | rem\_14 | XGB | weighted | 1 | 0.8048 | 0.7199 | 8.1783 | 7.2277 | 37.3 | 52.706 |
| WFI-XGB | rem\_14 | RF | weighted | 1 | 0.8048 | 0.7199 | 8.1783 | 22.4982 | 24.1353 | 54.8119 |
| WFI-RF | rem\_14 | XGB | weighted | 1 | 0.7942 | 0.7153 | 28.1665 | 7.3364 | 46.8033 | 82.3062 |
| SP | rem\_12 | XGB | weighted | 3 | 0.786 | 0.7107 | 0.5398 | 9.7235 | 46.6887 | 56.952 |
| SP | rem\_12 | RF | weighted | 3 | 0.7833 | 0.6985 | 0.5398 | 24.1702 | 101.9762 | 126.6863 |
| SP | rem\_13 | XGB | weighted | 2 | 0.7849 | 0.692 | 0.5398 | 9.7235 | 46.4174 | 56.6807 |
| SP | rem\_14 | XGB | weighted | 1 | 0.7849 | 0.692 | 0.5398 | 9.7235 | 27.8071 | 38.0704 |
| SP | rem\_13 | RF | weighted | 2 | 0.7847 | 0.692 | 0.5398 | 24.1702 | 63.7639 | 88.474 |
| SP | rem\_14 | RF | weighted | 1 | 0.7849 | 0.692 | 0.5398 | 24.1702 | 21.7626 | 46.4726 |

**Formatting Guideline for Sanstream Technical Reports**

**(Delete this section before submission)**

Follow guidelines as described in Table 1 consistently throughout the report.

Table 14: Formatting Guideline

|  |  |  |
| --- | --- | --- |
| **MS Word Feature** | **Where to set** | **Expected Value** |
| Style |  | **Header 1-4** for headings as appropriate.  **Normal** for the body |
| Font |  | **Arial 12 pt** for the body of the report  **Arial 11 pt and Italics** for Figure and Table captions  Example:  *Figure 1: Caption of figure 1* |
| Alignment |  | **Justify** for the body of the report  **Left** for text in Table  **Centre** for figure and table captions |
| Spacing |  | **1.5** for the body of the report |
| Captions |  | All figures and tables must have captions.  **Table** captions should be on **top** of the table  **Figure** captions should be at the **bottom** of the figure |

**Content Guidelines**

|  |
| --- |
| Header Types:  See the superscript on the Section Headers  a - Mandatory for all reports  b - Mandatory for survey reports  c - Mandatory for documenting a new methodology/approach proposed by you  d - Mandatory if describing a design of a software system, testbed, framework, etc.  e - Mandatory for reports on experimental work  *1. You are encouraged to use more sections (Heading 1) than listed here and subsections (Heading 2,3,4) as deemed appropriate for your report.*  *2. You can also slightly modify the title of the Section especially for the Design section to be more specific*  *3. Remove the superscripts from the headers and the “Formatting Guideline for Cistel Technical Report” section once the report is completed and ready for review.*  *4. Perform an Update Field on the Table of Contents, List of Figures and List of Tables.* |

Figure 11 Content Guidelines