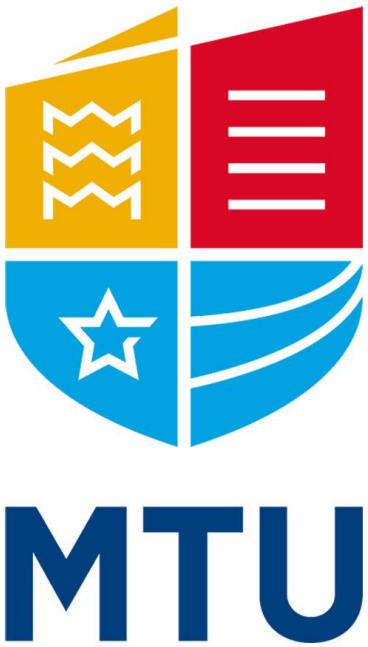


Real-Time Cyber Security Attack Detection For Network Traffic Using Machine Learning Models

Brian Higgins

Student No: R00239570

Supervisor: Dr. Haithem Afli

For the module DATA9003 – Research Project as part of the Master of Science in Data Science and Analytics, Department of Mathematics, 12/12/2023,

Declaration of Authorship

I, Brian Higgins, declare that this thesis titled 'Real-Time Cyber Security Attack Detection For Network Traffic Using Machine Learning Models' and the work presented in it are my own. I confirm that,

- This work was done wholly or mainly while in candidature for the Masters' degree at Munster Technological University
- Where any part of this thesis has previously been submitted for a degree or any other qualification at Munster Technological University or any other institution, this has been clearly stated
- Where I have consulted the published work of others, this is always clearly attributed
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project report is entirely my own work
- I have acknowledged all main sources of help
- I understand that my project documentation may be stored in the library at MTU, and may be referenced by others in the future

Signed: Brian Higgins

Date: 12/12/2023

Acknowledgment

Thanks to Dr. Haithem Afli and Dr. Vincent Cregan for their support during the project.

Contents

List of Figures

List of Tables

Abstract

Over the last number of years, Cyberattacks have become increasingly sophisticated which is posing an increased risk to individuals and organisations. Intrusion Detection Systems (IDS) need to improve to play a crucial role in preventing and safeguarding computer systems in real-time. Using Artificial Intelligence (AI) methods such as Machine Learning (ML) and Neural Networks (NN), IDS systems can be trained to recognise attacks in real-time. This research aims to conduct a comparative analysis of various ML techniques for detecting and classifying different types of network attacks using a dataset from the Canadian Institute of Cyber Security (CIC).

To simulate a real-world data flow, ML models will have a “baseline model” that will act as historical data and new data will be streamed into real-time models using Apache Kafka to detect attack instances. The models will then be retrained to simulate a real-world scenario of incremental learning. Using this approach allows the researchers to investigate the effectiveness of different ML models which will help to contribute to the development of more reliable and effective intrusion detection systems. The results of this research will provide valuable information on which of the models are best suited to enhance network security and allow faster detection of cybersecurity attacks in real time.

Clinical Relevance — This research will offer to enhance Intrusion Detection Systems by comparing the performance of various ML models in detecting attacks in real-time.

Keywords — Intrusion Detection Systems, Artificial Intelligence, Machine Learning, Simulation, Real-Time, Random Forest, Kmeans, Isolation Forest, One Class Support Vector Machine, CSE-CIC-IDS2018, Anomaly Detection, Cyber Security, Hoeffding Tree

1. Introduction

1.1. Background

Cybersecurity attacks have been increasing over the last number of years [1] and this has included two high-profile attacks on the HSE in 2021 [2] and Munster Technology University in 2023 [3], emphasizing the urgent need for effective Intrusion Detection Systems (IDS) to be able to identify and prevent these attacks. These attacks, while damaging to an organisation's reputation [4] can also be detrimental to a person's privacy [5] by releasing unauthorised personal information such as medical records, student records, financial records or private online correspondences.

Artificial Intelligence (AI) techniques such as Machine Learning (ML) and Neural Networks (NN) have shown huge promise in detecting cybersecurity attacks but still have many challenges [6]. ML and NN models can learn from historical data to identify anomalies and irregular patterns in network traffic to enable the detection of various attacks such as Man-in-the-Middle, Denial-of-Service, Malware Detection and Distributed Denial-of-Service attacks. This research will explore the CSE-CIC-IDS2018 [7] dataset where previous research on the CSE dataset has primarily concentrated on applying static training methods to the entire dataset, treating it as a singular event to identify the specific type of cyber-attack occurring within the network, this approach does not accurately reflect a real-world scenario where IDS systems need to continuously monitor network traffic to detect if an attack is happening. The HSE hackers were reportedly in the computer system for 8 weeks [8] before they were found. So the approach of having all the data upfront is unrealistic. IDS systems need to be designed to continuously monitor network data in real-time, triggering

alerts upon detection of any attack activities. This research will focus on this novel approach of aiming to classify attacks in real time and evaluate how different ML models perform.

The project architecture revolves around using Apache Kafka [9], a distributed streaming platform, for real-time data streaming and processing of the CSE-CIC-IDS2018 dataset. Initially, 50% of the data is employed to develop a baseline model reflecting a common practice in real-world scenarios to create a historic model. The remaining 50% of the data is segmented into batches and streamed through a Kafka Broker, which facilitates data transfer and management. Each batch is then evaluated to identify attack instances. The baseline model is then retrained with each batch to constantly adapt to evolving network traffic patterns.

1.2. Problem Statement

This project's research question: How do various machine learning models perform in a Kafka streaming environment for detecting cybersecurity attacks, and how does their effectiveness compare over time?

1.3. Objectives

The primary focus of this research is to assess various ML models within an Apache Kafka framework for real-time identification of attack instances using the CSE-CIC-IDS2018 dataset. Key objectives include:

- **Data Handing Approach:** Previous research treated the dataset as static, this research breaks the data into batches for streaming through Apache Kafka. This approach is better suited to a real-world scenario.

- **Model Effectiveness and Adaptability:** This research will explore both the baseline and real-time effectiveness of various supervised and unsupervised ML models and how they adapt over the 2,500 batches.
- **Training Efficiency:** A critical part of this research is to assess the training time, model size and performance metrics during each batch's retaining, as a balance of speed, size and accuracy are essential for an efficient IDS system.
- **Realistic Project Architecture:** A key objective is to create a streaming-based project outline using Apache Kafka to simulate a real-world approach.

2. Literature Review

This section provides an overview of existing literature that is relevant to this research, covering areas such as Intrusive Detection Systems (IDS), Machine Learning (ML), Neural Networks (NN) and the CSE-CIC-IDS2018. This literature review aims to establish a comprehensive understanding of the current state of research and identity gaps that this research aims to address.

2.1. Previous research

2.1.1. Intrusion Detection Systems

Intrusion Detection Systems (IDS) [10] are a critical component of an organisation's security infrastructure, vital in identifying and migrating unauthorised access, although their effectiveness remains under scrutiny [11]. IDS systems can be classified into two types: Host-Based and Network-Based [10]. Host-based IDS focus on monitoring individual machines while Network-Based IDS analyses traffic across the entire network. This research concentrates on Network-Based IDS systems.

2.1.2. Machine Learning Techniques with IDS Systems

Machine Learning (ML) models have demonstrated the potential in enhancing the detection abilities of IDS systems [12]. Various ML models, including Random Forest [13], Clustering and Gradient Boosting [14], Support Vector Machines (SVM) [15] and K-nearest Neighbour [16] have been applied in identifying anomalous patterns in network traffic. These models have been tailored to process large datasets while minimising false positives using datasets such as KDD99 [17] and NSL-KDD [18]. This research adopts a novel approach, deviating from training on full static datasets and instead focus on replicating real-world

approaches by streaming data into various ML models in batches. This method represents a significant and unique approach from previous studies.

2.1.3. Neural Networks with IDS Systems

Neural Networks (NNs) techniques like Convolutional Neural Networks (CNN) [19], Long Short Term Memory (LSTM) [20] networks and Autoencoders [21] have been employed to provide alternatives to traditional ML models at detecting Cyber Security threats. These methods include approaches like Principal Component Analysis (PCA) [22] and Mutual Information (MI) [23] which help in reducing the dimensionality of network traffic, which often encompasses hundreds of features. While NNs offer notable advantages, this research focuses primarily on ML models.

2.1.4. Previous Research on CSE-CIC-IDS20218 dataset

Previous research on the CSE-CIC-IDS2018 [7] dataset have successfully explored many ML models such as Random Forest [24], XG Boast [25], Support Vector Machines (SVM) [26], and K-Nearest Neighbour (KNN) with NN models such as CNN's, LSTM's and Autoencoders all achieving high performance results. Many of these studies had very high results with some studies having accuracy scores between 96% [27] and 100% [28], recall scores of 100% [28], [29] and precision scores of 100% [28], [29]. Such high performance could indicate some overfitting perhaps due to the imbalanced nature of the data.

2.1.5. Imbalanced Nature of Network Data

Data imbalance [30] is a significant challenge in IDS systems, as attack occurrences can vary greatly across different industries like banking, finance or e-commerce. Various techniques have been explored to deal with the imbalanced nature of this data with resampling methods [30] such as Undersampling (RU), Random Oversampling (RO) and Synthetic Minority Oversampling Techniques (RU-SMOTE) researched.

2.2. Gaps in the Previous Research this Project Addresses

2.2.1. Data Pre-Processing Steps

Prior research often lacked detailed data pre-processing steps, a critical aspect for replication and reliability. This research aims to address this gap by clearly documenting each preprocessing step along with its rationale.

2.2.2. Realistic Approach for Network Traffic Data

Previous research mostly used static training methods with the CSE-CIC-IDS2018 dataset and while static training is beneficial in certain situations it falls short in realistically simulating cyber-attacks for real-world IDS systems. This research addresses this limitation by using Apache Kafka to stream in the data in real-time, more accurately mirroring a real-world network environment. This allows the evaluation of ML models under conditions that more realistically reflect real Cyber-attacks.

2.2.3. Model Training with a Real-World Architecture

Static training methods fall short of replicating the complexity of a real-world IDS system. This research creates a larger and more realistic streaming-based project architecture for the evaluation of the ML model's performances.

2.2.4. Migrating the Risk of Overfitting

Previous studies on network anomaly detection often reported exceptionally high accuracies, sometimes reaching 100%, raising concerns about potential overfitting due to a lack of detailed preprocessing documentation. These consistently high results across various studies might also suggest that some models are inherently suited to the dataset characteristics, leading to naturally high accuracies. This research intends to delve deeper into these issues, selecting a Random Forest model as one of the primary models for investigation due to its 100% [28] results seen in previous research.

3. Methodology

3.1. Project Architecture

The development of a project architecture is crucial for this research as a real-life approach was very important. Different streaming options were investigated including; Apache Spark [31] and Amazon Sagemaker [32] with Apache Kafka [9] chosen.

Apache Kafka is a distributed streaming platform that allows for the processing of batches or streams of data in real-time. It is highly regarded and widely used in scenarios with real-time analytics and monitoring and is designed to work with large volumes of data. For this project, Apache Kafka serves as the backbone of the project for ingesting and processing real-time data streams. In Figure 1 we can see the outline of the architecture of this research.

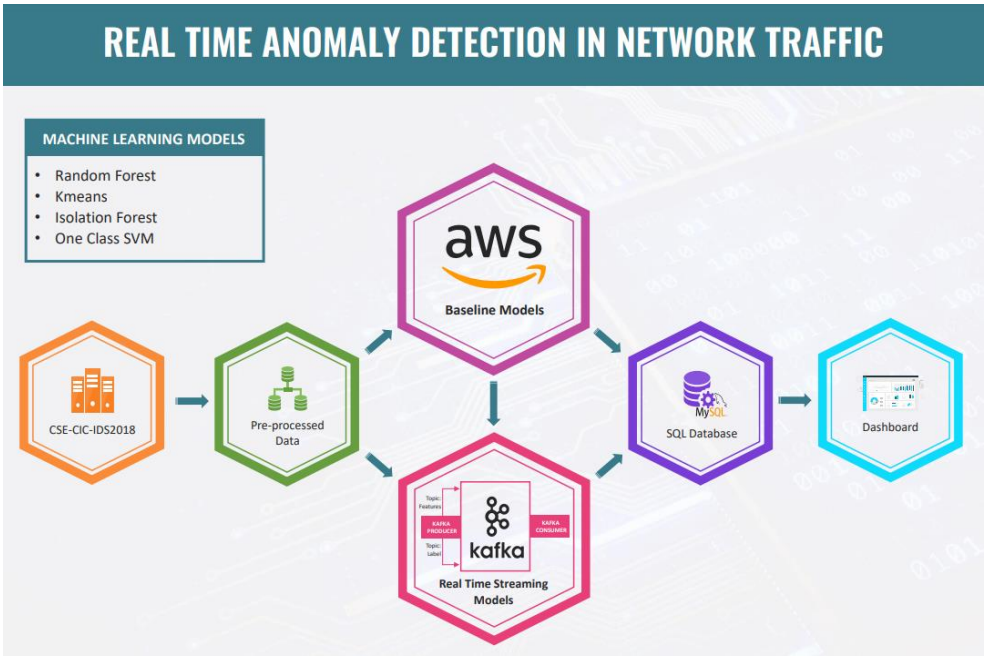


Figure 1: Project Architecture

3.1.1 Data Source

This research utilizes the CSE-CIC-IDS2018 dataset from the Canadian Institute of Cybersecurity which serves as the data source for the anomaly detection machine learning models. While the CSE-CIC-IDS2018 dataset has its limitations it remains one of the best freely available options. A more detailed description of the dataset and its implications for the research is provided in section 3.2.

3.1.2. Data Pre-Processing

Data preprocessing is a crucial aspect of this research, something that was frequently overlooked in many related studies as highlighted previously. Given the extensive size of the dataset and the processing of 2,500 batches for each real-time ML model, pre-processing was an essential step taken before model training. This approach was strategically designed to streamline the workflow and focus on the main research objectives. Typically, in real-world applications, the preprocessing steps would be integrated into the streaming process of the models, this would also allow for the use of different pre-processing steps to improve model performance. Detailed steps on the data preprocessing process are outlined in section 3.3.

3.1.3. AWS Cloud Computing for Baseline Models

In this research, the creation of a baseline model was pivotal in mirroring a real-world scenario for the detection of attack instances. Using baseline models on historical data a foundational understanding of the network traffic behaviour and patterns can be created. This baseline model is essential in a real-world anomaly detection system as it provides a reference point against which current benign and attack instances can be compared. Due to a lack of local computer resources the use of AWS instances for some of

the baseline models allowed for the baseline and real-time models to be run concurrently, AWS instances were mostly used to train the baseline models using 50% of pre-processed data. The ML models used are explored in Section 3.4.

3.3.4. Apache Kafka Core Framework

Apache Kafka forms the cornerstone of the research and an outline can be seen in Figure 2, acting as the primary server broker for streaming simulated batches into real-time ML models. Various distributed services were assessed for both cloud and local setup capabilities and their ability to scale. Ultimately, Apache Kafka was selected for its outstanding scalability, flexibility and its use of a producer-consumer approach which was found to be particularly effective for the research needs. This choice enabled efficient local development for the initial stages of the research but also the capacity for rapid scaling to a cloud-based system as needed.

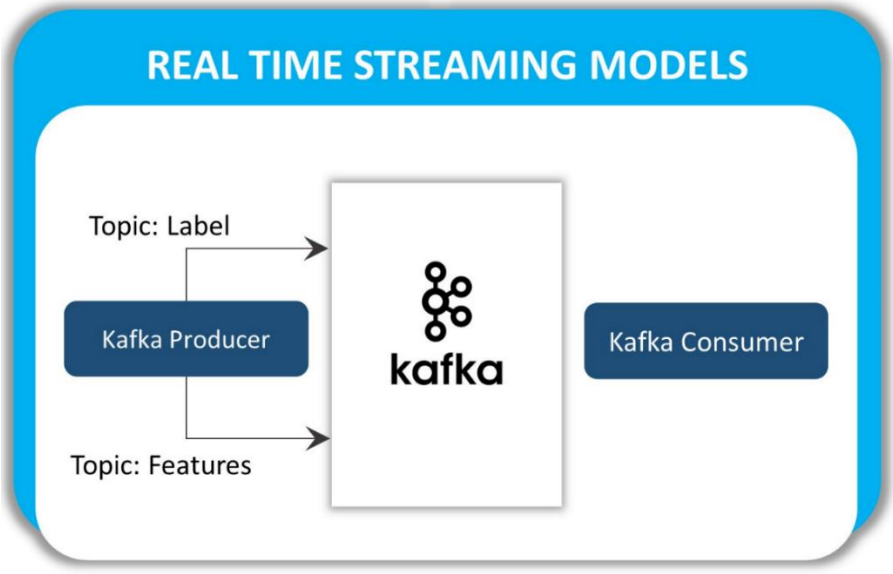


Figure 2: Kafka Architecture

Kafka Producer: Data Stream Simulation

The Kafka Producer plays a crucial role in processing the data for the real-time models. Batches of data, designed to replicate live network traffic, like those that enter an Intrusion Detection System, are organised into specific batch sizes for continuous streaming into the Apache Consumer via the Kafka Server, also called a Broker. Considerable time and effort was invested in exploring various batch sizes and compression methods which greatly enhanced the messages sent to the Kafka Broker from JSON to Apache Avro objects seen in Table 1. This shift reduced each batch message from 248Mb to 800Kb, keeping each batch below the optimal 1Mb for the Kafka Broker.

	Batch Numbers	Batch Sizes (Individual)	Object Types	Broker Message (Individual)
5% data	50 batches	16Mb	JSON file	248MB
0.2% data	5 batches	1Mb	Avro objects	800kb
100% data	2500 batches	1Mb	Avro objects	800kb

Table 1: Producer Batch Exploration

The batch sizes were incrementally increased from 5 to 100, 250 and finally 2,500 for each of the baseline models. Over 25,000 retrained batches were processed with twice that number trained in the development stage and discarded. The scalability and performance of Kafka were particularly noteworthy as once the difficult initial exploration of the process and setup was validated, scaling the producer to handle larger batches was straightforward and efficient.

Two distinct Kafka Topics were utilised to separate the labels and the features as shown in Figure 2 supporting the training, testing and evaluation of both the supervised and unsupervised models. A pivotal decision was made with the distribution of the attack instances in each batch. A choice of a random distribution, as opposed to a consistent

percentage, was chosen to better mirror the real world, an example of the frequency of attacks can vary seen in Figure 3. Models trained on varied attack percentages are expected to be more robust as they adapt to varying attack patterns, though it introduces extra complexity in analysing and comparing different model results. The Appendix includes an example of the Consumer Code for a detailed rundown.

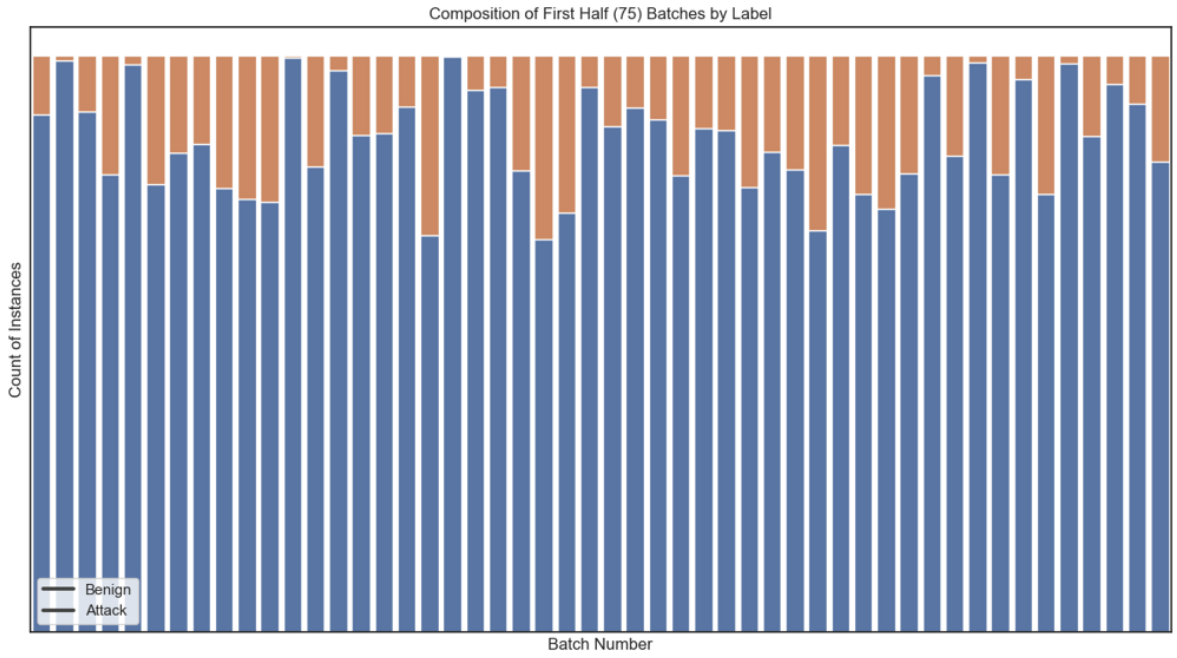


Figure 3: Example Distribution of Attack and Benign Instances Per Batch

Kafka Consumer: Real-Time Machine Learning Modelling

The Consumer processes the incoming messages sent through the broker and then loads and applies the schema to deserialise the Apache Avro objects into a suitable format for analysis and retraining. The Consumer then loads the specific baseline model and applies it to the incoming batches. Various performance metrics specific to each model type are computed and then stored in an SQL database for later analysis. Details of each model's metrics can be found in Section 3.6

Following the evaluation, the baseline model is then incrementally retrained with the current batch which updates the baseline model with the latest traffic, this ensures a dynamic adaptation to evolving data patterns, a critical aspect of real-time anomaly detection. This iterative approach of evaluation and enrichment not only refines the model's effectiveness but also maintains its relevancy and accuracy in a constantly changing network traffic environment.

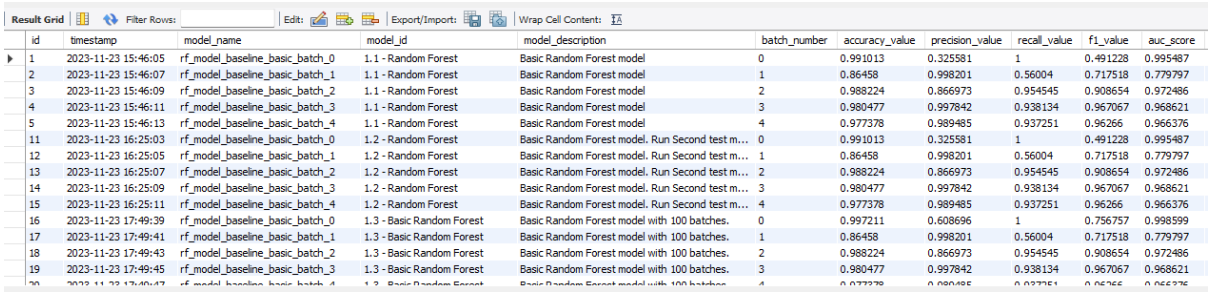
3.3.5. SQL Database for Result Management

The implementation of an SQL database in this research shows a realistic approach for storing and managing the vast array of model results. This approach is particularly important given the extensive volume of data generation from the real-time models where over 25,000 retained models' performances were stored. This structured approach to data management is essential for handling the complexity and scale of the data involved in this research. The SQL database not only provides an organised framework as seen in Figure 4 and Figure 5 but also facilitates efficient retrieval and analysis of the results in both individual notebooks and the eventual dashboard.

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	model_name	timestamp	model_paramete	confusion_matrix	accuracy_value	precision_value	recall_value	f1_value	auc_score
1	rf_model_baseline_basic	2023-11-11 13:45:04	n_estimators...	"[[1335345, 3680], [12801, 261893]]"	0.989787	0.986143	0.953399	0.969495	0.998741
2	rf_model_baseline_basic	2023-11-15 16:57:02	n_estimators...	[[1335345, 3680], [12801, 261893]]	0.989787	0.986143	0.953399	0.969495	0.998741
3	kmeans_model_baseline_basic	2023-11-15 21:12:25	{"algorithm": "...	NULL	NULL	NULL	NULL	NULL	NULL
4	iso_model_baseline_basic	2023-11-16 11:50:10	{"bootstrap": ...	[[579898, 89615], [125090, 12257]]	0.733901	0.471441	0.477695	0.473135	NULL
5	kmeans_model_baseline_best_serach	2023-11-22 15:40:46	{"algorithm": "...	NULL	NULL	NULL	NULL	NULL	NULL
6	iso_model_baseline_basic	2023-11-23 20:10:30	{"bootstrap": ...	[[0, 0, 0], [89615, 0, 579898], [12257, 0, 1250...	0.155033	0.0591452	0.303586	0.0990026	NULL
7	iso_model_baseline_best	2023-11-24 09:09:52	{"bootstrap": ...	[[0, 0, 0], [87211, 0, 582302], [17701, 0, 1196...	0.148286	0.0568162	0.290374	0.0950369	NULL
8	iso_model_baseline_basic	2023-11-29 10:12:07	{"bootstrap": ...	[[579898, 89615], [125090, 12257]]	0.733901	0.471441	0.477695	0.473135	NULL
9	oc_svm_model_baseline_basic_10_perc	2023-12-05 12:41:45	{"cache_size"...	NULL	NULL	NULL	NULL	NULL	NULL
10	oc_svm_model_baseline_basic_1_perc	2023-12-06 19:46:41	{"cache_size"...	NULL	NULL	NULL	NULL	NULL	NULL
11	oc_svm_model_baseline_basic_10_perc	2023-12-07 05:36:02	{"cache_size"...	NULL	NULL	NULL	NULL	NULL	NULL
12	oc_svm_model_baseline_basic_1_perc_nu_17	2023-12-07 13:23:43	{"cache_size"...	NULL	NULL	NULL	NULL	NULL	NULL
13	iso_model_baseline_best	2023-12-08 18:56:08	{"bootstrap": ...	[[0, 0, 0], [96336, 0, 573177], [23468, 0, 1138...	0.141138	0.0552497	0.276378	0.0920901	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 4: SQL Database - Baseline Models



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	timestamp	model_name	model_id	model_description	batch_number	accuracy_value	precision_value	recall_value	f1_value	auc_score
1	2023-11-23 15:46:05	rf_model_baseline_basic_batch_0	1.1 - Random Forest	Basic Random Forest model	0	0.991013	0.325581	1	0.491228	0.995487
2	2023-11-23 15:46:07	rf_model_baseline_basic_batch_1	1.1 - Random Forest	Basic Random Forest model	1	0.86458	0.998201	0.56004	0.717518	0.779797
3	2023-11-23 15:46:09	rf_model_baseline_basic_batch_2	1.1 - Random Forest	Basic Random Forest model	2	0.988224	0.866973	0.954545	0.908654	0.972486
4	2023-11-23 15:46:11	rf_model_baseline_basic_batch_3	1.1 - Random Forest	Basic Random Forest model	3	0.980477	0.997842	0.938134	0.967067	0.968621
5	2023-11-23 15:46:13	rf_model_baseline_basic_batch_4	1.1 - Random Forest	Basic Random Forest model	4	0.977378	0.989485	0.937251	0.96266	0.966376
11	2023-11-23 16:25:03	rf_model_baseline_basic_batch_0	1.2 - Random Forest	Basic Random Forest model. Run Second test m...	0	0.991013	0.325581	1	0.491228	0.995487
12	2023-11-23 16:25:05	rf_model_baseline_basic_batch_1	1.2 - Random Forest	Basic Random Forest model. Run Second test m...	1	0.86458	0.998201	0.56004	0.717518	0.779797
13	2023-11-23 16:25:07	rf_model_baseline_basic_batch_2	1.2 - Random Forest	Basic Random Forest model. Run Second test m...	2	0.988224	0.866973	0.954545	0.908654	0.972486
14	2023-11-23 16:25:09	rf_model_baseline_basic_batch_3	1.2 - Random Forest	Basic Random Forest model. Run Second test m...	3	0.980477	0.997842	0.938134	0.967067	0.968621
15	2023-11-23 16:25:11	rf_model_baseline_basic_batch_4	1.2 - Random Forest	Basic Random Forest model. Run Second test m...	4	0.977378	0.989485	0.937251	0.96266	0.966376
16	2023-11-23 17:49:39	rf_model_baseline_basic_batch_0	1.3 - Basic Random Forest	Basic Random Forest model with 100 batches.	0	0.997211	0.608696	1	0.756757	0.998599
17	2023-11-23 17:49:41	rf_model_baseline_basic_batch_1	1.3 - Basic Random Forest	Basic Random Forest model with 100 batches.	1	0.86458	0.998201	0.56004	0.717518	0.779797
18	2023-11-23 17:49:43	rf_model_baseline_basic_batch_2	1.3 - Basic Random Forest	Basic Random Forest model with 100 batches.	2	0.988224	0.866973	0.954545	0.908654	0.972486
19	2023-11-23 17:49:45	rf_model_baseline_basic_batch_3	1.3 - Basic Random Forest	Basic Random Forest model with 100 batches.	3	0.980477	0.997842	0.938134	0.967067	0.968621

Figure 5: SQL Database - RF models

3.3.6. Dashboard and Monitoring

Initial analysis of model results was conducted with Jupyter notebooks for the baseline and real-time model results providing an in-depth examination of each model's performance. While a dashboard was started to serve as an interactive visual interface its development was not completed due to a greater focus being placed on the in-depth analysis through the notebooks.

3.3.7. Coding and Documentation

The research was meticulously documented, developed and executed using a series of Jupyter Notebooks, each one addressing different aspects of the study. The following list of notebooks were used:

- Jupyter Notebook 1 – EDA – First Exploration on Data
- Jupyter Notebook 2 – Data Processing Steps
- Jupyter Notebook 3 – Data Processing Steps – Feature Selection – Brief Exploration
- Jupyter Notebook 5 – Notebook for Creating Datasets
- Jupyter Notebook 6 – AWS Notebook for ML Models
- Jupyter Notebook 7 – Kafka Producer
- Jupyter Notebook 8.1 – Kafka Consumer – Data Ingestion and Storage
- Jupyter Notebook 8.2 – Kafka Consumer – Model Performance and Retraining

- Jupyter Notebook 9 – Dashboard – This was moved to Pycharm
- Jupyter Notebook 10 – Various Pieces of Code
- Jupyter Notebook 11 - Exploring SQL Results – Baseline Models
- Jupyter Notebook 12 – Exploring SQL Results – Real-Time Models
- Pycharm – Dashboard Creation – In Progress

3.2. Data Source

The dataset used in this project is CSE-CIC-IDS20218 from the Canadian Institute for Cyber Security at the University of New Brunswick [33]. It simulates network traffic as seen in Figure 6 with six types of cybersecurity attacks (DDoS, DoS, Brute Force, Botnet, Infiltration and Web Attacks) to mimic real-world scenarios. This dataset includes several days of network traffic, encompassing over 16 million data points across 80 features in 6.4GB of data.

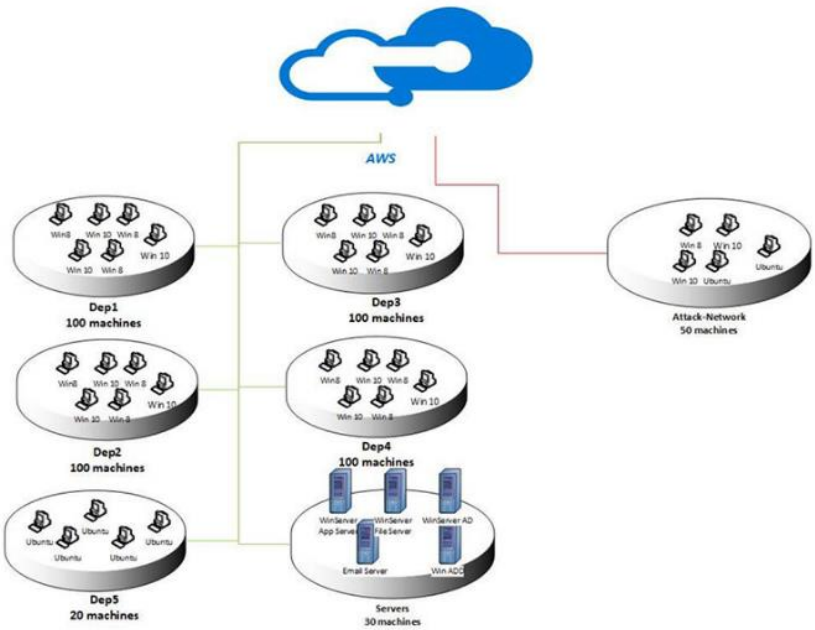


Figure 6: CSE-CIC-IDS2018 Network Diagram

The data contains both benign and attack labels which come from different parts of the network, including single machines, servers and switches. The data is highly imbalanced as seen in Table 2 where benign instances make up 83% of the data and combined attack instances less than 17%.

Traffic Type	Distribution
Benign	83.1%
DDoS	7.7%
DoS	4.0%
Brute Force	2.4%
Botnet	1.8%
Infiltration	0.994%
Web Attack	0.006%

Table 2: CSE-CID-IDS2018 Attack Traffic

3.3. Data Pre-Processing

In this section, we outline the data preprocessing steps employed in this research. The literature review highlighted a noticeable lack of detailed pre-processing steps in many existing studies and recognising this shortfall, this research outlines each of the steps taken and a rationale for each step. While different research may take different steps and there can be different justifications for steps taken, a clear outline of which steps are taken allows for transparency, credibility and reproducibility of the results, a step that is crucial to all research.

It is important to note that in a real-world scenario, data preprocessing typically occurs in real-time before the data is used in the ML models. However, for this research and

to maintain simplicity, the CSE-CIC-IDS2018 dataset underwent extensive cleaning and preprocessing before being partitioned to the baseline and real-time models. This approach ensures a clean and uniform dataset allowing for the research to focus on its objectives. Steps included:

1. **Remove Missing Values:** Identified rows with missing or NaN values were removed to maintain data consistency and avoid distortion from unknown values. This ensures that the data is consistent and is not being influenced by unknown values. This was feasible due to the low occurrence of missing values in the dataset.
2. **Remove Duplicate Rows:** A standard practice is to eliminate identical rows across all the columns to prevent dataset inflation and bias. Duplicate rows were few for the dataset and a decision was made to retain these rows as the nature of network traffic entries might represent regular traffic patterns.
3. **Remove Duplicate Columns:** Duplicate columns were removed to ensure accuracy in the training process by eliminating redundant data, which could lead to incorrect labelling.
4. **Remove Incorrect "label" Entries:** Entries with the wrong labels were identified and excluded. This step was crucial to maintain the integrity and reliability of the dataset.
5. **Drop Metadata columns:** Metadata features such as IDs and IP addresses were removed. Previous research showed these features lead to model overfitting.
6. **Handle Infinity Values:** Columns with "inf" values were addressed. Given their rarity, these values were removed to prevent disruptions in the training process.
7. **Normalization:** Using MinMaxScaler, and since all the features were numeric, normalisation was used to enhance the efficiency of the training models.
8. **Convert the Labels to Numeric Values:** Convert the label column of "attack type" and "benign" into a numeric value to ensure compatibility with all ML models.

9. **Data Randomisation:** Initially, streaming data based on timestamps was considered. However, due to the distribution of the attack instances in clusters, this was deemed infeasible. Instead, data was randomised, a common practice in the previous research.
10. **Optimise Object Types**: Investigating data type conversion for efficiency was considered but became redundant following normalisation. This step was initially aimed at reducing memory usage and speeding up model processing considering the data is 6.4GB in size and the demands of real-time models.
11. **Save to a Parquet Format:** To reduce memory overhead, data stored in pandas data frames was saved to a Parquet file format, significantly reducing memory efficiency and faster model training.
12. **Drop empty columns:** Columns with no data were eliminated to streamline the dataset and focus on relevant features. 9 columns were dropped with only zero values.
13. **Subsequent Steps:** The data was later randomised between the different attack and benign instances and then split into two equal subsets. One subset was used for the baseline model training, while the second supported the real-time anomaly detection segment.

3.4. Exploratory Data Analysis (EDA)

EDA, detailed in Figure 7 shows a total of 16,137,182 instances across the ten datasets after pre-processing. Day 4 has 7,889,295 instances and Day 9 has the lowest with 318,181 instances. The datasets are combined so this is not an issue in this research approach.

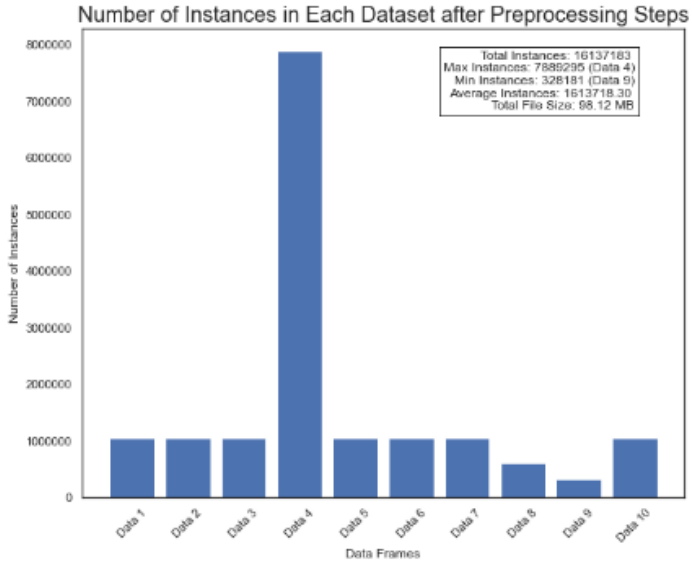


Figure 7: CSE-CIC-IDS2018 Full Dataset

Figure 8 presents the class distribution within dataset 1. There is a significant imbalance between the benign instances and the two types of attacks: SSH-BruteForce and FTP-BruteForce in this case. This can be seen in both the barplot and the pie chart which visually represents the same data. This imbalance can affect model performance.

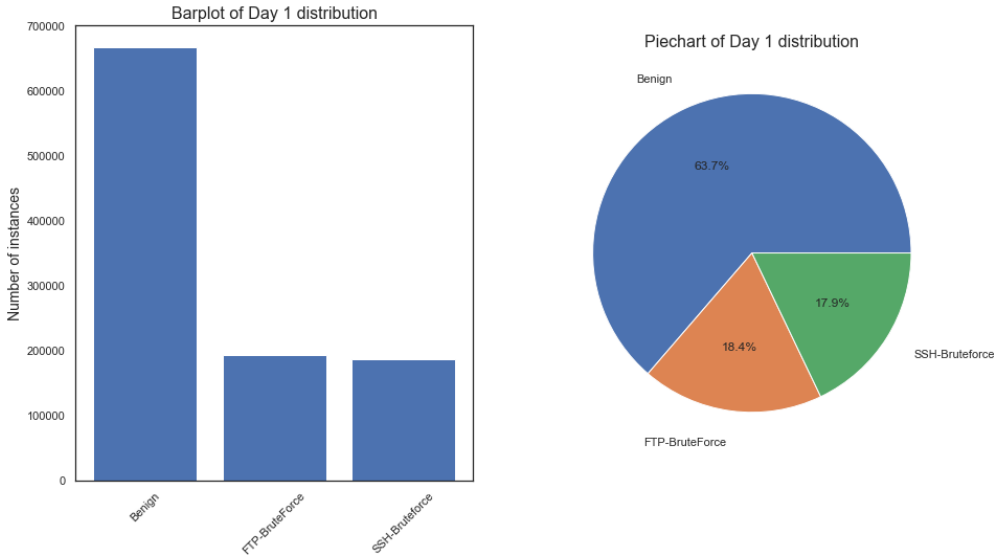


Figure 8: Barplot and Pie Chart for Day 1

Figure 9 shows the distribution of the label instances for day 1. With more data, an approach would have been taken to use ML models to identify these “clusters” of attacks but due to the limited amount of data for this case, the data was randomised, which enabled models to identify attack instances based on features rather than on the clusters. This method, while a comprise, was consistent with the previous research but still allowed for the development of a robust, scalable IDS system to identify attack instances.

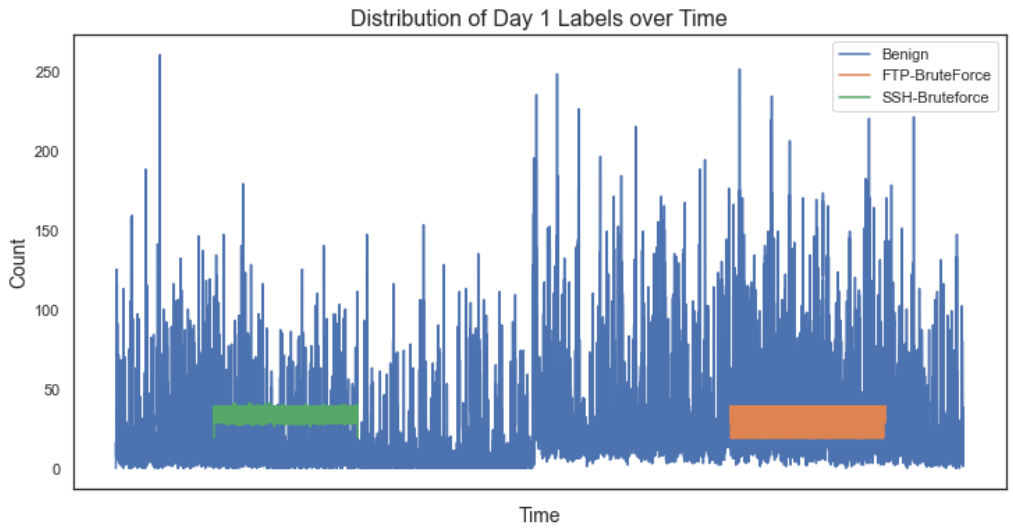


Figure 9: Distribution of Label Instances for Day 1

3.5. Machine Learning Models

This research employs a two-part approach to model training, incorporating baseline models and real-time models as outlined in Figure 1. Different models with different strengths were chosen to investigate a range of results, including Random Forest, Hoeffding Tree, Kmeans, Isolate Forest, One-Class SVM and Stochastic Gradient Descent (SGD). Factors that influenced the model selection included; previous research highlighting models like Random Forest, supervised and unsupervised models to offer a range of diversity, models that perform well in a real-time environment and with large data.

3.5.1. Random Forest Model

Supervised model selected because of its impressive performance in the literature review, making it an excellent model to establish a baseline for this research. It is predicted to have a strong performance and allows this research to explore the high performance seen in the previous studies, however, it lacks incremental learning which means it will need to retrain each time for each new batch, which is a drawback for real-time modelling.

3.5.2. Hoeffding Tree Model

Another Supervised model that specially designed for streaming data and can learn incrementally as data arrives continuously in each batch. Models can be updated without retraining the entire historical dataset, which is unrealistic for network traffic, suggesting promising results, as well as its decision tree roots.

3.5.3. K-means Clustering Model

Unsupervised clustering model which can be adapted for incremental learning without processing the complete dataset allowing it to update clusters as new batches are introduced.

3.5.4. Isolated Forest Model

Unsupervised model that is typically used for anomaly detection in static data which research shows has the ability to be adapted for real-time streaming data. This research will explore this ability which could allow Isolate Forests to be a good option because of its efficacy in detecting outliers.

3.5.5. One Class SVM Model

Another unsupervised learning model that is often used for novelty detection. Its computational needs were a consideration to be included given the volume of network traffic data and the potential for this model to perform well for the baseline models.

3.5.6. SGD Model

Stochastic Gradient Descent (SGD) in research shows it does well with large datasets, making it a viable option for network traffic. It also has a "Hinge" loss function that enables it to function similarly to an SVM for real-time decision-making.

For the baseline models, a default parameter model was run for each of the models. A Grid Search was then conducted for selected models to optimise their parameters and find the best parameters for these models. While not the primary focus of this research, optimising models is an important step to enhance model performance. The main parameters found are listed in Table 3 with more details in Section 4.

Model	Basic Model *(Default Parameters)*	Best Model *(Best Grid Search)*
Random Forest	N_estimators=100	*Not Needed*
Hoeffding Tree	N_trees=100	*Not Needed*
Kmeans	Clusters =2	Clusters =5
Isolate Forest	N_estimators=100	*Optimal Parameters Unidentified*
One Class SVM	1% of Data	20% of Data
SDG	Max_iter=1000	*Not Needed*

Table 3: Model Hyperparameters

3.6. Model Evaluation Metrics

The diversity of the different ML models meant no single metric was available for supervised and unsupervised models, or different unsupervised models. Table 4 shows a range of metrics that were stored for each model in the SQL database for the baseline and the retrained real-time models. Although a direct comparison can not be made for all models due to the different types of metrics, employing both performance metrics and universal ones like Model Size and Model Training Time enables evaluation of each model's efficacy in a general comparative manner.

Random Forest/ Hoeffding Tree	Kmeans	Isolate Forest	One Class SVM	SGD
Accuracy, Precision, Recall, F1	Silhouette Score	Accuracy, Precision, Recall, F1	Accuracy, Precision, Recall, F1	Accuracy, Precision, Recall, F1
Confusion Matrix	Cluster Centre	Confusion Matrix	Confusion Matrix	Confusion Matrix
Feature importance	Elbow Method	Anomaly Score	AUC Score	
AUC Score	Cluster Visualisation Plot	Anomaly Detection Plot	RUC Curve	
RUC Curve				
Precision-Recall Curve				
Model Size	Model Size	Model Size	Model Size	Model Size
Model Training Time	Model Training Time	Model Training Time	Model Training Time	Model Training Time

Table 4: Model Evaluation Metrics

Table 5 acts as a model reference framework, assigning different identifiers to the various baseline and real-time models. This structure allows a clearer system to identify models in the report and throughout the code part of the research.

Models	Baseline	5 Batches	100 Batches	250 Batches	2500 Batches
Radom Forest	1.0	1.1	1.3	1.4	1.5
Hoeffding Tree	2.0	2.1	2.2	2.3	2.4
Kmeans Basic	3.0	3.1	3.2	3.3	3.4
Kmeans Best	4.0	4.1	4.2	4.3	4.4
Isolate Forest Basic	5.0	5.1	5.2	5.4	5.4
Isolate Forest Best	6.0	6.1	6.2	6.4	6.4
One Class SVM Model 1	7.0	7.1	7.2	7.4	7.4
One Class SVM Model 2	8.0	8.1	8.2	8.4	8.4
One Class SVM Model 3	9.0	9.1	9.2	9.4	9.4
One Class SVM Model 4	10.0	10.1	10.2	10.4	10.4
SDG	11.0	11.1	11.2	11.4	11.4

Table 5: Machine Learning Model IDs

3.7. Additional Considerations: Feature Selection

The original dataset comprised of over 400 features that were created with a network protocol analyser that was used to capture and inspect data that travelled back and forth across the computer network in real-time. Recognising the need for a more manageable dataset, CSE and CIC streamlined it to 80 essential features. During this research, the dataset was further refined from 80 to 70 features in the pre-processing steps. However, due to time constraints and the extensive aspects of the research areas, a detailed exploration of feature selection was not feasible. However, Feature Importance was still explored and future research will focus on applying these insights to improve the model's performance.

4. Results

4.1. Baseline Model Results

In this research, the computational abilities of AWS instances were used to facilitate parallel processing of the baseline models and the development and processing of the real-time models concurrently. Optimization techniques such as Sklearn Grid Search [34] were then used to refine some models, were appropriate. While optimisation of these models was not a primary goal, given the extensive research in previous literature, this research still aimed to enhance models that could be improved.

4.1.1 Random Forest Baseline Model

The Random Forest baseline model exhibited excellent performance in identifying attack instances, a performance that is not surprising given the results seen in the literature review. Performance metrics show an accuracy score of 98.97% indicating the model's general effectiveness. The precision with a value of 98.61% and recall at 95.33% were notably very high as reflected in the F1 score of 96.95% underscoring the model's balanced classification. The confusion matrix in Table 6 further confirms the model's precision and recall ability displaying a substantial count of True Positives and True Negatives. Interestingly, given that the data is highly imbalanced, the model has performed very well. This could indicate the supervised Random Forest models are learning patterns.

	Predicated: No	Predicated: Yes
Actual: No	1335345	3680
Actual: Yes	12801	261893

Table 6: Random Forest Baseline Confusion Matrix

A ROC curve and a Precision-Recall curve illustrated in Figure 10 show an area under the curve (AUC) score of 99.87% indicating an exceptional ability to differentiate between normal benign and attack traffic. The Precision-Recall plot shows once again the high precision across all levels of recall until a quick shape decline.

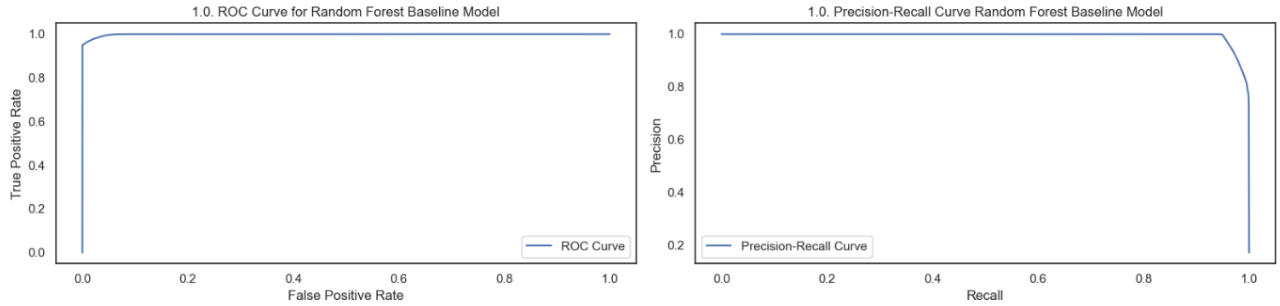


Figure 10: Random Forest Baseline ROC Curve

A detailed investigation into feature importance was not a primary focus of this research. However, it will be for the next phase to improve each model's performance so they are worth exploring. Figure 11 shows the features “Dst Port”, and “Tot Fwd Pkts” are some of the most important features. Indicating that certain ports and high amounts of traffic are good indicators of attack instances.

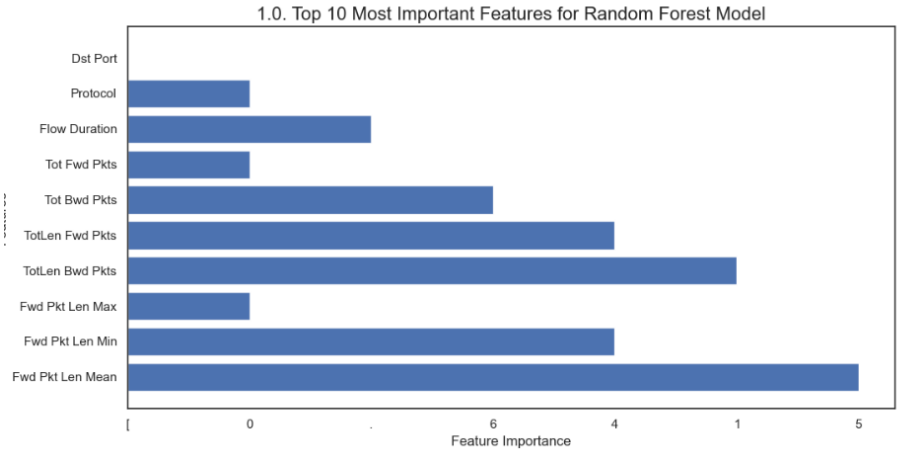


Figure 11: Random Forest Baseline Feature Importance

However, it took 4725 seconds to run the model and the end model default size before compression was 548.88Mb, Since the size of the model is a factor in the retraining of the real-time models this can be an important consideration.

4.1.2. Hoeffding Tree Baseline Model

The Hoeffding model was expected to match the results of the Random Forest model due to their decision tree similarities and it has met these expectations. The model achieved an accuracy of 99.02%, precision of 99.65%, recall of 94.58% and an F1 score of 97.01% indicating very high levels of predictive reliability. The model's AUC score was also exceptionally high at 99.75%, demonstrating its ability to differentiate between the different classes.

Another significant advantage of the Hoeffding Tree model over the Random Forest model was its fast training speed of 1396 seconds. Speed of training is a major factor in real-time systems. Also, it has been designed for incremental training whereas the Random Forest model has not.

4.1.3. Kmeans Basic Baseline Model

The Kmeans clustering model successfully partitions the network traffic into two distinct clusters achieving a Silhouette Score of 0.426138 showing a moderate delineation between the clusters. Figure 12 demonstrated this by using a 2D PCA reduced cluster visualisation that delineated the data into two principal components for a simplified visual analysis, which would not be possible with 70 features otherwise. The first cluster is tightly grouped towards the right side in three groups, which suggests the potential for additional cluster divisions. Additionally, this is the larger cluster and the dataset is imbalanced so likely indicates the benign instances. The second cluster is more scattered towards the

lower right, displaying greater variability and possibly representing outlier characteristics of attack instances.

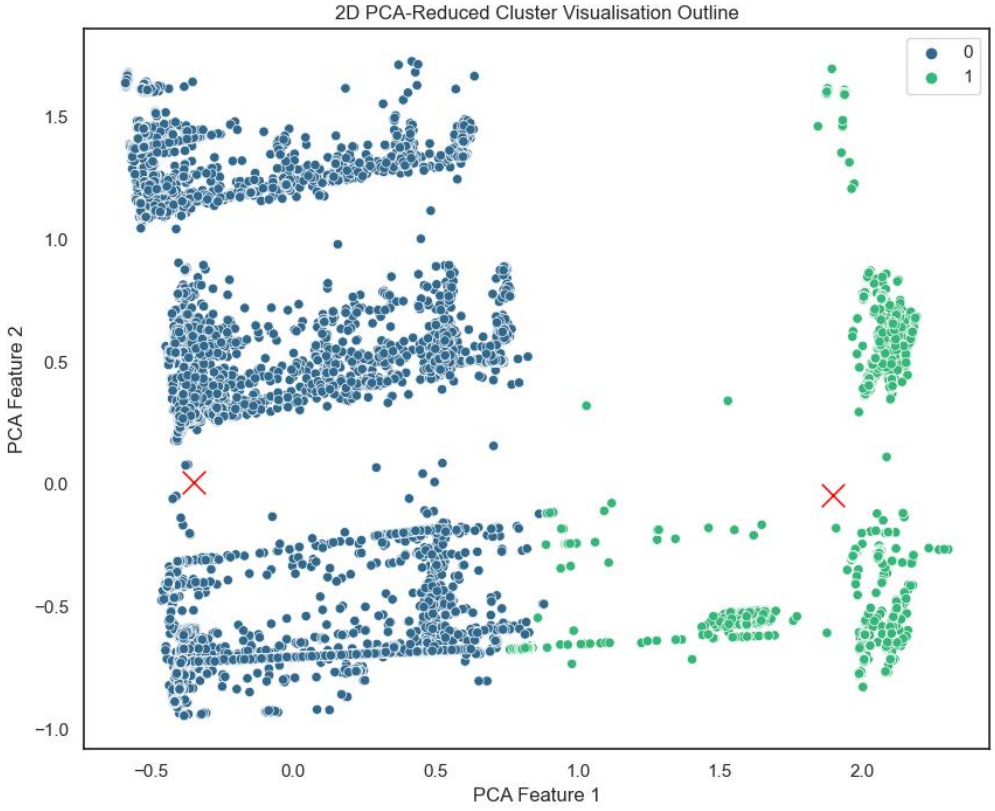


Figure 12: Kmeans Basic Baseline 2D PCA Cluster Visualisation Plot

Table 7 shows the most dominant features in each cluster with a cluster characteristic that matches the plot in Figure 12, pointing at feature importance. The model training was completed in a fast 44.5696 seconds with the model size being a compact 24.63Mb. Both of which are large improvements on the Random Forest model.

Clusters	Dominate Features	Cluster Characteristics
Cluster 0	Flow Duration, Fwd IAT Tot, Fwd IAT Min, Fwd IAT Mean, Flow IAT Min	Characterised by significant network flow duration and inter-arrival times suggesting this cluster captures typical traffic patterns.
Cluster 1	Protocol, PSH Flag Cnt, Fwd Seg Size Min, ACK Flag Cnt, RST Flag Cnt	Distinguished by protocol-related activities and specific flag counts, potentially indicate of anomalous or irregular traffic patterns.

Table 7: Kmeans Basic Baseline Dominate Features

The elbow method shown in Figure 13 further aids in identifying the optimal number of clusters providing a clear visual cue for refining the Kmeans Best Model. This indicates a cluster of 5 or 6 would likely perform well.

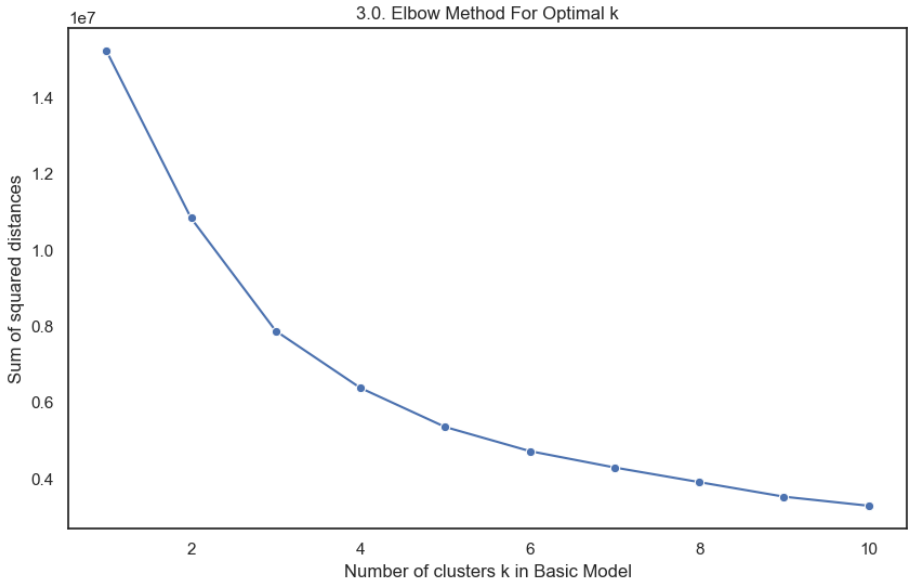


Figure 13: Kmeans Basic Baseline Elbow Method Plot

4.1.4. Kmeans Best Baseline Model

To determine the optimal number of clusters for the Kmeans Best model, a search was conducted to explore various cluster configurations from 2 to 7. This exploration process revealed that a 5-cluster model performed the best with a silhouette score of 0.449331, a slight increase over the basic kmeans model of 2 clusters. Figure 14 shows another 2D PCA plot to help visualise the new clusters.

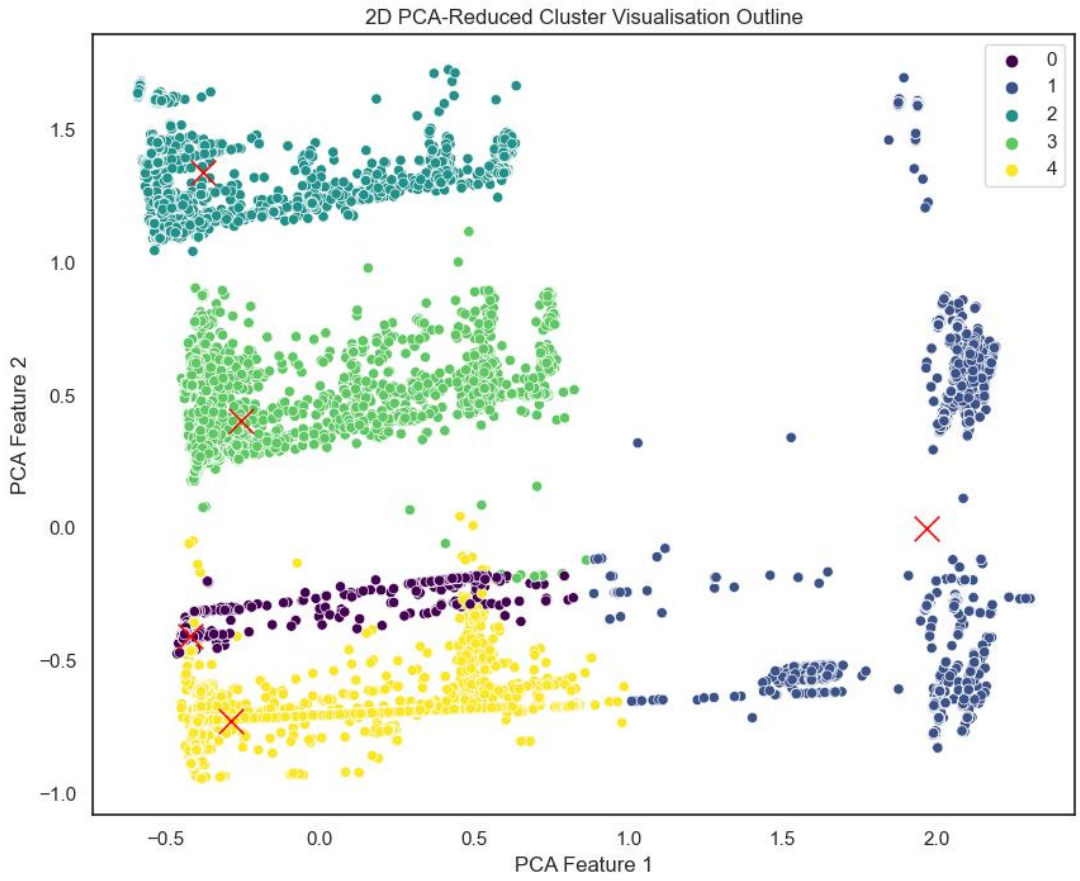


Figure 14: Kmeans Best Baseline 2D PCA Cluster Visualisation Plot

Table 7 shows the most dominant features in each of the clusters with clusters 0 and 1 potentially showing the clusters as anomalous network traffic, indicating different attacks can have different important features. The model's training time has increased to 100.452 seconds and the model file size is 24.93Mb demonstrating the efficiency of the model in terms of resources and storage. Whether the slight increase in Silhouette Score is worth the extra clusters and training time can be explored more in the real-time model training.

Clusters	Dominate Features	Cluster Characteristics
Cluster 0	Fwd Seg Size Min, Init Bwd Win Byts, PSH Flag Cnt, ECE Flag Cnt, RST Flag Cnt	These flags often relate to specific network errors, if occurring frequently might signify anomalous activities.
Cluster 1	Fwd PSH Flags, Dst Port, Protocol, Fwd Seg Size Min, ACK Flag Cnt	Anomalous activities often show through unusual port activity or the use of certain flag combinations in unexpected contexts.
Cluster 2	Fwd Pkt Len Min, Pkt Len Min, Bwd Pkt Len Min, Fwd Seg Size Min, Protocol	Involves small packet sizes and standard protocols, indicating basic, regular network traffic
Cluster 3	Flow IAT Mean, Fwd IAT Mean, Fwd IAT Min, Fwd IAT Tot, Flow Duration	Shows uniform data flow with regular timing, typical of consistent network activity.
Cluster 4	Init Fwd Win Byts, Init Bwd Win Byts, Protocol, Fwd Seg Size Min, PSH Flag Cnt	Characterized by window size and protocol, likely related to maintaining stable connections

Table 8: Kmeans Best Baseline Dominate Features

4.1.5. Isolate Forest Basic Baseline Model

For the Isolation Forest model, the following metrics were recorded: an accuracy of 73.39%, a precision of 47.14%, a recall of 47.77%, and an F1 score of 47.31%. This model does well in identifying anomalies by isolating outliers, demonstrating a moderate ability to distinguish between benign and anomalous traffic, however, the lower values are relatively low indicating a need for further optimisation to enhance its performance. Perhaps caused by the imbalanced nature of the data. The confusion matrix in Table 9 correctly identified 579,898 normal instances while erroneously marking 89,615 as anomalies. It accurately detected 12,257 anomalies but misclassified 125,090 instances as anomalous. This indicates a substantial number of false positives (instances marked as anomalies that are normal). This could indicate the model is not performing as well due to the imbalanced nature of the data.

	Predicated: No	Predicated: Yes
Actual: No	579898	89615
Actual: Yes	125090	12257

Table 9: Isolate Forest Basic Confusion Matrix

The training of the Basic Isolation Forest model was notable fast, only taking just over 5 seconds with a very lightweight model of only 0.61Mb. These characteristics underscore the model's practicality for real-time anomaly detection where both speed and storage are crucial but an issue with the detection of the minority class.

While silhouette score is not an Isolate Forest metric usually used it can be calculated by creating clusters based on the anomaly score to get a score to use as a comparison later. The Silhouette Score for the basic Isolate Forest model was 0.249068 showing the normal instances are not distinctly separated from the attack instances. An interesting approach but perhaps not useful.

4.1.6. Isolate Foret Best Baseline Model

This section aimed to use Grid Search with Isolate Forest to optimise the Basic Baseline Isolate Model above and with high expectations of results. Despite careful planning and dividing the project into twelve Jupyter Notebooks a coding error was made with the baseline ISO model to create the Silhouette Score which led to incorrect outcomes in the ISO grid searches.

Identifying the Error

The ISO model categorises data as inliers(1) and outliers(-1), however, an adjustment was made to the baseline model when creating the Silhouette Score where the prediction was remapped from -1 to 1 and 0 to 1. This remapping, while small was critical as with the next models ran on the same notebook causing the confusion matrix to change from a 2x2

to a 3x3 matrix. This is a red flag but the author did not notice as a confusion matrix was not run at that time and the error was assumed to be in the grid search's model performance rather than a coding mistake. This led to very long extended model searches and unproductive grid searches, including one model that ran for 9 days before being halted.

ISO Models	Parameters	Best Accuracy
Grid Search 1	n_estimators [200,300,400], max_samples_options = [0.5, "auto"], contamination = 0.13	14.8%
Grid Search 2	n_estimators = [100, 200, 300, 400, 500], max_samples = [0.25, 0.5, 0.75, "auto"] contamination = [0.05, 0.1, 0.15, 0.2].	N/A – The model ran for 9 days and was cancelled
Grid Search 3	n_estimators = [100, 200, 300] max_samples = [0.25, 0.5] contamination = [0.1, 0.15]	28%

Table 10: Grid Search Hyperparameters

Lessons Learned

1. **Project Management:** Even with a structured detailed project approach, errors can slip through. A better approach would have been to separate each model into separate notebooks.
2. **Attention to detail:** In complex multifaceted projects, small coding errors can have big consequences. Steps should be taken to separate sections to further reduce issues.

Despite the challenges and not having some metrics for later evaluation, one of the Grid Search models can still be used to run real-time modelling as the model itself has not been affected by the above code error on the confusion matrix and performance metrics.

4.1.7. One-Class SVM Baseline Model

Support Vector Machines are known for their computational intensity, often requiring substantial time and computer resources. To enhance their results, feature engineering would be used to reduce the dimensionality of the data. This is future research

for this project but they remain worthwhile models to explore. Table 11 shows a range of models explored with the 100% data model being omitted due to its impractical computational requirements.

Data	Shape	Parameters	Training Time	Accuracy
100%	6454872, 70	Linear Kernel, nu=0.1	Cancelled after 5 days	N/A
0.1%	6455, 70	Linear Kernel, nu=0.1	1.5 seconds	N/A
1%	64549,70	Linear Kernel, nu=0.1	66 seconds	32.1166%
10%	645487,70	Linear Kernel, nu=0.1	1506 seconds	32.1169%
1%	64549, 70	Linear Kernel, nu=0.17	101 seconds	36.5973%
10%	645487, 70	Linear Kernel, nu=0.17	1678 seconds	36.5975%
20%	1290974, 70	Linear Kernel, nu=0.17	Cancelled due to time constraints	

Table 11: One Class SVM Model Results

Exploring the results it's clear that the time to run these models increases exponentially with larger datasets. Running the model with 0.01% data takes 1.5 seconds, whereas 1% takes between 66-101 seconds, the 10% data model takes 1506-1678 seconds and the 20% data model took over 72 hours and was cancelled. However, there was not a large increase in the model's performance for this extra time. The increase in the data and training time only gives a small increase in accuracy.

4.1.8. SGD Baseline Model

The SGD model was used as an alternative to the SVM model, especially for its capability to use incremental learning which the One-Class SVM model lacks. The SGD model also has the “hinge” function as previously mentioned that is commonly used in linear SVM’s. This Hinge function allows the SGD model to mimic some of the characteristics of an SVM, especially with decision boundary foundations.

The SGV model performed very well with an accuracy of 93.31% indicating a high level of correctness in its predictions. Precision at 88.79% and recall at 87.07% show good predictions as well. An area where the SDG model performed very well was in its training speed which only took 16 seconds, highlighting its suitability for real-time models.

4.2. Real-Time Streaming Model Results

4.2.1. Random Forest Real-Time Model

Although the Random Forest model does not support incremental learning, meaning that for each batch the model is retraining itself each time, it is worthwhile exploring its performance across repeated batches. Figure 15 shows the model's accuracy across the 2,500 batches, despite each batch randomising the number of attack instances, the model is consistently achieving high accuracy. However, there are only 3,325 rows in each batch with a small number of attack instances meaning the model is performing exceptionally well. A linear regression line indicates a stable trend despite the model's limitations in incremental learning.

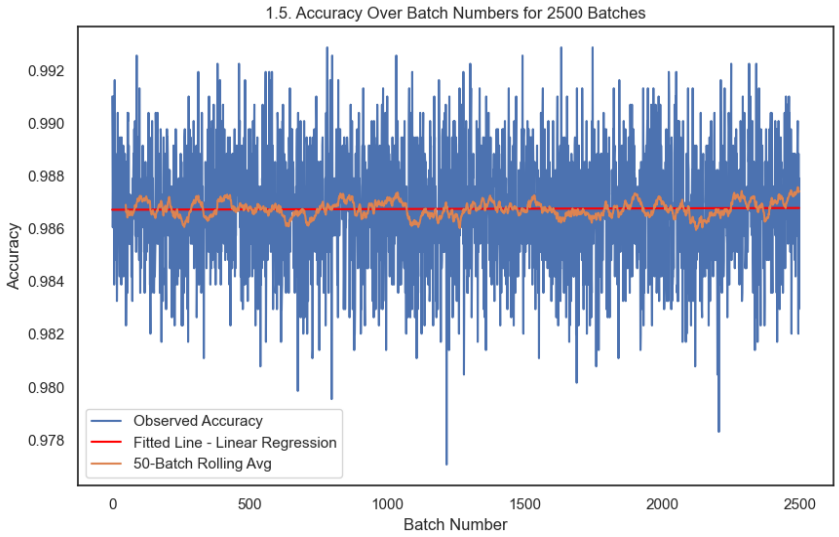


Figure 15: Random Forest Accuracy over 2500 batches

Showing that interesting results can still be found from this model, Figure 16 visualises the feature importance across the 2,500 batches. Features represented with smaller boxplots are considered less important over the iterations of the batches with features like “protocol”, “psh flags” and “urg flags” for example. Features with larger or higher boxplots suggest greater importance such as “flow duration”, “header len” and “subflow fwd pkts”. Identifying which features consistently influence a model's predictions can help with future retraining and would be important for future considerations of which features to keep and disregard.

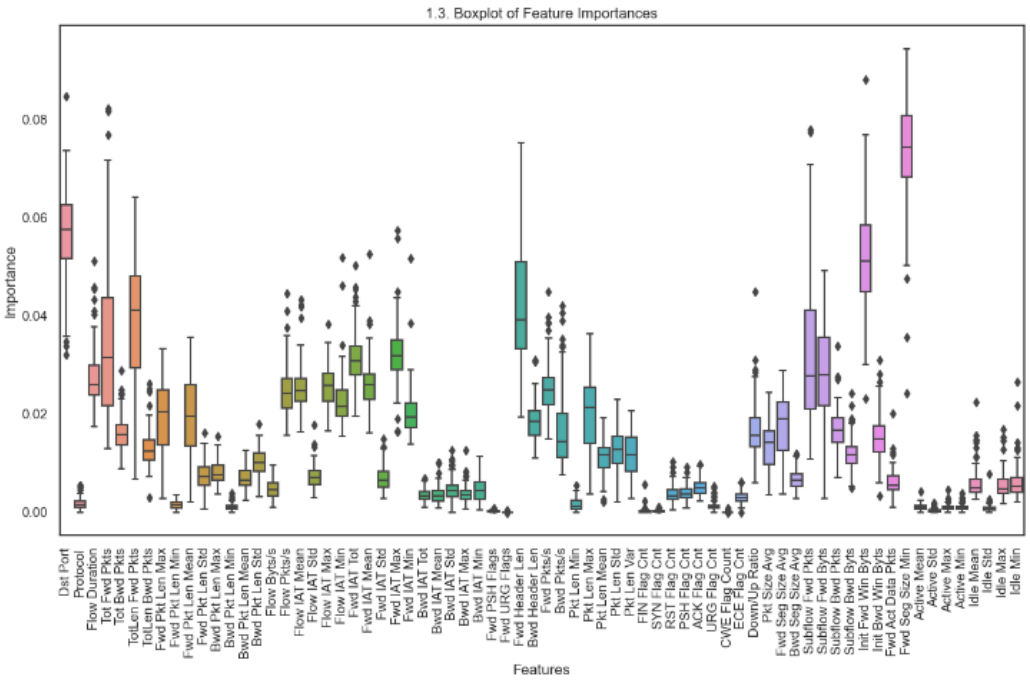


Figure 16: Random Forest Feature Importance

4.2.2. Hoeffding Tree Real-Time Model

Figure 17 captures the accuracy of the Hoeffding Tree model’s performance over the 2,500 batches. The model's accuracy fluctuates, which is expected from the varying attack instances in each batch but the tight clustering of the results shows the model is robust in its predictions maintaining an impressive average of 98.92 %. However, the density of the data

points across such a large number of batches introduces a noisy plot where it is difficult to see any trends. There is a drop in performance on one batch which is interesting as this dip was not seen with other models. Something to explore at a later point.

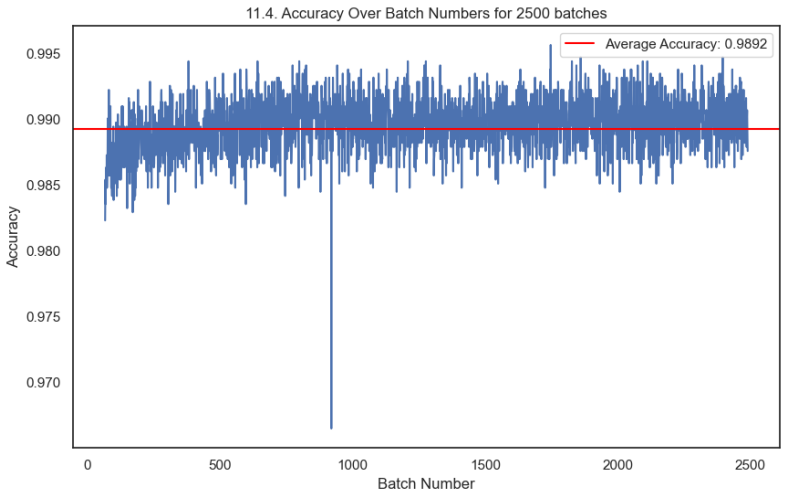


Figure 17: Hoeffding Tree Accuracy over 2500 batches

Figure 18 displays the application of using a 50-batch rolling average that clears up the plot and smooths out the short-term fluctuations. This shows a small but positive upward trend, indicated by the red trend line of a linear regression line. This indicates the model is not just maintaining its high level of performance but is also showing signs of incremental improvements across the dataset. Precision, Recall, F1 and AUC scores also follow this slight upward trend and very good results. Given the model's already exceptionally high performance, these modest improvements suggest a refinement in the model's refinement over time. Other window sizes were also explored with similar results.

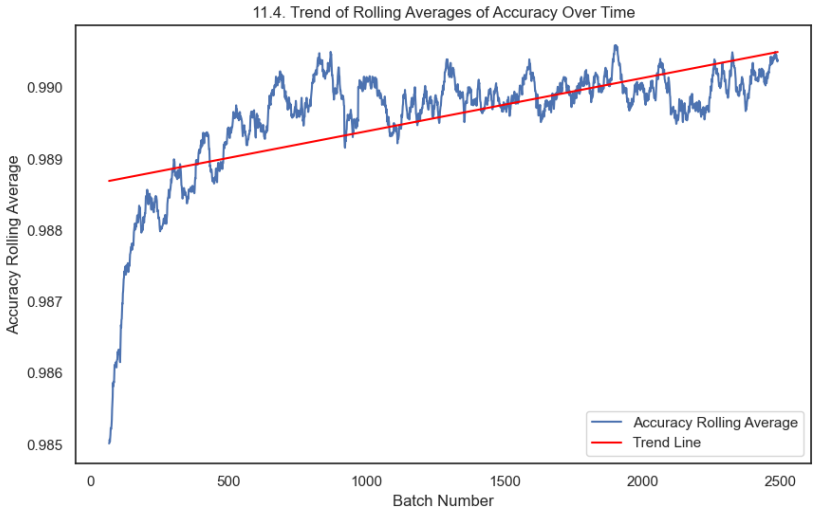


Figure 18: Hoeffding Tree Rolling Accuracy over 2500 batches

4.2.3. Kmeans Basic Real-Time Model

The Silhouette Score for the Kmeans basic model is relatively stable over the iterations of the batches with values in the low 40s, a range typically showing good clustering performance. The overall score remains within a small band suggesting the Kmeans clustering is reliable across the 2,500 batches. Figure 19 shows a small decline in the Silhouette Score over time.

In contrast, Figure 20 shows a marked downward trend in the centroid changes, showing that as more batches are processed the model centroids become more stable. This is a common sign of convergence which reflects the model's capacity to accurately capture the central characteristics of the clustered data. However, this downward trend can comprise the model's ability to adjust to new data and needs to be kept an eye on.

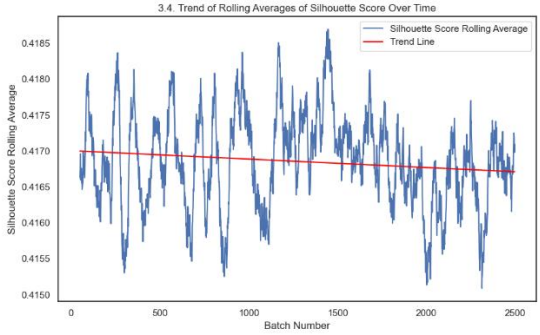


Figure 19: Kmeans Rolling Silhouette Score

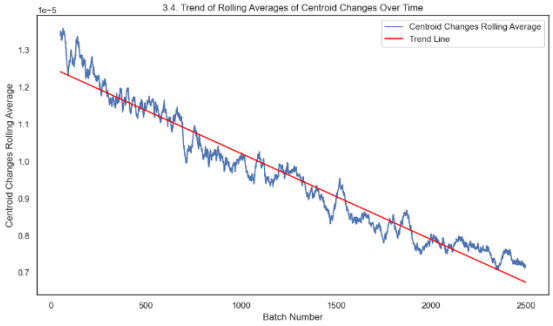


Figure 20: Kmeans Rolling Centroid Changes

Figure 21 presents a flat line for the model size across the 2,500 batches indicating the size of the model remains constant despite processing more data. This consistency in model size is beneficial as it suggests that the complexity of the model does not increase with more data, which is very advantageous with large network traffic. Figure 22 shows an upward trend in model training over the batches which is an issue to consider as the aim would be to have a continuous flow of data and this would be an issue.

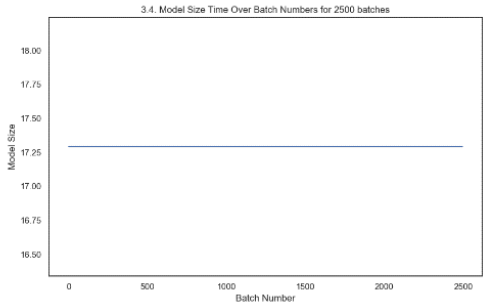


Figure 21: Kmeans Basic Rolling Model Size

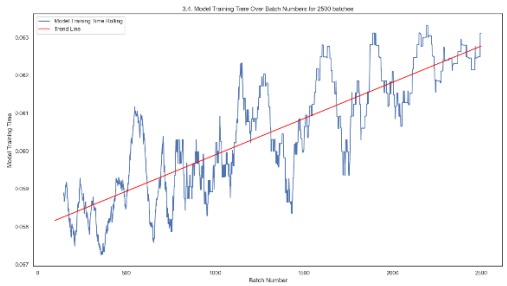


Figure 22: Kmeans Basic Rolling Training Time

4.2.4. Kmeans Best Real-Time Model

In the performance analysis of the Kmeans “Best” model which used 5 clusters, the Silhouette Score in Figure 23 remains consistently in the low 40s but shows a very slight downward trend that needs to be monitored. The Centroid stability in Figure 24 shows the centroids of the clusters have stopped changing and have stabilised completely. This can

indicate convergence, where the centroids now accurately represent the centre of their respective clusters indicating optima clustering has been obtained and no further learning is taking place. However, this also needs to be monitored as real-time network attacks can involve over time and the models may not be responsive to these changes.

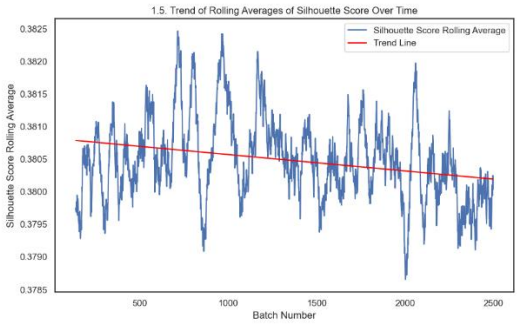


Figure 23: Kmeans Best Rolling Silhouette Score

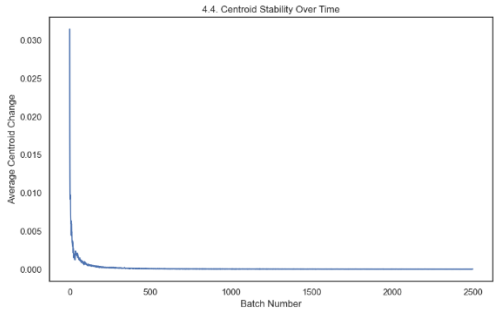


Figure 24:Kmeans Best Rolling Centroid Changes

The model size in Figure 25 remains constant showing good scalability over the 2,500 batches and the model training time in Figure 26 has dropped showing the model is retraining at a faster rate which is a good result.

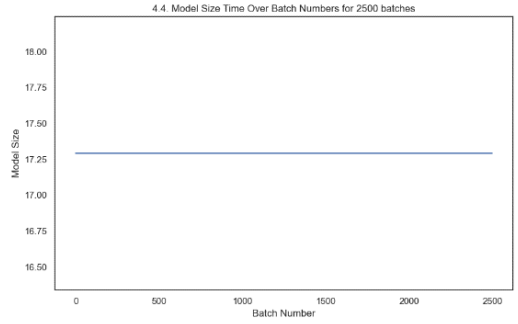


Figure 25: Kmeans Best Rolling Model Size

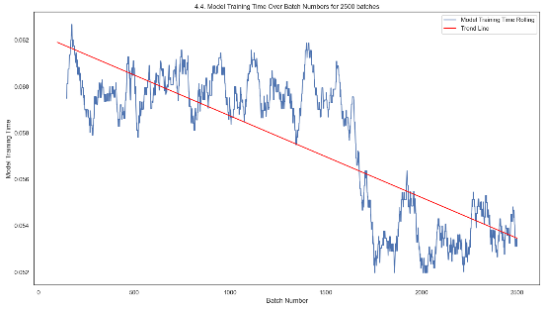


Figure 26: Kmeans Best Rolling Training Time

4.2.5. Isolate Forest Basic and Best Real-Time Models

While examining the Isolate Forest models, both the basic and best real-time models consistently yielded suboptimal results which caused some confusion. A deeper exploration revealed an issue in the real-time retraining of the models with each batch. The model should have been retraining with each batch to create a new updated model, instead each time a fresh model was being retrained with unlabelled and low data, with only 3257 rows of data. This approach was identified as the root cause of the disappointing results.

Given the time to retrain these models, it is not feasible to retrain the Isolate Forest models within the current project timeline.

Exploring the Isolate Forest Results

Isolate Forest models are normally unsupervised models and generally do not use labels. However, since this data did contain the labels, an approach was used to use them to evaluate each batch's performance. The performance metrics indicated some significant issues. Exploring Figures 31-34 reveals a mixed picture of the results, accuracy appears similar to the baseline models in the low 70s range but the precision, recall and f1 scores are all notably low, much lower than the baseline results in the high 40s.

This indicated that the accuracy might be misleading as it does not accurately reflect the model’s ability to distinguish between normal and attack instances. This outcome was unexpected as the Isolated Forest model was expected to do well.

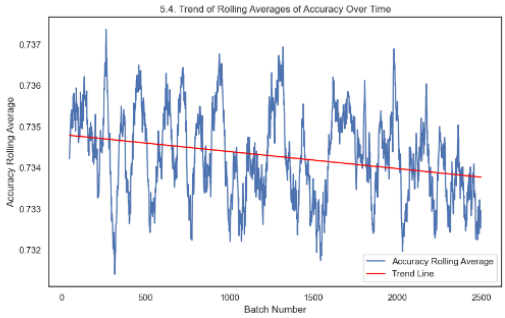


Figure 27: ISO Basic Accuracy

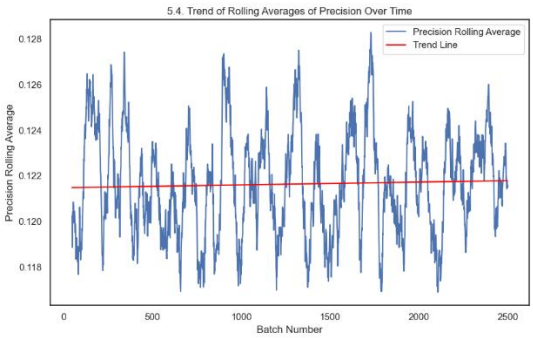


Figure 28: ISO Basic Precision

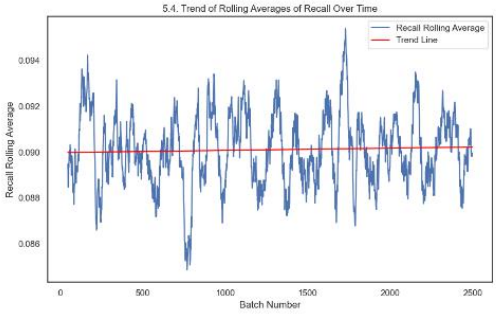


Figure 29: ISO Recall

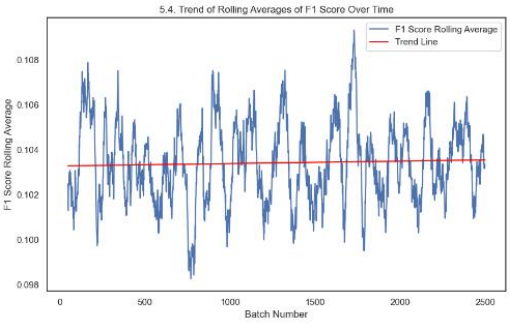


Figure 30: ISO Basic F1

4.2.6. SGD Real-Time Models Explored

Figure 31 shows that the accuracy over the 2,500 batches actually drops, a surprising trend. Despite this, the overall accuracy remains high. The downward trend would need to be monitored but it's the largest drop over the real-time models, albeit a very small amount.

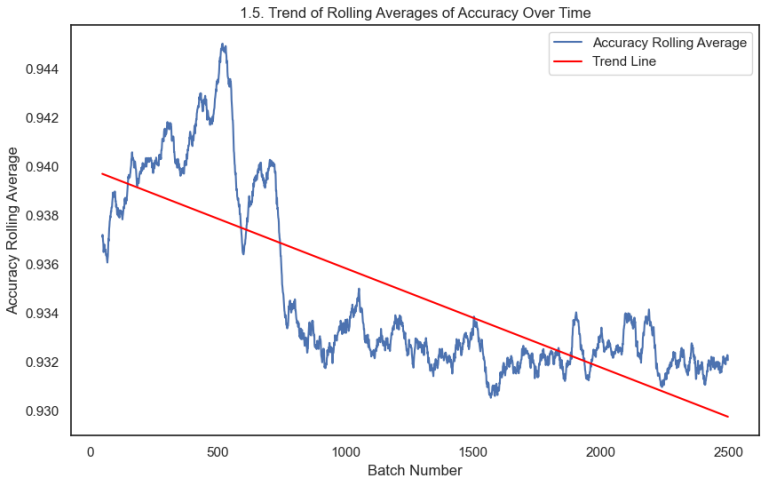


Figure 31: SGD Accuracy

Figure 32 displays the Precision, Recall and F1 scores which also show the downward trend indicating the model's effectiveness in identifying instances is going down over the batches. This could indicate the decision boundary created using the hinge function may not be well placed and might not be adapting to the data in the batches.

Figure 32: SGD Precision, Recall, F1

Figure 33 shows the model's training time which maintains a level trend over the length of the batches but with quite a bit of fluctuations. What this shows is the model training time remains the same as the model's performance is dropping.

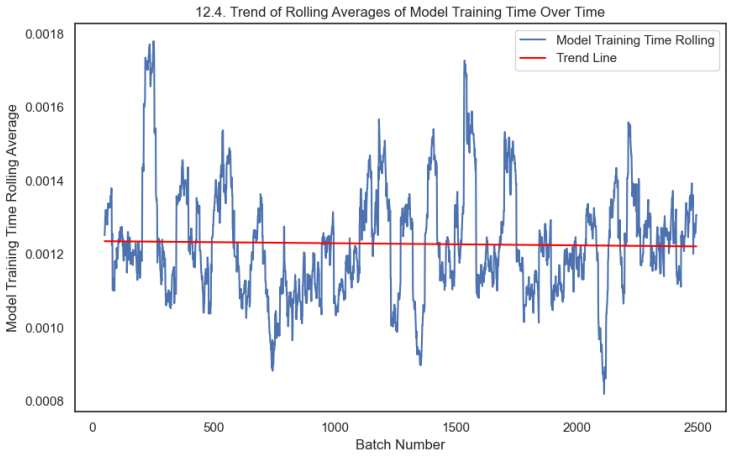


Figure 33: SGD Model Training Times

5. Discussion

5.1 Overview of Findings

5.1.1. Overview Baseline Models

Table 14 shows a comparison table of the results of the different models.

Baseline Models	Training Time (seconds)	File Size	Accuracy/Silhouette Score
Random Forest	4725	548.88Mb	98.98%
Hoeffding Tree	1396	N/A	99.02%
Kmeans Basic Model	44.5696	24.63Mb	0.4261
Kmeans Best Model	100.452	24.93Mb	0.4493
Isolate Forest Basic Model	5.78245	0.61 MB	73.39%
Isolate Forest Best Model	N/A	N/A	N/A
SVM Model 1 – 1% Data	101	3KB	32.60%
SVM Model 2 – 10% Data	1678	30Mb	36.60%
SGD Model	16	4Kb	93.31%

Table 12: Baseline Model Comparisons

Random Forest: The Random Forest model achieved an impressive accuracy of 98.98% but is severely limited by the longest training time at 4725 seconds and the largest model at 548.88Mb. Added to this it can not do incremental learning.

Hoeffding Tree: Improving on the already impressive RF results with an accuracy of 99.02% but also had a more efficient training time of 1396 seconds. Issues with storing the model could cause potential issues but it does have the ability to incremental learn.

Kmeans Models: Both the basic model with 2 clusters and the best model with 5 clusters had Silhouette Scores in the 0.42-0.44 ranges showing moderately good results. Both models have fast training times of 44-100 seconds and small file sizes at 24 MB.

Isolate Forest: Exceptional fast with low file sizes, but plagued by issues for this research. While the Basic model had an accuracy of 73.39%, the precision, recall and f1 scores were much lower in the 40's range.

One Class SVM: While the training time for both models is low, this is only for 1% and 10% of the data. Training of the models becomes exponentially longer and true to form, the SVM models are very computationally intensive, making them unusable for real-time models.

SGD: Stands out for its rapid training of only 16 seconds and minimal file size of 4K which still achieves a high accuracy of 93.31%. This model presents the most promising balance between speed, size and performance.

5.1.2. Real Time Model Comparisons

The Random Forest model, while unable to incrementally learn, still consistently demonstrated high accuracy even when retraining from scratch for each batch. The Random Forest also provided valuable insights into feature importance, contributing to future feature selection. The Hoeffding Tree model however improves on the Random Forest model showing a steadily improving performance but also gave smaller and faster models.

Both the Kmeans models showed consistent clustering performance in the low 40s. The basic models' performance dropped slightly over the batches and the training time increased, both not desirable results which could cause issues later for the continuous flow of network traffic. The Kmeans best model also has a small downward trend in performance with a larger drop in training time, however, the centroids reached convergence which could be an issue for new types of attack instances.

The Isolate Forest models faced challenges and further investigation and refinement are needed. While they do not natively provide incremental learning, they have the potential and their very impressive training speeds and small model size merit additional research.

Lastly, the SGD model trained very quickly with a low file size but showed a downward slight trend in model performance which might indicate the model's decision boundary might need adjustment. With adjustments, it could be a viable option for real-time monitoring systems.

5.2. Interpretation of Results

The interpretation of the various ML models tested in this research provides critical insights into their respective capabilities and limitations within the context of identifying attack instances in network traffic. Supervised models, especially decision tree-based models performed with superior results but this is highly likely due to their ability to learn from the labelled data. In the real world, labelled network data is unlikely to be available, making the direct application of supervised models less feasible. The next phase of this research will involve developing a simulated network where there are no labels and unsupervised models will be needed which do not rely on labelled data, a more realistic approach.

With Random Forest, its strong accuracy across all the batches highlights it as a strong model. However, the absence of incremental learning presents a challenge for real-time data as the model requires to complete retraining with each new batch, this makes it unusable. However, it can still offer insights, especially its ability to rank feature importance.

The Hoeffding Tree model does exhibit incrementally learning, which allows it to adjust and improve over time. An approach that is more aligned with a real-world system. Its performance with quicker training times and smaller model sizes makes it one of the stand-out models in this research.

The Kmeans models did not perform as well as hoped but in the next phase when there is no labelled data I believe they will be of more use than the supervised models. The tendency for both models to slightly decrease in performance over time would need to be monitored for true real-time data, as well as their tendency to reach convergence.

The Isolated models, despite their lower training speeds and smaller model sizes, did not perform as expected. While I was disappointed in their results for this research, their fast training speed and low model size and ability to be adapted for incremental learning still makes then an interesting model that will be taken to the next phase.

The SGD model is another stand-out model I was impressed with, its rapid training and small size model size make it the almost perfect model to balance speed, size and performance. The slight downward trend on performance needs to be investigated with appropriate adjustments.

5.3. Reflection on Previous Research

The field of anomaly detection has been extensively explored, with a diverse range of approaches and methodologies. Below is a comparison with points of overlap and improvements:

Performance Metrics: Consistent with the previous studies of [13], and [24], this research resulted in similar high-performance results of ML models in the context of cybersecurity.

However, this research goes a step further by introducing incremental learning, allowing the models to adapt to new data dynamically showing small but promising improvements.

Scalability and Real-Time Processing: Using Apache Kafka for real-time data streaming batches sets our work apart from earlier studies like [35] that relied on static model training. This research more importantly mirrors the real-world operational requirements of modern IDS systems, thereby providing practical insights into scalability and model results.

Contribution of the Field: Overall, this research contributes to the current body of knowledge by applying a practical real-world approach to anomaly detection using the CSE-CIC-IDS2018 dataset. Additionally, the comparative analysis of the different models further advances real-time anomaly detection and offers a new baseline for future research in the field of network anomaly detection.

5.4. Data and Methodological Reflection

Data Choice: While the CSE-CIC-IDS2018 was large it had its limitations that meant the overcall research had to look at single attack instances rather than trying to look over multiple batches to find a single attack cluster.

Data Preprocessing: In this research, the data preprocessing was executed before the application of the baseline models and real-time approaches. This was a strategic decision to streamline the workflow and manage the research's extensive data volume efficiently. While this simplifies the process for research a real-world scenario would need to use the preprocessing steps to adapt the model to improve on the results and also need to adapt to continuously evolving real-world changes in data.

Feature Selection: Although the CSE-CIC-IDS2018 itself had some feature selection included in the original data and a limited initial feature reduction was conducted which combined reduced the feature set from 400 to 71, as explored in the above sections there is room for a more rigorous feature selection process. Future work should include a detailed analysis of feature relevance and how this affects model performance.

Real-Time Streaming Architecture: Apache Kafka was selected after evaluating several alternatives, including Apache Spark and Amazon Sagemaker. Further investigation is required to explore these technologies' impact on batch size, number of batches, loading time, scalability and costs.

5.5. Limitations of Research

Limited Computer Resources

Faced with limited computing power reduced the ability of the research to explore additional ideas or retrain the models to improve their results. These constraints significantly impacted our ability to experiment with various model architectures and hyperparameter settings.

Streaming Data Challenges

Streaming data offered considerable challenges in our research but meticulous experimentation using 5 batches allowed experimentation with batch numbers, batch sizes, object types and compression types. For this research, the focus was on identifying attack instances but in the future, the focus will change to identifying attacks that come in large clusters. One attack had a cluster of over 50,000 instances, and as this research had batches with 3257 rows it needs further development on how an attack cluster of this size can be trained.

Model Selection and Comparison

The project focused on comparing various ML models with six models chosen for this research. As each model was different, comparing their results was an issue.

Real-Time Data Processing

The requirement to process data in real time introduces additional complexities. This not only involved technical challenges related to the data pipeline and model updating but also raised questions about the scalability of the system under increasing load conditions.

Data for Anomaly Detection Projects

The lack of realistic network data was already been mentioned and presents a significant challenge for research in this domain. Due to this limitation, compromises are necessary when research directions where chosen. For instance, the initial concept of designing an alarm-based anomaly detection system to identify attack clusters was adapted to instead detect individual attack instances due to a lack of available data. However, this also offers an exciting direction where a custom network is created and will be explored in section 5.6 with future research.

Limitations of the CIC-IDS-2018 dataset for this project

The CSE-CIC-IDS2018 dataset itself, at a large 6.4GB, also imposed significant constraints on the direction of the project. Its size presented various challenges, affecting the scalability of the baseline models and the volume of data processed in real-time streaming models. The dataset also impacted the selection of hyperparameters during grid search optimization and also informed decisions on batch training strategies. Additionally,

with 2500 batches for each baseline model, training and retraining each batch became a time-intensive process.

5.6. Future Research

The most significant constraint encountered in the current research is the lack of real-life network traffic. While the CSE-CIC-IDS2018 dataset provided a foundational framework for developing and testing network traffic for cyber security attacks, it imposed certain limitations and boundaries on the research and what could be explored. To address this limitation and advance the scope of our research, the creation of our own simulated network environment is a crucial next step.

This creation of a custom simulated enterprise network will offer a controlled setting for a more robust investigation into network traffic behaviours. This enables the generation of a comprehensive historic traffic profile which will serve as a baseline for further experiments and offer true real-time model creation and retraining. With a tailored enterprise network simulation we can make the research truly real-time, rather than simulated and then introduce a range of attack types and observe their interactions with normal traffic. Multiple models can be run at the same time and outputted results to a real-time dashboard that would provide a richer dataset. This approach greatly improves the emphasis on real-time approaches and removes much of the “simulated approach” that caused issues.

The CSE-CIC-IDS2018 dataset only included numeric data but from creating our own simulated enterprise network, this would allow for a richer set of data types, such as system logs, application logs and network packet payloads. Expanding on more than just numeric

data would enable the development of a more nuanced detection system and the exploration of other ideas that can be used to uncover attack patterns.

Finally, Integrating Large Language Models (LLMs) into the analysis pipeline represents an exciting direction for enhancing threat protection and offering more sophisticated analytical tools to cybersecurity professionals. With the creation of a custom network and the ability to interpret various forms of log data, LLMs can be invaluable in generating comprehensive reports that summarise security incidents and provide actionable insights. Incorporating these models into a real-time dashboard could significantly enhance decision making which allows for quicker and more informed responses to emerging cyber threats.

6. Conclusion

6.1. Reflection on Research Objectives

Data Handing Approach: The methodology outlined in Figure 1 allows for the streaming and processing of data using Apache Kafka to effectively simulate a real-world scenario. This approach I will be using in the next phase as it offers the depth and flexibility needed and once connected to a dashboard will be a step up in terms of visibility.

Model Effectiveness and Adaptability: The various supervised and unsupervised models varied in their ability to accurately detect attack instances, with supervised models showing higher effectiveness. The adaptability of models like Hoeffding Tree, which improved over time, highlights the potential for incremental learning in real-time systems.

Training Efficiency: There was a clear trade-off seen between model complexity and training effectively. Models like One Class SVM are unusable, whereas models like SGD are highly efficient.

6.2. Key Points

- **Apache Kafka:** Performed very well and offers an excellent backbone architecture to evaluate the performance of ML models.
- **Data Limitations:** Comprises with the CSE-CIC-IDS2018 dataset-directed research in directions that take away from a real-world approach. The next phase is to create a simulated network that allows continuous streaming of network traffic, this allows for more exciting opportunities to test other approaches, such as identifying clusters, looking at LLMs to create reports, etc

- **Incremental Learning Models:** The next phase will have continuous streaming network traffic and models like Hoeffding Tree and SGD show considerable promise in this direction.
- **Efficiency V's Accuracy:** The next phase will simulate attacks at different periods and for different cluster lengths, efficiency V's accuracy will become more important. Too many alarms and a security analyst will ignore them, too few and a hacker could be in the network for weeks.

6.3. Practical Application

The significance of this research extends beyond the theoretical and into the practical application of cybersecurity defences. The application of the research findings and the methodologies can be implemented in various industries where network security is paramount. Below are practical applications of the research:

Enhanced Intrusion Detection Systems (IDS): The ML models, which had positive results, once integrated into existing IDS frameworks can improve the detection of cyber threats. The real-time analysis provided by our models can empower IDS systems to respond more dynamically to threats, thereby reducing the window of opportunity for attacks to cause harm.

Real-Time Analytics for Managed Security Service Providers: Managed Security Service Providers can integrate our research outcomes into their service offerings, providing clients with cutting-edge real-time analytics and provide proactive threat-hunting services. This integration can enhance their value but providing not just monitoring but also predictive insights and swift incident response services.

Improvement of Security Posture for Small and Medium Enterprises (SMEs): SMEs often lack the resources to implement comprehensive security solutions. The results developed through this research can be tailored to create cost-effective security solutions for SMEs, allowing them to maintain strong security positions with limited investment in infrastructure resources.

Scalable Security Solutions for IoT Devices: With the Internet of Things (IoT) becoming increasingly popular, our research can be applied to secure vast networks of interconnected devices. The scalability of the Apache Kafka-based framework enables it to handle high throughput and diverse data types generated by IoT ecosystems.

6.4. Personal Journey and Reflection

Throughout this research, I have notably enhanced my experience in developing the bigger picture of combining the different modules of the masters into a more realistic project. The project outline was crucial to this research in developing the project pipeline to manage large datasets, applying machine learning models for both static and real-time ML models, using larger systems like Apache Kafka and combing SQL databases and visualisations. =The domain of anomaly detection has sparked numerous ideas for innovative plans, exploration and methodologies to apply to challenging data as well as different approaches to detecting cyber security attacks.

The project's scale exceeded my initial expectations, with model training times emerging as a primary constraint in exploring new methods. Much time was spent developing dead-end ideas that could take days to run on my limited computer resources. But even this was enjoyable and served as a learning process. In the end, 25,000 retrained

models were developed with at least twice that dumped while perfecting baseline and real-time models. This work has not only reinforced the teachings of my Masters's program but has also deepened my interest in the intersection of Data Science, Cybersecurity, and Anomaly Detection.

6.5. Concluding Remark

In concluding this research, its clear the field of machine learning and real-time detection of cyber security attacks is both complex and interesting. This research underscores the strengths and limitations of various models, especially highlighting the trade off between efficacy and accuracy. While the supervised models demonstrated high accuracy, their application in the real world is limited due to the lack of unlabelled data with real-world network traffic. With unsupervised models, a greater emphasis is needed on model refinement to achieve comparable results to the supervised models.

The experience and challenges encountered while searching for network traffic data and the CSE-CIC-IDS2018 dataset have sparked a plan to develop a simulated enterprise network. This will allow a deeper exploration of machine learning approaches that closely mirrors real world traffic. With this simulated traffic there is more opportunity to explore logs especially and this would allow of the exploration of using Large Language Model (LLMs), which offer an interesting new direction.

References

Reference style is IEEE.

[1] "The Latest Cyber Crime Statistics (updated May 2023) | AAG IT Support." Accessed: May 09, 2023. [Online]. Available: https://aag-it.com/the-latest-cyber-crime-statistics/

[2] "How the HSE cyber attack changed the face of online crime globally," The Irish Times. Accessed: Mar. 17, 2023. [Online]. Available: https://www.irishtimes.com/technology/data-security/2023/03/11/how-the-hse-cyber-attack-changed-the-face-of-online-crime-globally/

[3] "Hackers threaten to publish 'confidential' MTU data unless ransom is paid, High Court told," The Irish Times. Accessed: Mar. 17, 2023. [Online]. Available: https://www.irishtimes.com/ireland/education/2023/02/11/hackers-threaten-to-publish-confidential-mtu-data-unless-ransom-is-paid-high-court-told/

[4] "How reputational damage from a data breach affects consumer perception | Imprivata." Accessed: May 09, 2023. [Online]. Available: https://www.imprivata.com/blog/reputation-risks-how-cyberattacks-affect-consumer-perception

[5] S. Romanosky, R. Telang, and A. Acquisti, "Do Data Breach Disclosure Laws Reduce Identity Theft?," *J. Policy Anal. Manage.*, vol. 30, no. 2, pp. 256–286, 2011.

[6] M. Pawlicki, R. Kozik, and M. Choraś, "A survey on neural networks for (cyber-) security and (cyber-) security of neural networks," *Neurocomputing*, vol. 500, pp. 1075–1087, Aug. 2022, doi: 10.1016/j.neucom.2022.06.002.

[7] "CSE-CIC-IDS2018 dataset." [Online]. Available: https://www.unb.ca/cic/datasets/ids-2018.html

[8] T. McNally, "HSE hackers were in health service's computer system for eight weeks before cyber attack," TheJournal.ie. Accessed: May 09, 2023. [Online]. Available: https://www.thejournal.ie/hse-hack-report-5626054-Dec2021/

[9] "Apache Kafka." [Online]. Available: https://kafka.apache.org/

[10] Amrit Pal Singh, "Analysis of Host-Based and Network-Based Intrusion Detection System".

[11] A. C. Solutions, "Why Next Generation IDS Systems are Flawed." Accessed: May 09, 2023. [Online]. Available: https://blog.ariacybersecurity.com/blog/why-next-generation-ids-systems-are-flawed

[12] S. M. Othman, F. M. Ba-Alwi, N. T. Alsohybe, and A. Y. Al-Hashida, "Intrusion detection model using machine learning algorithm on Big Data environment," *J. Big Data*, vol. 5, no. 1, p. 34, Sep. 2018, doi: 10.1186/s40537-018-0145-4.

[13] N. Farnaaz and M. A. Jabbar, "Random Forest Modeling for Network Intrusion Detection System," *Procedia Comput. Sci.*, vol. 89, pp. 213–217, Jan. 2016, doi: 10.1016/j.procs.2016.06.047.

[14] P. Verma, S. Anwar, S. Khan, and S. B. Mane, "Network Intrusion Detection Using Clustering and Gradient Boosting," in *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Jul. 2018, pp. 1–7. doi: 10.1109/ICCCNT.2018.8494186.

[15] D. S. Kim and J. S. Park, "Network-Based Intrusion Detection with Support Vector Machines," in *Information Networking*, H.-K. Kahng, Ed., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pp. 747–756. doi: 10.1007/978-3-540-45235-5_73.

[16] Y. Liao and V. R. Vemuri, "Use of K-Nearest Neighbor classifier for intrusion detection11An earlier version of this paper is to appear in the Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, August 2002," *Comput. Secur.*, vol. 21, no. 5, pp. 439–448, Oct. 2002, doi: 10.1016/S0167-4048(02)00514-X.

[17] "UCI Machine Learning Repository: KDD Cup 1999 Data Data Set." Accessed: May 10, 2023. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/kdd+cup+1999+data

[18] “NSL-KDD | Datasets | Research | Canadian Institute for Cybersecurity | UNB.” Accessed: May 10, 2023. [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html
[19] “(PDF) A Survey of CNN-Based Network Intrusion Detection,” *ResearchGate*, doi: 10.3390/app12168162.
[20] F. Laghrissi, S. Douzi, K. Douzi, and B. Hssina, “Intrusion detection systems using long short-term memory (LSTM),” *J. Big Data*, vol. 8, no. 1, p. 65, May 2021, doi: 10.1186/s40537-021-00448-4.
[21] A. Singh and J. Jang-Jaccard, “Autoencoder-based Unsupervised Intrusion Detection using Multi-Scale Convolutional Recurrent Networks.” arXiv, Apr. 07, 2022. Accessed: May 09, 2023. [Online]. Available: http://arxiv.org/abs/2204.03779
[22] “Intrusion Detection Using Principal Component Analysis and Support Vector Machines | IEEE Conference Publication | IEEE Xplore.” Accessed: May 10, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9264568
[23] F. Amiri, M. Rezaei Yousefi, C. Lucas, A. Shakery, and N. Yazdani, “Mutual information-based feature selection for intrusion detection systems,” *J Netw. Comput. Appl.*, vol. 34, pp. 1184–1199, Jul. 2011, doi: 10.1016/j.jnca.2011.01.002.
[24] “Increasing the Performance of MachineLearning-Based IDSs on an Imbalancedand Up-to-Date Dataset”, [Online]. Available: https://www.researchgate.net/publication/339194281_Increasing_the_Performance_of_Machine_Learning-Based_IDSs_on_an_Imbalanced_and_Up-to-Date_Dataset
[25] J. L. Leevy and J. Hancock, “Detecting Cybersecurity Attacks Using Different Network Features with LightGBM and XGBoost Learners”, [Online]. Available: https://ieeexplore.ieee.org/document/9319392
[26] M. A. IIyas, “Machine learning approaches to network intrusion detection for contemporary internet traffic”, [Online]. Available: https://link.springer.com/article/10.1007/s00607-021-01050-5
[27] L. D’hooge, T. Wauters, B. Volckaert, and F. De Turck, “Inter-dataset generalization strength of supervised machine learning methods for intrusion detection,” *J. Inf. Secur. Appl.*, vol. 54, p. 102564, Oct. 2020, doi: 10.1016/j.jisa.2020.102564.
[28] V. Kanimozhi and T. P. Jacob, “Artificial Intelligence outflanks all other machine learning classifiers in Network Intrusion Detection System on the realistic cyber dataset CSE-CIC-IDS2018 using cloud computing,” *ICT Express*, vol. 7, no. 3, pp. 366–370, Sep. 2021, doi: 10.1016/j.icte.2020.12.004.
[29] F. S. de Lima Filho, F. A. F. Silveira, A. de Medeiros Brito Junior, G. Vargas-Solar, and L. F. Silveira, “Smart Detection: An Online Approach for DoS/DDoS Attack Detection Using Machine Learning,” *Secur. Commun. Netw.*, vol. 2019, p. e1574749, Oct. 2019, doi: 10.1155/2019/1574749.
[30] S. Bagui and K. Li, “Resampling imbalanced data for network intrusion detection datasets,” *J. Big Data*, vol. 8, no. 1, p. 6, Jan. 2021, doi: 10.1186/s40537-020-00390-x.
[31] “MLlib: Main Guide - Spark 3.4.0 Documentation.” Accessed: May 11, 2023. [Online]. Available: https://spark.apache.org/docs/latest/ml-guide.html
[32] “Sage Maker.” [Online]. Available: https://aws.amazon.com/sagemaker/
[33] “IDS 2018 | Datasets | Research | Canadian Institute for Cybersecurity | UNB.” Accessed: Mar. 17, 2023. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2018.html
[34] “Sklearn Grid Search.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[35] “Machine Learning Techniques for Anomaly-Based Detection System on CSE-CIC-IDS2018 Dataset”, [Online]. Available: https://www.researchgate.net/publication/370384673_Machine_Learning_Techniques_for_Anomaly-Based_Detection_System_on_CSE-CIC-IDS2018_Dataset

Appendix

Apache Producer

```
# Sets the directory where the batches have been stored
avro_folder = folder location

# Loop though for 100 batches
for batch_number in range(100):

   # Construct the file path for the features and labels
    feature_avro_filename = os.path.join(avro_folder, f"features_batch_{batch_number}.avro")
   label_avro_filename = os.path.join(avro_folder, f"label_batch_{batch_number}.avro")

   # Open and read in the feature avro files for the current batch
   with open(feature_avro_filename, "rb") as feature_file:
      features_data = feature_file.read()

   # Open and read in the label avro files for the current batch
   with open(label_avro_filename, "rb") as label_file:
      label_data = label_file.read()

   try:
      # Sent the featues and lables to each topic with a callback report
      producer.produce("batch-network-data", key=str(batch_number), value=features_data,
                       callback=delivery_report)
      producer.produce("test-batch-labels", key=str(batch_number), value=label_data,
                       callback=delivery_report)

      # Wait for the message to be delivered and print batch success
      producer.flush()
      print(f"Flushed Batch {batch_number}. Waiting for callback....")


   except Exception as e:
      # Print error messages if there is an issue.
      print(f"Error sending Batch {batch_number}: {e}")

   # Wait 1/5/10 seconds and send next batch
   time.sleep(10)
```

Apache Consumer

```
################################################################
# Kmeans Basic and Best Consumer Running Model
################################################################
# Set number of batches # CHANGE THIS EACH MODEL
num_batches = 2500

# Initialise MiniBatchKMeans with the parameters from the baseline model
kmeans_model_baseline_basic_current_model =
MiniBatchKMeans(n_clusters=kmeans_model_baseline_basic.n_clusters)

################################################################
# Start Consumer polling
################################################################
# try:
#     while True:
#         msg = consumer.poll(1,0)

#         if msg is None:
#             continue
#         if msg.error():
#             print(f"Consumer error: {msg.error()}")
#             continue

################################################################
# Get batches for features and labels
################################################################

########################
# Set loop to go though batches - Only features needed for kmeans
for i in range(num_batches):

    features_file = os.path.join(folder_path, f"features_batch_{i}.avro")
    print(f"Processing Batch {i}")

    ########################
    # Deserialise the avro files, both features and
    # Load features batch avro using deserialise function
    with open(features_file, "rb") as avro_file:
        avro_bytes = avro_file.read()
        current_features = deserialise_features_avro_record(avro_bytes, features_avro_schema)

        # Convert the current features to a numpy array
        current_feature_array = np.array([list(feature.values()) for feature in current_features])

################################################################
# Update the K-means model with current batch of features
################################################################

    ########################
    # update kemans with partial model which allows for incremental training.
    start_time = time.time() # start of model training
    kemans_model_baseline_basic_current_model.partial_fit(current_feature_array)
    end_time = time.time() # end of model training
```

```
################################################################
# Results for Database
################################################################

   ########################
   # Model Name - Str
   model_name = f"kmeans_model_baseline_basic_batch_{i}"

   ########################
   # Model ID - Str
   model_id = "3.4 - Basic Kmeans"

   ########################
   # Model Description - Str # To add in more detail while training models
   model_description = "Basic kemans model with 2500 batches"

   ########################
   # Batch Number - Int
   batch_number = i

   ########################
   # Number of clusters
   number_of_clusters = 2

   ########################
   # Model Description - Str # To add in more detail while training models
   model_paramaters_prep = {
      "n_clusters": kemans_model_baseline_basic_current_model.n_clusters,
      "init": kemans_model_baseline_basic_current_model.init,
      "max_iter": kemans_model_baseline_basic_current_model.max_iter,
      "tolerance": kemans_model_baseline_basic_current_model.tol,
      "n_init": kemans_model_baseline_basic_current_model.n_init}

   model_parameters = json.dumps(model_paramaters_prep)

   ########################
   # Silhouette Score
   silhouette_score_avg = silhouette_score(current_feature_array,
kemans_model_baseline_basic_current_model.labels_)
   silhouette_score_avg = float(silhouette_score_avg)

   ########################
   # Cluster centers
   cluster_centers = kemans_model_baseline_basic_current_model.cluster_centers_.tolist()
   cluster_centers = json.dumps(cluster_centers)
   cluster_centers

   ########################
   # elbow Method results
   wcss = []
   for i in range(1, 11):  # Example: Test for 1 to 10 clusters
      km = KMeans(n_clusters=i, random_state=42)
      km.fit(current_feature_array)
      wcss.append(km.inertia_)
   elbow_method_results = json.dumps(wcss)
```

```
    ########################
    # Cluster Visualisation
    cluster_labels = kemans_model_baseline_basic_current_model.predict(current_feature_array)
    cluster_visualization = {
        "batch_number": i,
        "data_points": current_feature_array.tolist(),
        "cluster_labels": cluster_labels.tolist(),
        "cluster_centers": kemans_model_baseline_basic_current_model.cluster_centers_.tolist()
        }
    cluster_visualisation_plot = json.dumps(cluster_visualization)


################################################################
# Retrain the model with current batch data.
################################################################

    ########################
    # Additional items to save
    ########################

    ########################
    # New model training time
    model_training_time = end_time - start_time

    ########################
    # Model Size # current model size # Save and overwrite each time to save Disc Space
    # Save current model to folder
    retrained_model_folder_path = os.path.join(retrained_models_folder,
"kemans_baseline_best_search_batch_retrained_model.joblib")
    joblib.dump(kemans_model_baseline_basic_current_model, retrained_model_folder_path,
compress=False)

    # Get size
    model_size_bytes = os.path.getsize(retrained_model_folder_path)
    model_size = model_size_bytes/1024

################################################################
# Second the Data to the Database
################################################################

    # call and save to DB
    insert_model_results_to_db(config, model_name, model_id, batch_number, model_description,
                               model_parameters, number_of_clusters, silhouette_score_avg, cluster_centers,
                               elbow_method_results, cluster_visualisation_plot, model_training_time, model_size)
```