

# Outlier Detection in Virtual Machine Resource Data

A dissertation submitted in partial fulfilment of  
the requirements for the degree of  
BACHELOR OF ENGINEERING in Computer Science  
in  
The Queen's University of Belfast  
by  
Liam Reid  
3<sup>rd</sup> May 2022

**SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER  
SCIENCE**

**CSC3002 – COMPUTER SCIENCE PROJECT**

**Dissertation Cover Sheet**

A signed and completed cover sheet must accompany the submission of the Software Engineering dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name:	Liam Reid	Student Number:	40231992
Project Title:	Outlier Detection in Virtual Machine Resource Data.		
Supervisor:	Dr Son. T. Mai		

**Declaration of Academic Integrity**

Before submitting your dissertation please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

**By submitting your dissertation you declare that you have completed the tutorial on plagiarism at <http://www.qub.ac.uk/cite2write/introduction5.html> and are aware that it is an academic offence to plagiarise. You declare that the submission is your own original work. No part of it has been submitted for any other assignment and you have acknowledged all written and electronic sources used.**

6. If selected as an exemplar, I agree to allow my dissertation to be used as a sample for future students. (Please delete this if you do not agree.)

*Student's signature*

Liam Reid

*Date of submission*

03/05/22

## **Acknowledgements**

I would like to acknowledge my project supervisor, Dr Son. T. Mai, his expertise and knowledge contributed greatly to the success of this project.

I would like to thank all the teaching staff at Queen's University Belfast for all the support during my degree.

## **Abstract**

Humans are growing increasingly dependent on the applications and services deployed on cloud-based platforms. When the cloud fails, organisations risk financial losses, legal implications and damage to their reputation. Cloud services are deployed on Virtual Machines (VMs), where remote resources are used to run applications. If remote resource usage is plotted against time during the failure of a cloud service, an outlier may be observed. An outlier is described as “an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism” [1]. To mitigate the risk of cloud service failure, it is necessary that an effective technique of detecting outliers in VM resource data is proposed.

This dissertation proposes ESMOD, an outlier detection technique and Outlier Detection Dashboard, a web application for monitoring Cloud Resource Usage. The proposed solutions aim to improve the uptime and availability of cloud services by providing Cloud Systems Administrators with the tools to effectively detect outliers in VM resource usage. The dashboard can monitor resource usage in real time and is a useful platform for performing outlier detection experiments. The ESMOD algorithm produces good scores for accuracy, precision, recall and f1 and is effective in making real time predictions in streams of data.

## Contents

Acknowledgements .....	3
Abstract .....	3
Contents .....	4
1.0 Introduction and Problem Area .....	8
1.1 Introduction.....	8
1.2 Cloud Computing in 2022 .....	8
1.3 Risks of Hosting Cloud Resources.....	9
1.4 VM Resource Monitoring.....	10
1.5 Outlier Detection.....	10
Evaluating Outlier Detection.....	11
1.6 Datasets Used.....	11
1.7 Contribution Summary .....	12
ESMOD.....	12
Outlier Detection Dashboard.....	12
1.8 Summary of Experiments .....	12
Comparison of Traditional Outlier Detectors.....	13
CPU Utilization Experiment Summary .....	14
Dengue Fever Rates Experiment Summary .....	14
Review of Outlier Detection Dashboard.....	14
2.0 Solution Description .....	16
2.1 ESMOD .....	16
Ensemble Voting .....	16
Design Motivation .....	16
2.2 Outlier Detection Dashboard .....	16
Real Time Outlier Detection .....	17
Experimental Space .....	17

Storing Data.....	17
Usability .....	18
Robustness.....	18
2.3 System Requirements .....	18
ESMOD Functional Requirements .....	18
ESMOD Non-Functional Requirements .....	19
Outlier Detection Dashboard Functional Requirements .....	19
Outlier Detection Dashboard Non-Functional Requirements .....	20
3.0 Design .....	21
3.1 Design of ESMOD .....	21
Moving Average .....	21
Moving Median .....	21
Moving Boxplot.....	21
Histogram.....	22
Ensemble Voting .....	22
Iterative Learning.....	23
3.2 Database Design.....	24
3.3 Real Time Detection Process Design .....	25
3.4 Experimentation Design .....	26
3.5 Software Architecture Design.....	28
3.6 Detector Evaluation Design .....	28
3.7 User Interface Design.....	29
Real Time Detection UI.....	29
Experimental Space UI .....	32
Unsupervised Detection UI .....	32
Supervised Detection UI .....	33
Ensemble Testing Space UI .....	34

Experimenting Tabs UI.....	34
4.0 Implementation.....	35
4.1 Development Methodology .....	35
4.2 Implementation Language .....	35
4.3 Code Conventions .....	35
4.4 TDD (Test Driven Development) .....	36
4.5 Error Handling .....	36
4.6 Development Environment.....	36
4.7 Database Implementation .....	36
4.8 Server Implementation .....	37
4.9 Web Application Implementation.....	38
4.10 Detection Algorithm Implementation.....	38
4.11 Architecture.....	38
4.12 Deployment to The Cloud.....	40
5.0 Testing.....	41
5.1 Continuous Integration – Automated Test Suite.....	41
5.2 Test Plan for Design and Implementation of the Automated Test Suite.....	41
Introduction .....	41
Scope.....	41
Quality Objective.....	41
Test Methodology.....	42
Test Deliverables .....	43
5.3 Test Environment – Docker.....	43
5.4 Automated Test Suite Results .....	44
6.0 Detector Evaluation .....	45
6.1 ROC Curve.....	45
6.2 Labelled Data - CPU Utilization Outlier Detection Experiment .....	46

Abstract .....	46
Introduction to the Experiment.....	46
Hypothesis .....	46
Methods.....	47
Evaluating Results .....	48
Conclusion.....	49
6.3 Unlabeled Data - Dengue Fever Rate Experiment .....	49
Abstract .....	49
Introduction .....	49
Hypothesis .....	50
Methods.....	50
Evaluation of An Giang Data .....	50
Evaluation of Bac Lieu Data .....	51
Comparison with a Traditional Classification Technique – KNN .....	52
Comparison of Results.....	52
Conclusion.....	53
6.4 Future Work.....	53
6.5 Closing Thoughts .....	54
References .....	55
7.0 Appendices .....	63
7.1 Initial Outlier Detection Results.....	63
7.2 EC2 CPU Experiment Results .....	65
7.3 Dengue Fever Rates Experiment.....	68

## **1.0 Introduction and Problem Area**

### **1.1 Introduction**

Cloud computing, although not fundamentally new, has become one of the fastest growing technologies in the 21<sup>st</sup> century [2]. Applications and services deployed on cloud-based platforms are becoming increasingly complex with the rise in popularity. Consequently, cloud engineers are constantly working to make their platforms robust and failsafe, however problems with cloud services are still prevalent today [3].

On October 4<sup>th</sup>, 2021, Facebook, WhatsApp, and Instagram suffered an outage due to a problem with their shared cloud infrastructure. The effects were felt by its billions of users globally, claiming they felt “discomfort and displeasure” during the 6-hour period [3] [4]. Facebook shares fell 5.4% and losses in revenue were estimated to be around \$99.75 million [5]. Disasters like this one can have huge legal and financial implications for an organisation, therefore it is essential that cloud systems are constantly maintained and monitored.

Maintaining cloud systems within an organisation is the responsibility of the Cloud Systems Administrator. A Cloud Systems Administrator deploys services, monitors and analyses cloud resource performance and resolves any issues reported. This role has become so important in recent years that organisations dedicate entire teams of Cloud Systems Administrators to manage their cloud infrastructure [6].

### **1.2 Cloud Computing in 2022**

Over 90% of organisations use some type of cloud service for a variety of different applications [7]. “Data backup, disaster recovery, email, virtual desktops, software development and testing, big data analytics, and customer-facing web applications” are among those listed by Amazon Web Services (AWS) [8].

In recent years, millions of people around the world have been forced to work from home because of the coronavirus crises [9]. Remote working was forced upon many unprepared organisations and their workforces. Consequently, the use of cloud-based virtualisation technology was utilised to provide staff with the resources necessary to carry out their roles remotely [10]. BYOD (Bring Your Own Device), paired with a VDI (Virtual Desktop Interface), provided employees with an interface to their organisation’s network, applications and resources [11], [12]. Therefore, with the aid of cloud technology, organisations were able to operate as normal through the height of the pandemic.



### 1.3 Risks of Hosting Cloud Resources

With the increased use of cloud technology, competition for resources on shared networks is growing. As such, networks can become overloaded, resulting in increased downtime and a decrease in availability and reliability [13]. This, and many other risks could result in the failure of a cloud service.

Resource exhaustion is a major risk to a VM host server, where software using a server will exhaust its resources and effect the availability of other Virtual Machine's sharing the same resources [14]. Resource exhaustion may occur for several reasons, for example, an infinite loop that prevents a process from terminating or a cyber-attack that results in a server's resources being misused. If availability is impacted, a systems software may fail and introduce vulnerabilities to the system [15]. An effective Outlier Detection algorithm paired with a monitoring dashboard can reduce the risk to availability by notifying the user of a Virtual Machine on a server exhausting more resources than it normally needs.

Physical resources for Virtual Machines are managed by software called the hypervisor. The hypervisor is responsible for provisioning new VMs and distributing the resources where necessary. Hypervisor security can be breached by 'various malicious attacks.' "Session hijacking, man-in-the middle attack, flooding attack (and) Malware-Injection attack(s)" can allow malicious intruders unauthorized access to a server's resources. In session hijacking, a hacker will steal a legitimate user's session ID when it is generated after login. Once the session is hijacked, the hacker can copy the VM and gain access to all the data. A hacker can use a man-in-the-middle attack to intercept data and gain access to a VM. A hacker can perform a Malware Injection attack by inserting malicious code into an application. This makes the resources, applications, and data on the VM vulnerable [16]. These events are liable to happen often and when they do, they can go undetected.

Misuse or unauthorized access of a VM management console can be a serious risk. A hacker could gain access to a VM management console using techniques like phishing, malware attacks, and brute force attacks or by learning login credentials from a system or website breach. An insider from an organisation with authorised access to VM management tools can also be a risk. With access to a VM management console, a person could cause a lot of damage by deprovisioning VMs running important processes. They could also provision additional VMs that are not needed. This would result in a huge cost for the organisation as resources are scaled to meet demand. If resources restrictions are set, availability of resources is decreased for other VMs running important processes [17].

VM sprawl is the process whereby a VM is duplicated then forgotten about. These unmanaged VMs can operate on a network for weeks or even months without being detected. Without proper monitoring, these VMs can miss important security patches. As a result, there is an increased risk to security. There are many methods to mitigate the risk of VM sprawl. For example, implementing processes to govern VM lifecycle management using automated scripts can reduce the risk of VM sprawl, but this comes at a cost [13].

The risks discussed above are not likely to occur day-to-day but when they do, physical resource usage on a VM host server may appear abnormal. If resource usage is plotted against time during one of these events, an outlier may be observed [1]. Taking early action on these outliers could mitigate risk to a host server and prevent important cloud services becoming unavailable. An effective automated outlier detection technique is needed to monitor cloud resource data and alert Cloud Systems Administrators of abnormal behaviour.

#### **1.4 VM Resource Monitoring**

Amazon Web Services (AWS) provides a monitoring and management service for cloud resources called ‘CloudWatch’. This service is used to collect resource data, monitor resources using a dashboard and provide detailed analysis about how a service is operating [18]. The ‘metric collection’ feature is an important tool for Cloud Systems Administrators. Weeks of log data can be aggregated to troubleshoot problems with a cloud service. CloudWatch provides an ‘Anomaly Detection’ feature, where statistical and machine learning algorithms are applied to resource metrics. Using two weeks of data, a machine learning model is trained and used to detect abnormal behaviour [19].

#### **1.5 Outlier Detection**

In addition to detecting abnormal behaviour in cloud resource metrics, outlier detection algorithms are used for datasets in many different fields for a variety of reasons. For example, banks may use outlier detection to detect fraud in abnormal spending patterns, a hospital to detect irregularities with a patient’s heartbeat, or in sports where a team evaluates player performance to determine outstanding attributes [20]. There are many different techniques for detecting outliers with statistical, nearest neighbour, clustering and classification-based detection being among the most popular [21].

There are three main learning methods attributed to outlier detection: supervised, unsupervised and semi-supervised. Supervised techniques depend on datasets where each data point has been labelled either an outlier or an inlier. This data is used to train a model. Unsupervised techniques do not require the data to be labelled, models are trained under the presumption that most of the

data is normal. Semi supervised models are trained using a small portion of data that does not contain outliers. A test dataset is used to evaluate the effectiveness of this detection [22].

### Evaluating Outlier Detection

Different outlier detection techniques can be compared based on their ability to detect outliers. Scores can be generated based on the confusion matrix of the classifications produced by the detection technique. A confusion matrix is defined as “a summary of prediction results on a classification problem” [23]. Visually, a confusion matrix is a table showing the number of false positive, false negative, true positive and true negative classifications made for a given dataset. Using the confusion matrix, an ‘f1’ score can be calculated [24]. The ‘f1’ score is a common method used for comparing detection techniques and is described in the equation below.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

Equation: Calculating F1 Score [25].

Precision and recall are needed to calculate the f1 score, they can be calculated using the following equations.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Equations: Calculating Precision and Recall [25].

## 1.6 Datasets Used

The data provided by AWS’s ‘CloudWatch’ service can be described as time series data. Time series data can be defined as “a sequence of observations ordered chronologically” [22]. There are two types of time series data, ‘Univariate’ or ‘Multivariate’. Cloud resource usage falls under the univariate category, whereby data is considered a sequence of scalar values [26]. Within univariate time series, data can be observed to have two different behaviours, stationary and non-stationary. The mean and variance of stationary data does not change much over time. Conversely, the statistical properties of nonstationary data changes throughout time [27].

Outlier detection in stationary time series data could be solved by manually setting high and low thresholds. More advanced non regressive techniques such as artificial neural networks could be implemented to determine the thresholds automatically [28]. However, regressive techniques are required to detect outliers in cloud resource data because of the phenomenon known as concept drift.

Concept drift occurs when the relationship between data and time gradually changes [29]. With cloud resource usage, this could be due to a larger number of consumers using a cloud service

at a particular time of day or when a new application is deployed to a VM. If thresholds are pre-defined using non regressive techniques, data may be wrongly classified since concept drift is not accounted for. Hence, an iterative approach of determining thresholds is required for outlier detection in cloud resource usage data.

## **1.7 Contribution Summary**

### **ESMOD**

One of the contributions of this dissertation is ESMOD (Ensemble of Statistical Methods for Outlier Detection) which is a technique designed to detect outliers in non-stationary univariate time series data. The ESMOD algorithm comprises of an ensemble of four ‘weak’ outlier detection methods: moving average, moving median, boxplot and histogram [30]. They work individually to get an outlier ‘confidence’ score, then vote on a final classification using the combined confidence values. ESMOD accounts for the idea of concept drift, whereby the relationship between data points and time changes throughout the series [29]. It is designed to make the classification quickly so that real-time outlier detection is possible. This is a key feature to this detection technique as it provides Cloud Systems Administrators the ability to detect discrepancies with performance and quickly diagnose issues with a cloud service.

### **Outlier Detection Dashboard**

In addition to ESMOD, this dissertation proposes ‘Outlier Detection Dashboard’, a web application to aid Cloud Systems Administrators in detecting abnormal behaviour in cloud resource data.

Outlier Detection Dashboard provides a platform to detect discrepancies in cloud resource data using various outlier detection techniques. The detection can run in real time to monitor CPU usage and identify outliers. The solution is also a platform for performing experiments on new outlier detection techniques with a variety of datasets. The proposed solution aids Cloud Systems Administrators in performing their role and automates the process of detecting discrepancies in cloud resource usage.

## **1.8 Summary of Experiments**

To determine the effectiveness of ESMOD, several experiments were performed. One of which compares the detection time, accuracy, precision, recall and f1 score against other outlier detection techniques.

KNN (k-nearest neighbour) is a kernel-based technique which was implemented because it is widely used in pattern recognition and can be used to identify outliers. KNN identifies outliers

by calculating the Euclidean distance (line distance between two points) of a point(x, y) from its nearest neighbour. If the distance calculated is beyond a pre-defined threshold from all neighbours, then the datapoint is classified as an outlier [31].

HBOS (Histogram Based Outlier Score) is a statistical method of detecting outliers. For each feature of a dataset, a histogram is plotted. If the height of a bin in a histogram is significantly lower than the height of the rest of the bins, then the bin is said to be an outlier range [32].

LOF (Local Outlier Factor) was researched and implemented. This method calculates the density of clusters of datapoints by calculating the Euclidean distance between them. Areas of low density are said to be outlier areas, datapoints that lie within these areas are classified as outliers [33].

Isolation Forest was implemented as part of this experiment. This outlier detection technique works by passing a datapoint through an isolation tree. The tree is generated using a test dataset. An outlier score is generated based on the outcome of passing the data through the isolation tree [34].

Finally, SVM (Support Vector Machine) was implemented in the initial research of outlier detection techniques. This technique transforms and separates datapoints so that a straight line plotted through the data will separate them into two categories (outliers and inliers) [35].

### Comparison of Traditional Outlier Detectors

Five datasets were selected to compare the evaluation metrics of the outlier detection techniques. The datasets have been downloaded from and labelled by the Numenta Anomaly Benchmark (NAB). NAB is a platform for testing detection techniques on time series data. It contains 58 labelled datasets used for scoring algorithms [36]. Labelling data is expensive [37], but these datasets are available for free from NAB and are crucial to this research. The datasets selected have varying dimensionality, this will test the versatility of each detection technique. An example of the dataset is shown in table 1.8.1. The average scores produced from these techniques are shown in Table 1.8.2.

Table 1.8.1 Sample from 'speed\_7578' Dataset

Row No.	Timestamp	Data
0	2015-09-08 11:39:00	73
1	2015-09-08 11:44:00	62
2	2015-09-08 11:59:00	66
3	2015-09-08 12:19:00	69
4	2015-09-08 12:24:00	65

Table 1.8.2 Experimental Evaluation Average Metrics Comparison

Detector	Accuracy	Recall	Precision	f1	Time to execute
<b>ESMOD</b>	89.82	81.18	55.96	57.88	12.3498
<b>KNN</b>	89.36	88.06	27.08	40.12	0.7602
<b>SVM</b>	89.46	89.68	31.14	43.96	5.4344
<b>iForest</b>	88.98	65.24	12.38	20.18	1.5459
<b>LOF</b>	89.08	68.88	15.08	23.70	0.8267
<b>Histogram</b>	89.00	0.00	0.00	0.00	1.1356

ESMOD outperformed the traditional classification methods by producing the best f1 score thus making it the most suitable outlier detector for in non-stationary univariate time series data. SVM produced a better score for recall, but its low precision resulted in a weak f1 score. The fastest detector was KNN, taking less than a second on average to complete the detection. ESMOD is designed to detect outliers in streams of data, consequently, it was ten times slower than KNN on average on these large datasets.

### CPU Utilization Experiment Summary

This experiment tests the effectiveness of the newly implemented technique of detecting outliers, ESMOD. The experiment is performed on Cloud platform CPU usage. Detection is evaluated using accuracy, recall, precision and f1. Results show that ESMOD can detect outliers in the CPU usage data with an average f1 score of 0.2446. Occasionally, ESMOD produces nearly perfect scores but does not perform well against unstable data. Full details of this experiment are discussed in Chapter 6.

### Dengue Fever Rates Experiment Summary

The aim of this experiment is to test the effectiveness of ESMOD on unlabelled datasets. The datasets used in this experiments are based on Dengue Fever rates in regions of Vietnam. Observations of generated graphs show that this technique is effective in detecting outliers. Obvious outliers (peaks/troughs) and some subtle outliers (abnormal data for a time of year) can be detected using this method but on rare occasions an outlier is missed and there are many false alarms. A comparison with a traditional classifier proves that ESMOD performs similar to, and often better than KNN.

### Review of Outlier Detection Dashboard

The software proposed as part of this dissertation, Outlier Detection Dashboard, has proved to be effective in detecting outliers in real time streams of data. The dashboard interface, shown

in Fig. 1.8.1, provides many useful metrics that can aid a Cloud Systems Administrator in detecting and troubleshooting abnormal behaviour in VM resources. Additionally, the application can be used to generate accuracy, precision, recall and f1 scores for various outlier detectors and datasets making it a useful experimentation platform for evaluating detection techniques.

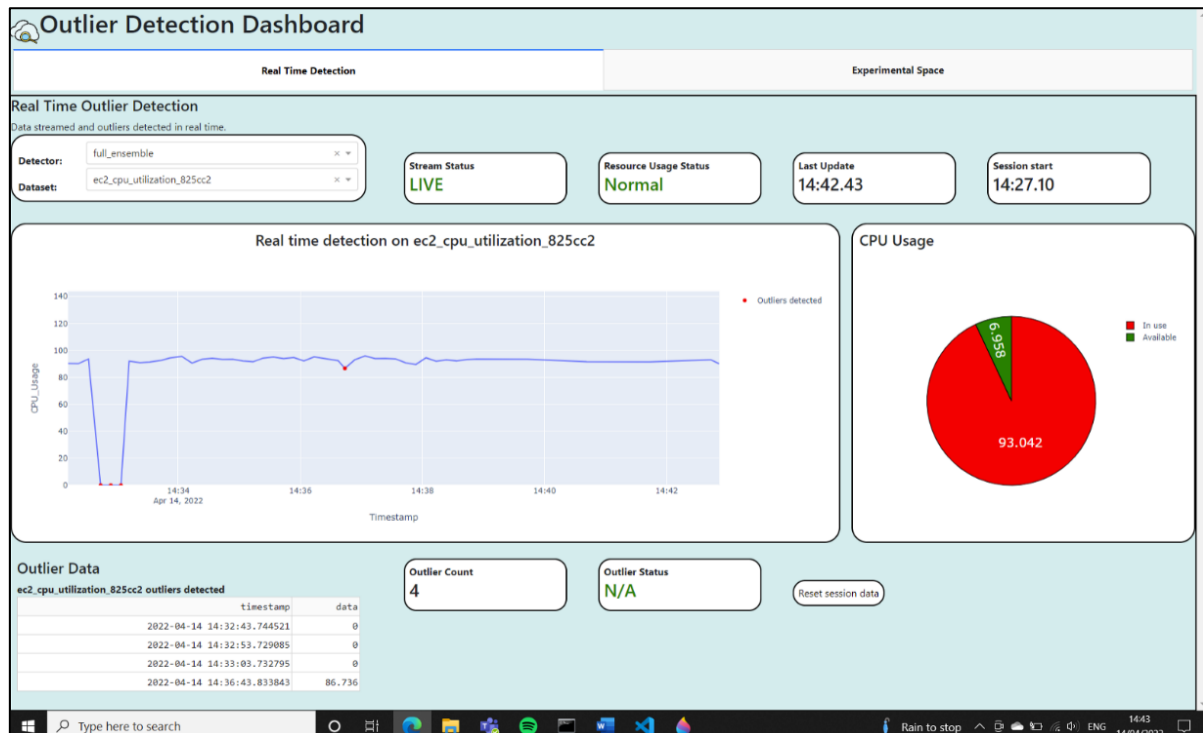


Fig. 1.8.1 Outlier Detection Dashboard.

## 2.0 Solution Description

### 2.1 ESMOD

Chapter 1 introduced ESMOD, an outlier detection technique that is especially effective in detecting outliers in non-stationary univariate time series data. It solves the problem of detecting outliers in VM resource data hence, it is proposed as part of this dissertation. This section discusses the key elements considered when implementing ESMOD and the design motivation.

#### Ensemble Voting

For the detectors to work together they must vote on a classification for a specific data point. Two voting systems are proposed. The first, ‘Majority Classification’, makes a prediction based on the individual classifications made by the detectors in the ensemble. The classification with the most occurrences therefore wins the vote. This method of voting runs quickly, and experiments show it produces accurate results.

The second voting system, ‘Combined Confidence’, extracts a confidence value from each detection made by the detectors and combines them to generate a final confidence. Confidence is calculated from the distance a data point is from the thresholds of each detector. Results show this technique produces good f1 scores, however it is slower than the aforementioned ‘Majority Classification’ voting system.

#### Design Motivation

Chapter 1 discusses AWS’s ‘CloudWatch’ service and the ‘Anomaly Detection’ feature. If two weeks of metric data is not collected, AWS’s detection model is ineffective because it is not possible to train a supervised model without labelled data. Without two weeks of metric data, an unsupervised technique should be implemented to detect outliers. Additionally, training complex supervised detection models requires large, labelled datasets and can take a long time. This would not be effective for real time monitoring of newly created VM resources.

However, ESMOD is an unsupervised technique of detecting outliers, it does not require labelled datasets to work effectively, nor does it take a long time to train a complex model. Due to the efficiency of this technique, resource metrics can be analysed for a newly deployed VMs, and real time outlier detection is possible.

### 2.2 Outlier Detection Dashboard

In addition to ESMOD, Chapter 1 introduced ‘Outlier Detection Dashboard’, a web application to aid Cloud Systems Administrators in detecting abnormal behaviour in cloud resource data. This section specifies the key elements considered when developing this solution.



## Real Time Outlier Detection

Users can visualise the stream of data over time with outliers detected marked by red circles. Data readability is greatly improved by visualising graphs thus, the main component of the real time detection is an animated graph that updates in real time [38]. Additionally, a pie chart depicting the availability of the CPU is displayed to the user. The pie chart is updated in real time and represents CPU usage in a comprehensible format.

## Experimental Space

The ‘Experimental Space’ of this application contains the functionality to perform experiments and generate scores for detectors. Graphs are the key component in representing the detector classifications. Time series data is plotted, true positive classifications are represented by green dots, false positive by red dots and false negative by black. Outlier detection data that has been processed by the application is fed to the scoring system with the true outlier labels and the metrics are calculated. Text containing accuracy, recall, precision and an f1 score, provide a user with all the data they need to evaluate the detection.

There are many options to assist the user in performing detection on a specified dataset. The user can select a detector from a dropdown box containing all detectors implemented in the system. New detectors can be copied to the projects code base and tweaks in the detection script and configuration can allow the user to use a newly implemented detector with the application. The user can select a dataset from a dropdown containing all datasets that are available in the project resources.

If a new dataset needs to be added, it is simply copied to the project resources and the configuration (containing the dataset name) is updated. The dataset is then available for use by the user. The software is highly configurable allowing for many combinations of detectors and datasets. This provides a thorough experimentation platform for examining detectors with multiple different time series datasets.

## Storing Data

Data is stored so that generated detection data can be recalled later. This saves time when executing experiments since some complex detection techniques could take several minutes. A database has been implemented to store the detection data. It has been designed using common practices such as normalisation. Tables within the database do not contain data that can be derived [39]. For example, accuracy, precision, recall and f1 are not stored since they can be derived from the true positive, false negative, false positive and true negative values. There are

many combinations of detectors and datasets, so a large amount of data may need to be stored, therefore the database provides a scalable solution for this application's needs.

### Usability

Usability is described in part 11 of the ISO 9241 as “The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.” [40]. Good UI (User Interface) design is crucial to the success of any client facing software system, so this application contains a UI that is customer focused [41]. The UI implemented with this software system is minimalistic. It contains a limited number of components consisting of dropdown boxes and graphs. For monitoring software, a bombardment of information would take away from the focus of the application. The primary focus is to detect outliers in real time and display relevant data to the user.

### Robustness

To test the robustness of the software and the quality of the code, unit testing has been implemented with this software. The “test early, test often” approach allows new features to be tested against the functionality of the entire system very quickly and very easily [42]. A test suite developed with the aid of a test plan means full coverage of the testable methods in the system has been achieved. The test plan follows a format described by T. Hamilton in an article titled “Test Plan Template: Sample Document with Web Application Example [43]” and can be found in chapter 5. Unit testing has been paired with continuous integration so that an automated test suite runs after every commit.

## 2.3 System Requirements

This chapter introduced ESMOD, the outlier detection technique and ‘Outlier Detection Dashboard’, a web application for detecting outliers in real time and performing experiments. The design and implementation of ESMOD and ‘Outlier Detection Dashboard’ were guided by the system requirements which are detailed below.

### ESMOD Functional Requirements

1	<p><b>When selected as the detector in the ‘Outlier Detection Dashboard’ application, outliers are identified in the data.</b></p> <p>ESMOD will be an available detector option in the detection application. When selected, this method should be used to perform detection on any given dataset.</p>
---	---

## ESMOD Non-Functional Requirements

1	<p><b>ESMOD should produce good detections scores.</b></p> <p>ESMOD should be an effective technique for detecting outliers. This can be verified by using this detector to perform detection on labelled datasets. Using the application, scores can be generated for accuracy, precision, recall and f1. These scores should perform similar to, or better than existing techniques for detecting outliers.</p>
2	<p><b>ESMOD should classify data in real time.</b></p> <p>To benefit Cloud Systems Administrators, ESMOD should be able to classify data as it is read into the system. This means the graph showing the real time stream of data should point out any outliers detected.</p>

## Outlier Detection Dashboard Functional Requirements

1	<p><b>The web application is capable of performing real time outlier detection on a stream of data.</b></p> <p>This section of the application will include all the tools necessary to aid a Cloud Systems Administrator in observing the behaviour of physical cloud resources in real time. When the application is directed to a stream of data, it plots the data against time. The stream is plotted on a graph and any outliers that may be identified are marked in red.</p>
2	<p><b>Real time status of server health and data behaviour should be available to the user.</b></p> <p>As the data is updated in real time, metrics such as server health and data behaviour should be displayed to the user. On each update, the application should check that a connection to the server still exists and that the data is behaving normally (no outliers detected). If abnormal behaviour is detected, or the connection to the server is lost. Then the user should be notified.</p>
3	<p><b>Outliers detected in real time are stored in the database.</b></p> <p>Any outliers detected in the stream of data should be stored in the database. This information is useful for Cloud Systems Administrators who are resolving an issue with a cloud service. The data stored should be displayed alongside the stream of data.</p>

4	<p><b>The web application is a platform for performing outlier detection experiments.</b></p> <p>This section of the application provides the user with multiple detectors and datasets that are combined to produce detection data. When a user selects a detector/dataset combination, the application performs the detection and produces a graph showing the outliers detected.</p>
5	<p><b>Detection data is stored in a database.</b></p> <p>The detection data generated from running an experiment is stored in a database so that it can be recalled later. This will prevent the need to waste time running the same experiment twice. The database follows relevant database design concepts. Entities and attributes are in third normal form.</p>
6	<p><b>Scores are generated for experiments involving labelled data.</b></p> <p>With labelled data, a confusion matrix can be extracted from the predictions a detector makes. The data in a confusion matrix can be used to calculate accuracy, precision, recall and f1. When the user runs detection on labelled data, these metrics are calculated and displayed.</p>

### Outlier Detection Dashboard Non-Functional Requirements

1	<p><b>The User Interface should be minimalistic.</b></p> <p>The dashboard should contain only the minimum amount of functionality required to meet the requirements. This will make the UI intuitive and direct the user's concentration to the detection data produced.</p>
---	--

## 3.0 Design

### 3.1 Design of ESMOD

ESMOD uses the combined predictions of an ensemble of ‘weak’ classifiers to detect outliers. There are 4 classification techniques in the Ensemble.

#### Moving Average

This technique uses the average of the previous data points in the time series to classify the next. After the average is calculated, the standard deviation of the previous data points is calculated. The standard deviation is used as a threshold, if they next data point is less than or greater than the average calculated plus or minus the threshold then the data point is classified as an outlier [44]. Fig. 3.1.1 shows this technique in practice, the red lines represent the boundaries, and the red dots are the outliers detected.

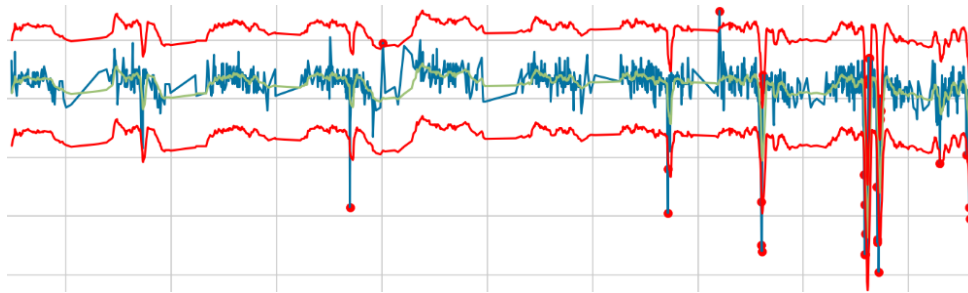


Fig. 3.1.1 Moving Average outlier detection showing boundaries

#### Moving Median

This technique follows the same steps as the previous except a median is calculated instead of the average [45]. Fig. 3.1.2 shows this technique in practice, observe how the boundaries are like moving average but different outliers have been detected.

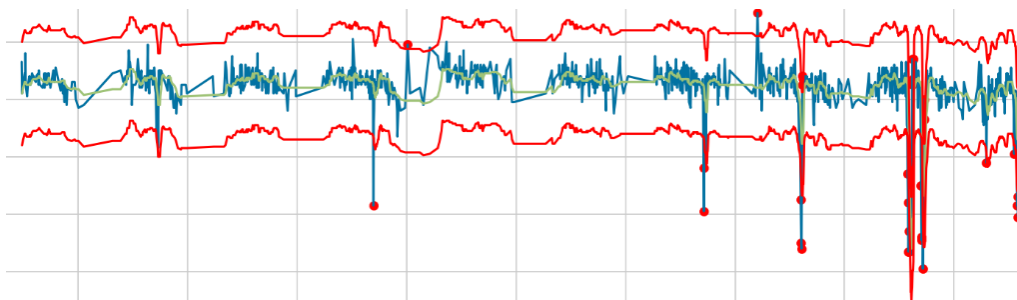


Fig. 3.1.2 Moving Median outlier detection showing boundaries

#### Moving Boxplot

This technique takes several of the previous data points and generates a boxplot. The interquartile range is combined with the upper and lower quartiles to produce a threshold (1.5

\* the inter-quartile range). If the next data point is outside the minimum or maximum thresholds, then the data point is classified as an outlier [46].

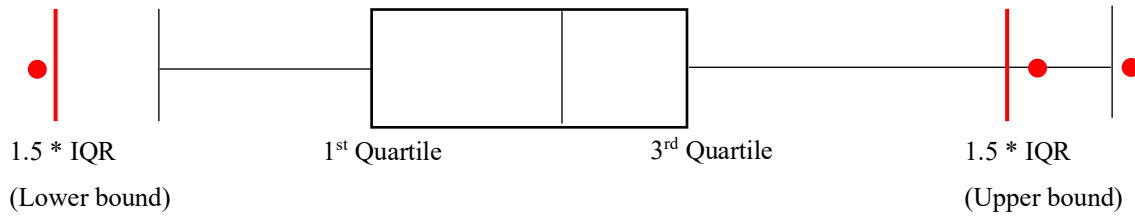


Fig. 3.1.3 Boxplot Outlier Detection Example

## Histogram

This technique plots histograms of subsets of the data. If a range in the histogram has a height less than a defined threshold, then the range is said to contain outliers [28]. If a range has a height below the threshold, but the ranges beside it have a height higher than the threshold then it is considered a borderline inlier.

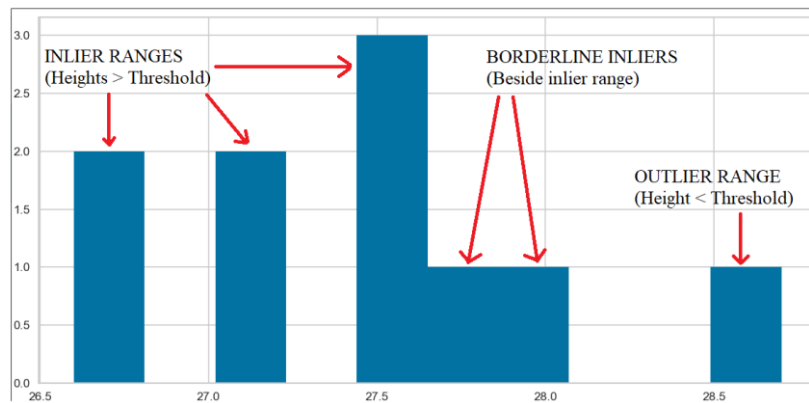


Fig. 3.1.4 Histogram Based Outlier Detection

## Ensemble Voting

The detectors described generate a prediction for a piece of data (outlier or inlier), the predictions are combined to produce a final classification. Two methods of voting are used to produce the final classification. Two voting systems were proposed.

### 1 – Majority Classification

This voting technique labels outliers based on the classification made by the majority of detectors in the ensemble.

$$\text{Outlier if: } \left( \sum_{i=0}^n \text{Detector}_i(\text{prediction}) \right) \geq t$$

Equation: Majority Classification Voting Formula

Where ‘n’ is number of outliers in the ensemble and ‘t’ is the predefined number of detectors that must classify the data as an outlier.

Experiments performed using this technique proved this voting system to be efficient (running detection quickly) and accurate (producing a good score for accuracy) but did not generate good scores for recall, precision and f1 compared to voting system 2.

## 2 – Combined Confidence

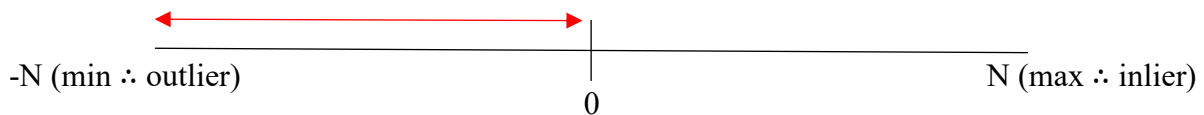
Detectors in the ensemble run individually first, generating a prediction and a ‘confidence’ score. The confidence scores are combined to generate a final prediction. The formula behind this voting mechanism is described below.

Confidence is calculated by the distance between a data point and the threshold.

$$Confidence = \max \left[ \frac{\text{datapoint distance from threshold}}{\text{threshold}}, 1 \right]$$

$$\text{Outlier if: } \left( \sum_{i=0}^n \text{Detector}_i(\text{prediction}) \cdot \text{Detector}_i(\text{confidence}) \right) < 0$$

For predictions, -1(outlier) and 1(inlier), the above equation computes a minimum prediction of -n and a maximum of n. By visualising possible outputs on a spectrum, it can be said that an outlier score < 0 is likely to be an actual outlier.



Experiments proved that this voting system performed the best, generating better scores for precision, recall and f1. Although it takes nearly double the time to complete the detection.

## Iterative Learning

To account for the phenomenon known as ‘Concept Drift’ [47], the detectors in the ensemble use an iterative learning cycle to determine thresholds. This cycle is described by Fig. 3.1.5.

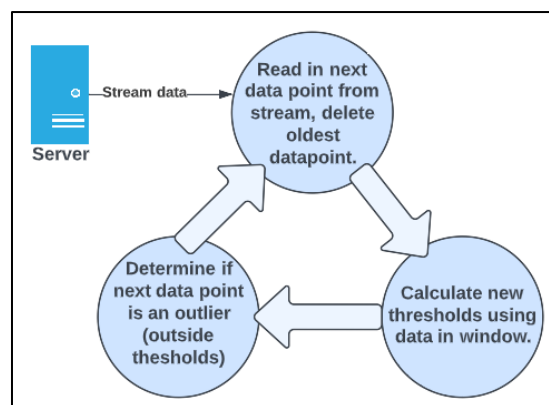


Fig. 3.1.5 Iterative Threshold Finding Cycle.

### 3.2 Database Design

As per the specification, a database has been designed to store the detection data. The database is designed around the concepts discussed in the book “Relational Database Design and Implementation” [48]. Attributes stored in the database were selected based on the requirements of the software system.

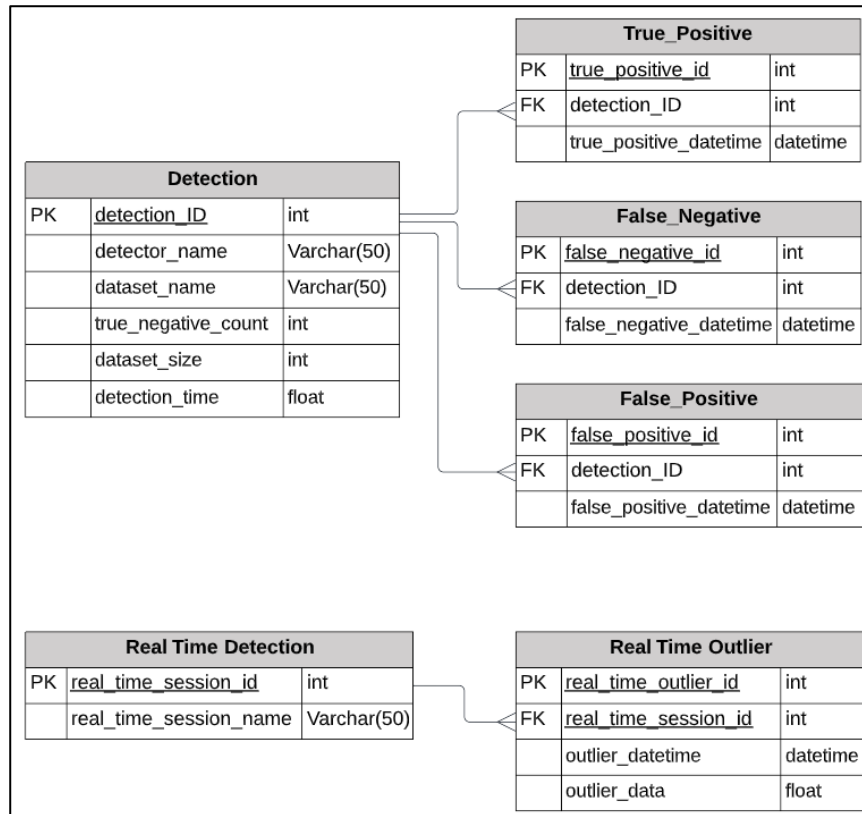


Fig. 3.2.1 Database Entity Relationship Diagram

One of the requirements of the software is to perform and evaluate detection techniques on datasets. Therefore, information about the detector and dataset are stored, along with the data needed to evaluate the detector. To evaluate the detector, values for false positive, false negative and true positive are needed. These values are used to calculate accuracy, precision and f1 score. The dataset size, true positive and true negative count are needed to calculate accuracy. And the detection time is required to compare with other detectors. Accuracy, precision, recall and f1 score are not stored in the database, these values can be derived using the data stored.

Another requirement of the software is to detect and store outlier data in real time. A session name is stored in the database along with the coordinates of the outliers associated with the session.

Normalization was considered as part of the database design, as per the specification. After analysing the requirements and determining what data needs to be stored in the database, functional dependencies were identified among the attributes. This meant properties could be



split out into entities and second normal form was achieved. The data, now in second normal form, was analysed and there were no transitive dependencies. Therefore, third normal form was achieved [39].

Fig. 3.2.1 shows an entity relationship diagram for the database. Entities and their attributes are in third normal form. There are dedicated tables for True Positive, False Negative and False Positive values. The database is designed this way because coordinates are needed to plot the detection classifications on a graph. Since many coordinates may be associated with one detection, these values are stored in separate tables. The true negatives are not labelled on the detection graph, thus only a true negative count needs to be stored (to calculate accuracy).

### 3.3 Real Time Detection Process Design

Amazon's 'CloudWatch' service would be an effective method of streaming real world VM resource usage, Fig. 3.2.2 demonstrates how this would be possible. Unfortunately, due the budget of this project, it is not possible to test the detection algorithms on AWS VM's hosted in the cloud. Multiple VM's running a variety of complex programs to trigger outliers would be required to prove the effectiveness of the detection algorithm.

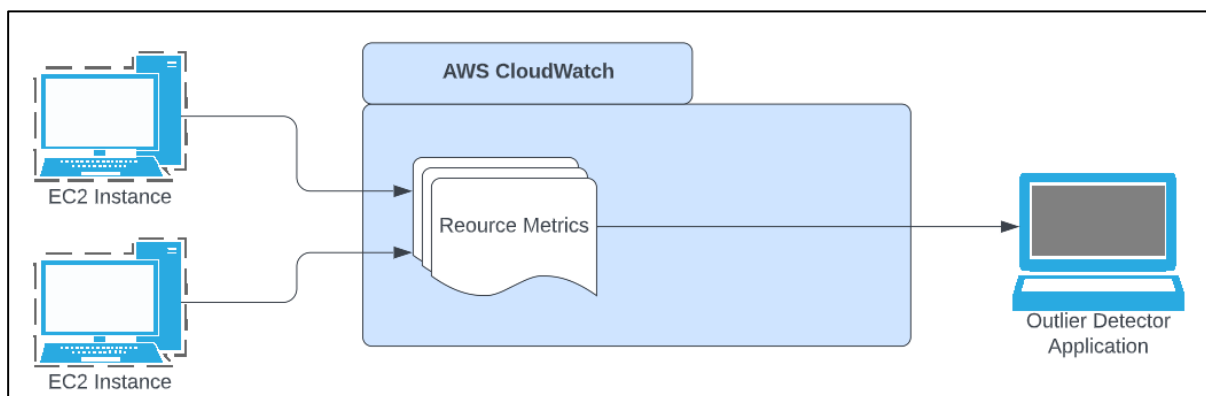


Fig. 3.3.1. VM Resource Streaming Data Pipeline. Based on diagram in “How Amazon CloudWatch works” [49].

An alternative solution has been designed. Data is periodically read in from an external server posting CPU usage and the detection is performed as it is streamed. Multiple VM resource datasets are available to be streamed through the application, this is defined by the user. Outliers detected in the stream of data are stored in the database and a graph of the time series is plotted. Outliers are marked in red on a graph. Storing outlier data in the database makes it available to other parts of the application, meaning data relevant to the detection is available to other components.

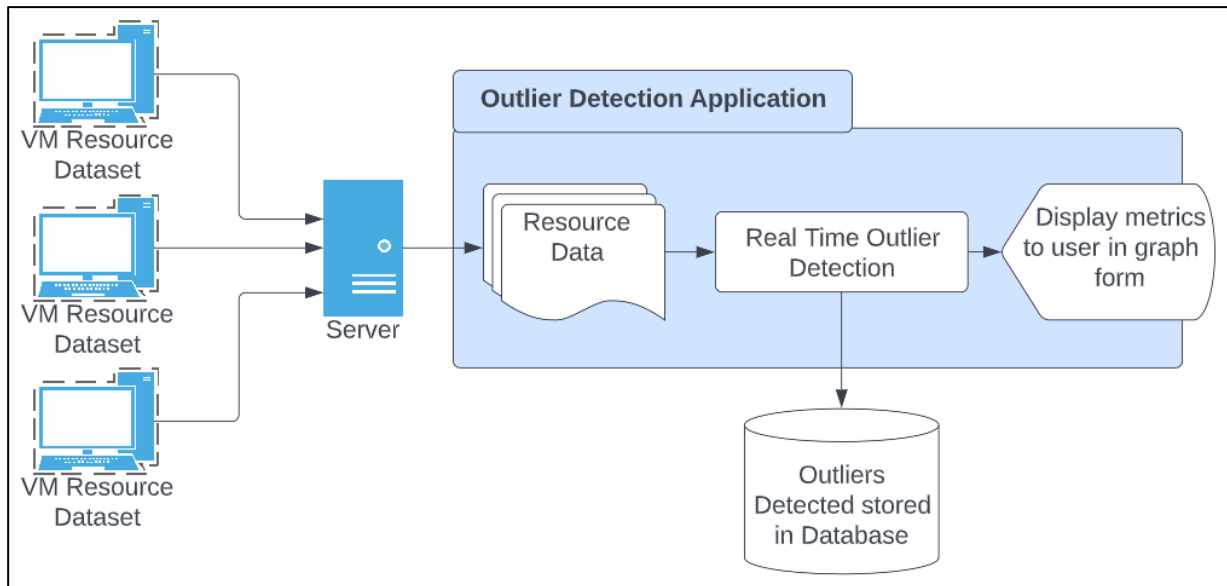


Fig. 3.3.2. Simulated VM Resource Streaming Data Pipeline

### 3.4 Experimentation Design

As part of the specification, a platform to perform experiments has been designed. The implementation allows users to test detection techniques on multiple different datasets. When a user selects a dataset/detector combination, the detection is performed. The detection data is stored in the database so that if the dataset/detector combination is used in the future, the detection data can be loaded from the database and the user does not have to wait for the detection to run. After the detection data is stored, the detector evaluation metrics are calculated (accuracy, precision, recall, f1 and detection time). A graph is generated from the detection data and displayed to the user.

The user can select a detector based on two learning styles, supervised or unsupervised. The supervised learning style requires a model to be trained with labelled datasets. The size of dataset used to train the model influences the performance, so this design gives the user the ability to define the size of the training dataset size [50]. After the dataset is loaded in, the data is prepared. The preparation process involves splitting the data into test and train datasets, the train dataset is used to train the model and the test dataset is used for evaluation. Once the model has been evaluated, the results are displayed to the user. This process is described in Fig. 3.4.1.

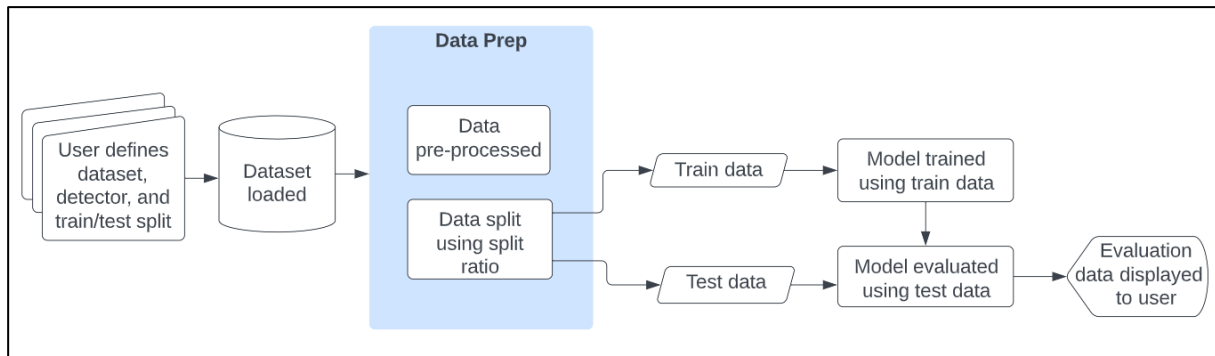


Fig. 3.4.1. Supervised Model Training, Testing and Evaluation Pipeline.

The detection process for experimenting using unsupervised techniques is different from the one described in Fig. 3.4.1. Unsupervised techniques do not require labelled data to perform detection, therefore, detection is possible with unlabelled datasets. However, it is impossible to evaluate detection performed on unlabelled datasets since the true outliers are not known. Consequently, the process of experimenting with unsupervised techniques will only evaluate detection performed on the labelled datasets. The unsupervised detection process is described diagrammatically in Fig 3.4.2.

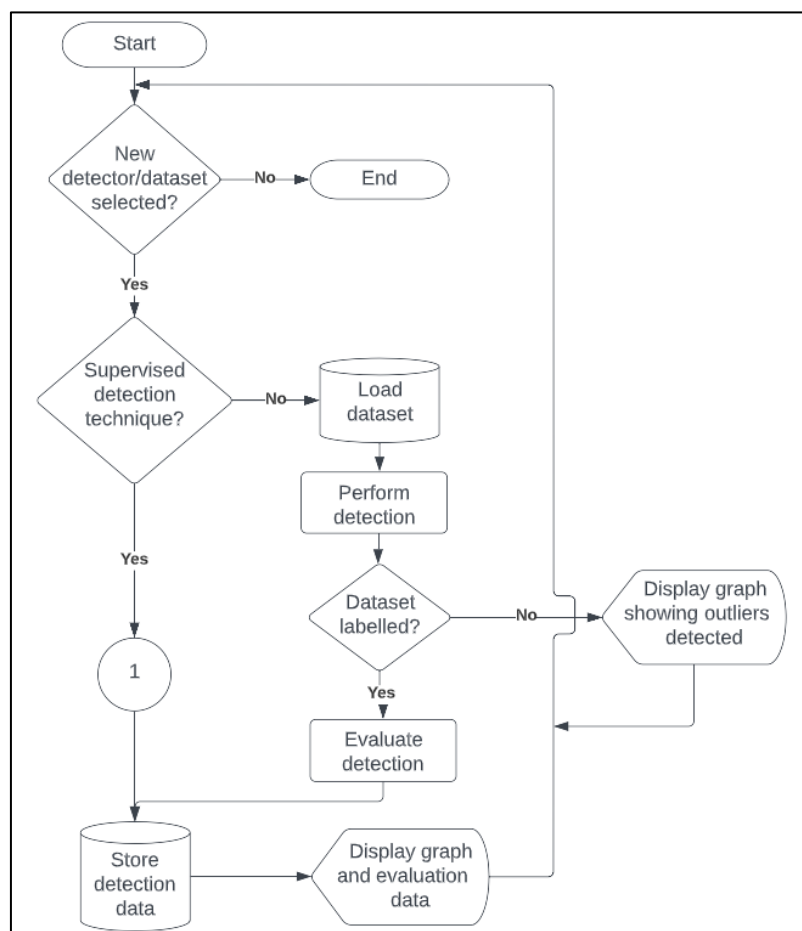


Fig. 3.4.2 Flowchart for Experimental Space of Application (Connector '1' refers to the supervised detection pipeline detailed in Fig. 3.4.1).

### 3.5 Software Architecture Design

The development of software involves collaboration among teams of developers and testers. Code bases can become increasingly difficult to understand if pre-defined conventions are not followed [51]. Hence, clean architecture has been used in the design of the software solution.

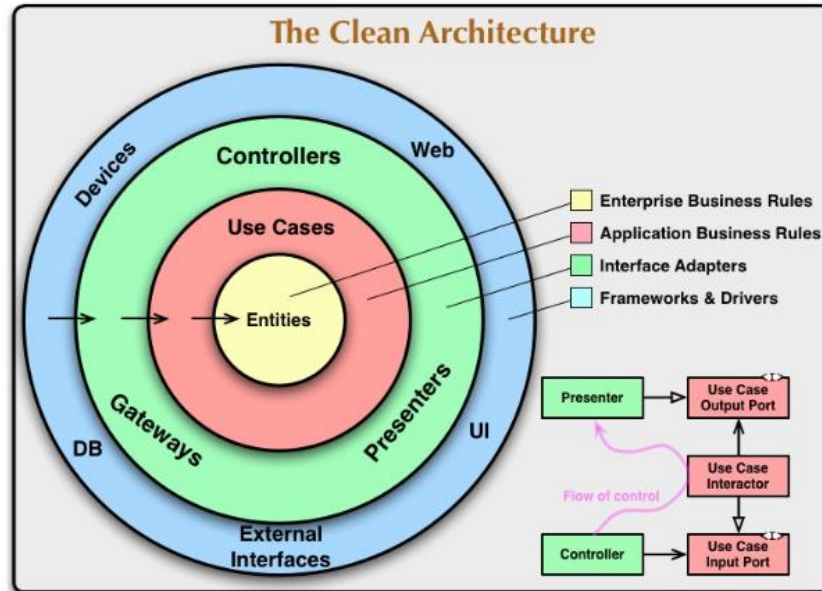


Fig. 3.5.1. The Clean Architecture [52].

Fig. 3.5.1 Describes clean architecture in a diagrammatic form. It consists of a group of labelled concentric circles that represent different components of a software system.

The circle at the centre of the diagram is labelled 'Entities'. At this level, classes are implemented that do not depend on any other component of the system. The detection algorithms have been implemented at this level.

The next circle is labelled 'use-cases'. At this level, classes are implemented that orchestrate the flow of data to and from the entities in the inner most layer. The classes at this layer act as an interface between data and detection.

The next circle is labelled 'Interface Adapters'. Software at this level is used to convert data to different formats to and from the 'use-cases' classes. Methods to plot data on graphs or pass input to use cases are implemented at this level.

The outermost circle is 'Frameworks and Drivers'. At this level, classes concerning the web application are implemented.

### 3.6 Detector Evaluation Design

As per the specification, detection techniques are evaluated based on accuracy, precision, recall and f1 score. Additionally, the detection time is calculated and displayed to the user for evaluation purposes. To calculate the metrics, the number of true positives, false positives, false

negatives and true negatives detected are required. Those values are used in the following equations.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{No.\ Datapoints} \quad Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

Equations used to evaluate the detector against the dataset.

### 3.7 User Interface Design

The requirement of a user is the most important factor when designing a UI (User Interface), therefore, the design is based on the information most relevant to a Cloud Systems Administrator [41].

#### Real Time Detection UI

Fig. 3.7.1 shows the user interface of the real time outlier detection part of this application.

The components have been labelled and the decisions around the design are discussed below.

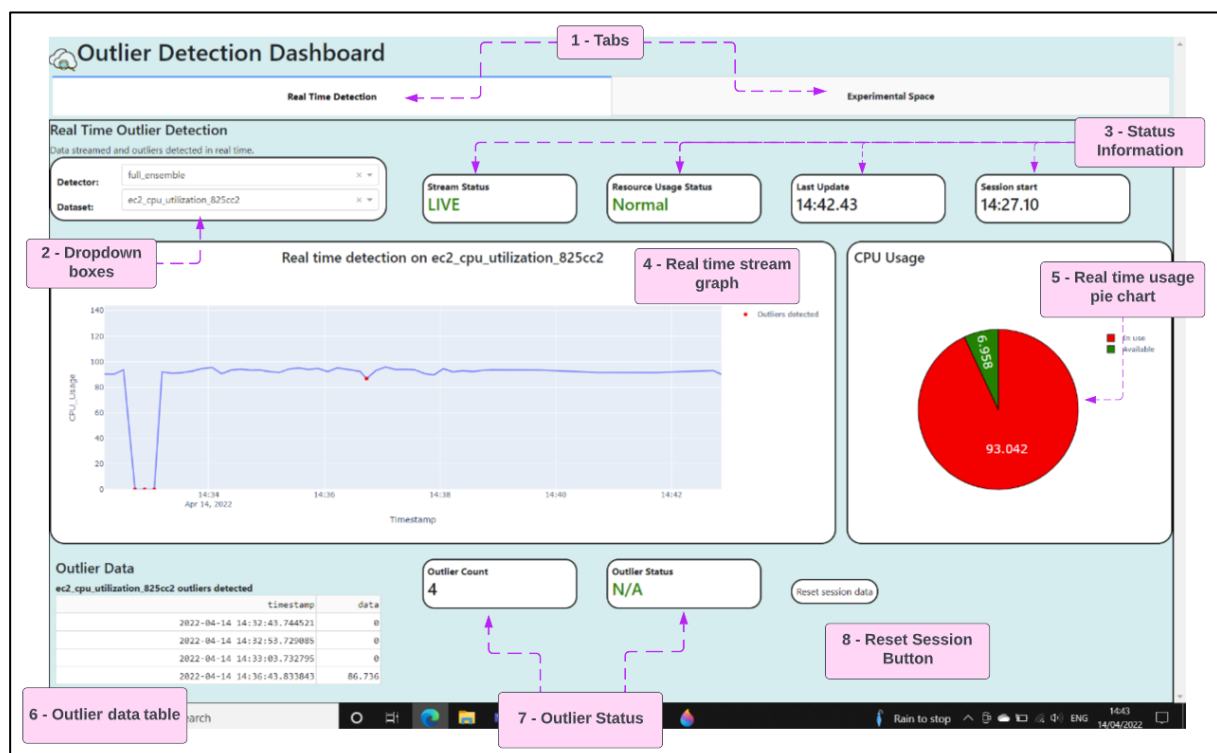


Fig. 3.7.1 Real Time Detection UI

#### 1-Tabs

Tabs have been implemented as part of this design to separate the real time outlier detection part of the system from the experimental space. Separating the experimental components allows the Cloud Systems Administrator to concentrate on the data associated with the resource usage.

## 2-Dropdown Boxes

As part of the requirements, this design allows users to select a detector to perform the detection. Additionally, the user can select which VM resource data to stream into the application. These dropdown boxes are clearly labelled 'Detector' and 'Dataset'. Dropdown box options are auto filled on app start up to aid the user in understanding the application.

## 3-Status Information

This section of the application is designed to display information about the stream of data.

The first box labelled 'Stream Status', informs the user about the server connection. If the application can make a connection to the server the stream status is set to 'LIVE', otherwise it is set to 'DOWN'. Live server status is observed in Fig. 3.7.1. Down server status is shown in Fig. 3.7.2.

The box labelled, 'Resource Usage Status', is designed to alert the user the abnormal behaviour with physical resource usage. If no outliers have been detected recently, the status is set to 'Normal' as seen in Fig. 3.7.1. If outliers have been detected, then the status is set to 'Alert' as seen in Fig 3.7.2.

The next box displays the time that the components in the application were last updated. This has been included as part of the design as it may be useful for debugging issues with components due to an issue with the server.

The last box on the right contains the 'Session Start' time. This information has been included as part of the design so that the user can keep of track of how long the monitoring session has been taking place. This information may be useful for Cloud Systems Administrators in the event that they are tracking the behaviour of a VM during a specific time of the day.

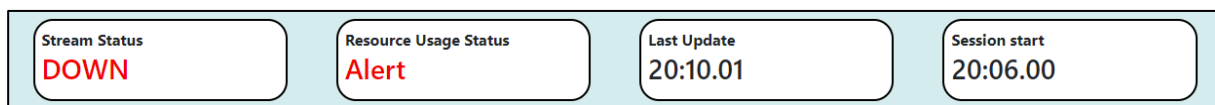


Fig. 3.7.2 Stream Status when Server Fails.

## 4-Real Time Stream Graph

This graph is designed to update in real time, it plots the resource usage and marks any outliers detected. CPU usage is plotted on the Y axis against time, plotted on the X axis. The stream of data over is represented by a blue line. Any outliers detected in the stream are marked by a red dot as observed in Fig. 3.7.3. Additionally, the user can hover their mouse over the data for details about the time and usage (x and y coordinate).

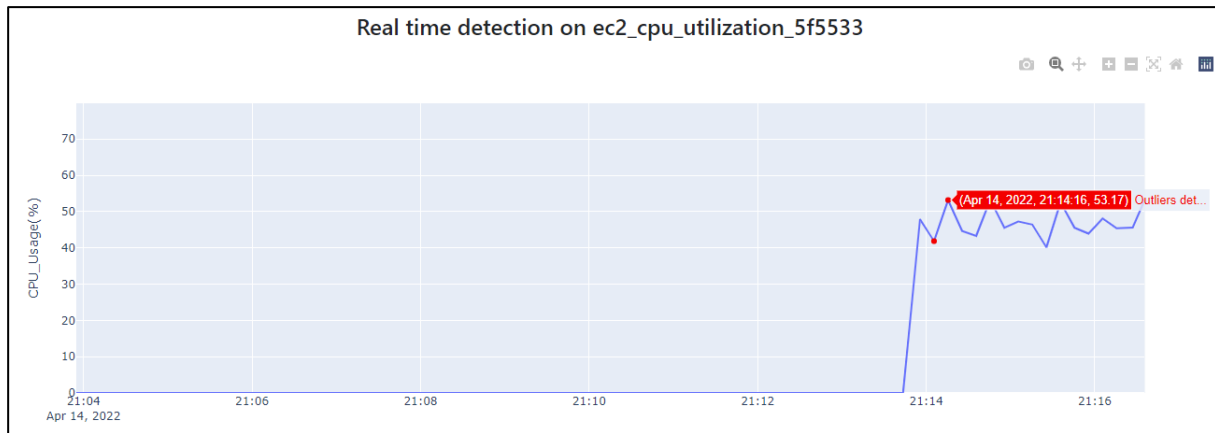


Fig. 3.7.3 Real Time Outlier Detection Graph

### 5- Real Time Usage Pie Chart

A pie chart has been included as part of this design to show the availability of the resource being analysed. It updates in real time to show the portion of the resource in use and the portion that is available. This information is useful to visualise as it can alert a Cloud Systems Administrator of exceptionally high or low CPU usage that the detection algorithm has not flagged. The detection algorithm may not identify this behaviour because of a gradual increase or decrease in usage (concept drift).

### 6- Outlier Data Table

This section is designed to display information about the outliers detected in the current session. When outliers are detected, information is stored in the database. This table represents the data loaded from the database.

### 7- Outlier Status

This section of the UI is designed to display information about outliers detected in the current session. The box labelled 'Outlier Count' displays the number of outliers detected in the session. The box labelled 'Outlier Status' contains information about the most recent outlier detected. The information displayed in this box when an outlier is detected is shown in Fig. 3.7.4.

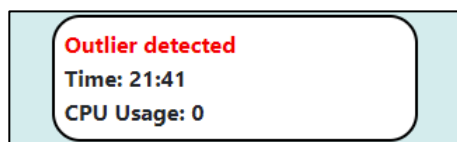


Fig. 3.7.4 Outlier Status when Outlier Detected.

### 8- Reset Session

This button is included as part of the design to allow the user to reset the session. A Cloud Systems Administrator may wish to do this if an issue with a defective VM is resolved, and the new behaviour is considered for outlier detection.

## Experimental Space UI

The ‘Experimental Space’ tab is designed to section off the tools needed to perform experiments with detectors and datasets. The experimental space is split into 5 sections as seen in Fig. 3.7.5.

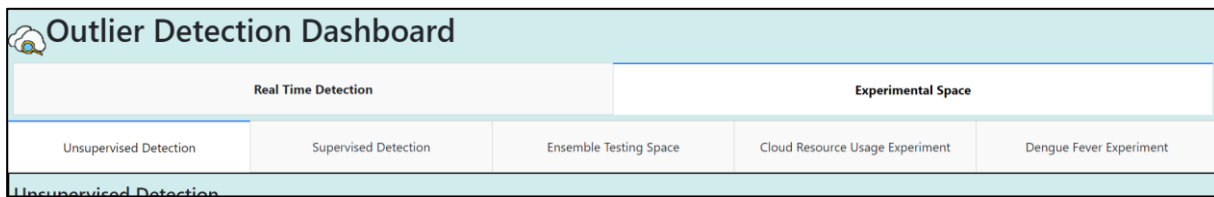


Fig. 3.7.5 Experimental Space Tabs

## Unsupervised Detection UI

The first tab, ‘Unsupervised Detection’, is designed to allow the user to perform experiments using unsupervised techniques. Fig. 3.7.6 shows this interface with labelled components, followed by a discussion of design choices.

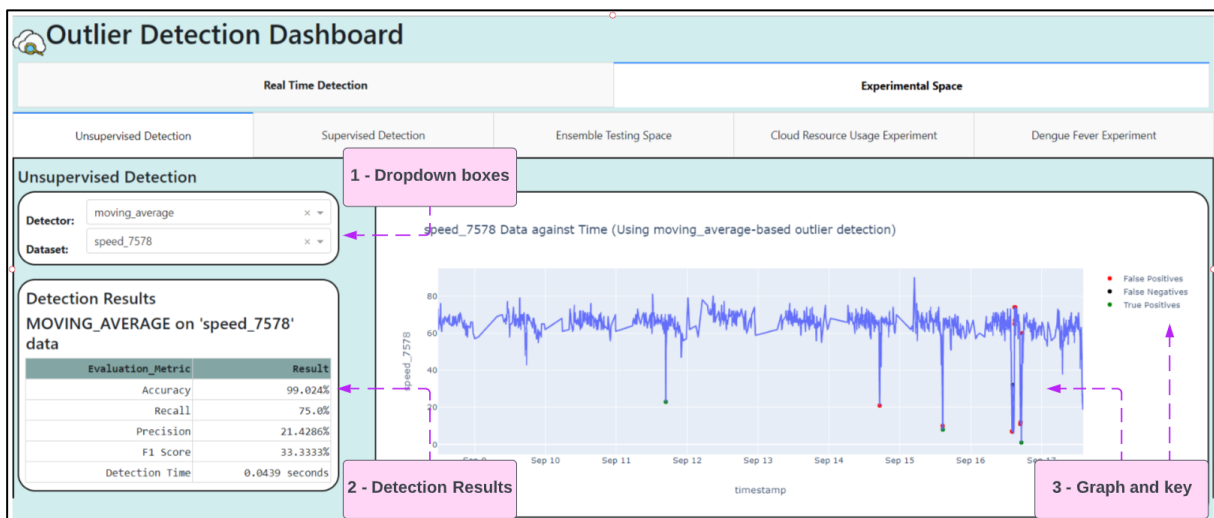


Fig. 3.7.6 Unsupervised Detection UI

### 1- Dropdown Boxes

The user can specify a detector and a dataset, then the detection is performed. The dropdown boxes are filled with lists of available datasets and detectors in the applications configuration. If the user selects a new dataset or detector the detection is performed.

### 2- Detection Results

The datasets available are labelled, therefore the detection results can be evaluated to determine the effectiveness of the detector. A table is included as part of this design to present the evaluation data.

### 3- Graph and Key

A graph showing the time series data along with the classifications is included in the design of this interface. A key instructs the user as to what each classification means. Unmarked data



points are true negatives. True positives are green, false negatives are black and false positives are red. Displaying the data like this can aid the user in determining a weakness with an outlier detection technique.

## Supervised Detection UI

The second tab, 'Supervised Detection', is designed to allow the user to perform experiments using supervised techniques. Fig. 3.7.7 shows this interface with labelled components, followed by a discussion of design choices.



Fig. 3.7.7 Supervised Detection UI

### 1- Text Input

The user can specify a split ratio for the test and train data. This is achieved by typing the desired percentage of the dataset that will be allocated for training. The dropdown boxes in this UI are the same except the unsupervised methods are not available in the detector list.

### 2- Learning Graph

A graph showing the training data is included as part of this design. This data is useful for the experimenter as they can visualise how the clusters of data have affected the models learning. Training data points are blue, green is test data points the detector classifies as inliers and red are outliers.

## Ensemble Testing Space UI

As per the specification, a section has been designed to allow users to test different combinations of detectors working in an ensemble. Fig. 3.7.8 shows this UI with labelled components followed by a discussion of design choices.

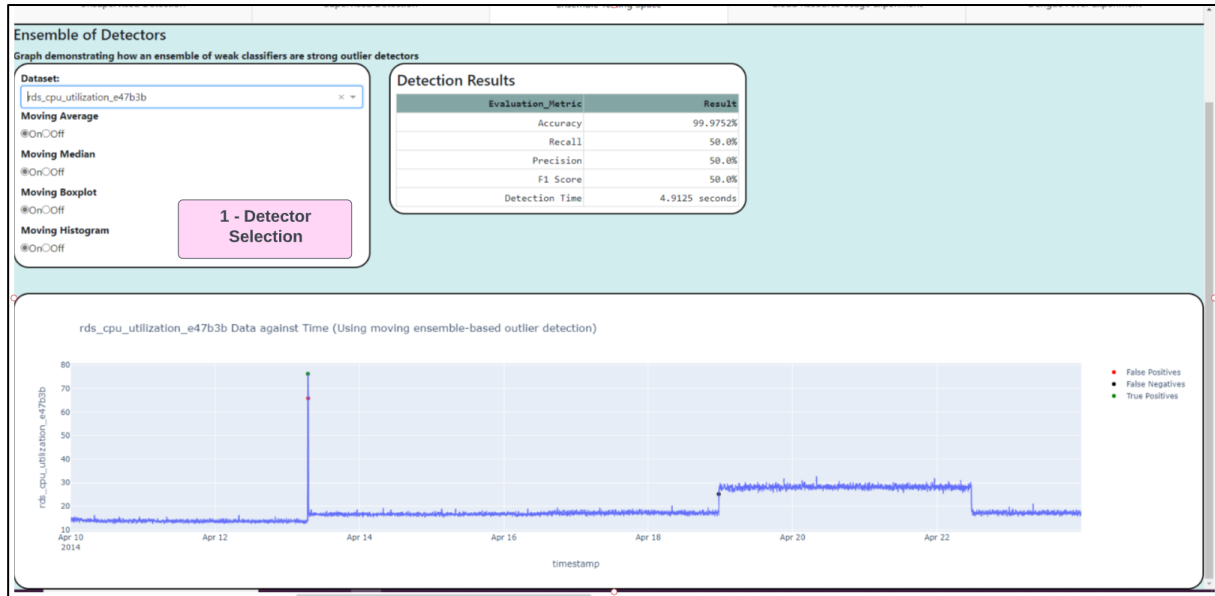


Fig. 3.7.8 Ensemble Testing Space UI

Detector selection is the only new component in this design. Radio buttons are used to ‘switch-off’ detectors in the ensemble to determine which combinations work best for different time series datasets.

## Experimenting Tabs UI

The last two tabs, ‘Cloud Resource Usage Experiment’ and ‘Dengue Fever Experiment’, have been included as part of the design as a containerised section to perform experiments, where only the necessary datasets are available. The experiments detailed in Chapter 6 were executed under these tabs. The designs are similar to Fig. 3.7.6. The design for the Dengue Fever experiment tab deviates from the rest of the system in that there are no detection results. The datasets used in the Dengue Fever experiment are unlabelled, thus scores cannot be determined. The purpose of this experimental is to compare the performance of ESMOD against other outlier detectors on unlabelled datasets.

## 4.0 Implementation

### 4.1 Development Methodology

The software solution has been developed using elements of the ‘Agile’ development methodology. ‘Incremental’ and ‘Iterative’ development practices were used to gradually implement and test features of the application. This allowed an initial working prototype with minimal features to be gradually developed into the final solution. Features implemented in older versions could be revisited and improved [53].

### 4.2 Implementation Language

Python was selected as the implementation language for this software solution because of its versatility. It is commonly used for web-development, data analysis and machine learning. Python “has become a staple in data science”, with many tools making data visualisation quick and effortless [54]. It is used in web development to implement back end functionality and libraries exist to allow secure communications with databases and software. Additionally, it is the most popular computer programming language (as of April 2022) [55], there is a large community of Python developers that constantly generate and update resources and documentation.

### 4.3 Code Conventions

This applications code base follows the ‘Snake Case’ naming convention (referred to in the python style guide as “lower\_case\_with\_underscores”) [56]. All function and variable names are consistent with this practice. Constant variables are written in a similar fashion, with underscores separating words, but they are written with capital letters. None of the variables or functions in the code base are named by a single character, except for counter variables such as ‘i’ or ‘j’ used with ‘while’ loops.

Tabs are used for code blocks that require indentation and, in some cases, multi-line statements are used to shorten the length of code. Shortening the length of long lines of code makes it possible to view several files side by test. This is useful for developing unit test suites around a class.

Docstrings are written as the first statement in all functions and classes. The docstring contains text explaining the purpose of the code [57]. One-line docstrings are used where possible, these contain a one-line definition of the function. When further detail is needed, multi-line docstrings are used. Multi-line docstrings contain details about a function’s arguments, return values, error handling and optional arguments.

#### **4.4 TDD (Test Driven Development)**

TDD guided the implementation of new functionality. Classes and functions are kept short and concentrate on the task they are designed to perform. Unit tests were written in parallel with new code, this ensures code changes that break functionality are detected and fixed early. Extensive test coverage is achieved with the TDD approach, paired with the iterative development practices, large scale refactoring with effective debugging was possible [58].

#### **4.5 Error Handling**

Custom error classes were implemented as part of this applications development. These were required because the built-in exceptions didn't serve the purpose of describing the actual problem with the application. For example, an exception class named 'InvalidPercentageFloatValueError' was developed. This exception is raised when a float value representing a percentage is passed to a function that is less than zero or greater than one. An invalid percentage value would result in data being miscalculated, raising this error prevents that from happening.

#### **4.6 Development Environment**

Visual Studio Code was selected as the development environment. It has features such as automatic code completion, bracket matching, auto-indentation and an interactive debugger. Visual Studio Code has support for Git, source control is possible without leaving the IDE and changes from previous commits are highlighted, making it easy for the developer to modify changes [59]. Additionally, according to a stackoverflow survey, Visual Studio Code is the most popular IDE among professional developers [60].

#### **4.7 Database Implementation**

The software solution required storage for the detection results so that complex, resource intensive processes did not need to run redundantly to generate pre-existing data. The sqlite3 software library is used to implement a server-less, self-contained database [61]. It is "small, fast, fully featured, SQL database" which serves as a perfect solution for the software's storage needs. As part of this implementation, a schema was created using the entity relationship diagram from Chapter 3, Fig. 3.2.1.

```

CREATE TABLE detection(
    detection_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    detector_name text NOT NULL,
    dataset_name text NOT NULL,
    tn_count int,
    data_size int,
    detection_time float
);

CREATE TABLE true_positives(
    tp_pk INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    detection_id int NOT NULL,
    true_positive_datetime datetime NOT NULL,
    FOREIGN KEY(detection_id) REFERENCES detection(detection_id)
);

CREATE TABLE false_positives(
    fp_pk INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    detection_id int NOT NULL,
    false_positive_datetime datetime NOT NULL,
    FOREIGN KEY(detection_id) REFERENCES detection(detection_id)
);

CREATE TABLE false_negatives(
    fn_pk INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    detection_id int NOT NULL,
    false_negative_datetime datetime NOT NULL,
    FOREIGN KEY(detection_id) REFERENCES detection(detection_id)
);

CREATE TABLE real_time_detection(
    real_time_session_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    real_time_session_name text NOT NULL
);

CREATE TABLE real_time_outliers(
    real_time_outlier_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    real_time_session_id int NOT NULL,
    outlier_datetime datetime NOT NULL,
    outlier_data float NOT NULL,
    FOREIGN KEY(real_time_session_id) REFERENCES real_time_detection(real_time_session_id)
);

```

Fig. 4.7.1 Detection Database Schema.

## 4.8 Server Implementation

The design chapter described a method of simulating a real time stream of data using an external server that periodically posts a value for CPU usage. This was implemented using the Python ‘http.server’ library. Separately to Outlier Detection Dashboard, a HTTP server runs handling GET requests. The server has access to multiple datasets representing real-world cloud CPU usage utilisation over time. On each request, a new value of CPU usage is returned as if it were a real time stream of data. This is accessed by the software using the ‘Requests’ library. ‘Requests’ allows HTTP requests to be sent and the return data can be stored as a variable and used in the application [62].

## 4.9 Web Application Implementation

An interactive dashboard application that runs in most browsers was implemented using the Dash library. The front end was implemented by defining the ‘layout’ of the web-application using “Dash HTML components”. Back-end functionality is implemented using “Callback” functions [63]. ‘Callback’ functions were used to bridge the front end to the data generated by the detection algorithms.

‘Plotly’ is used for generating interactive graphs [64]. The software solution is a monitoring tool thus graphs are an important component in aiding with data readability. Line plots, scatter plots and pie charts are implemented throughout the software solution using this library. Additionally, interactive live update graphs are possible with Plotly, this feature is used to display the real time detection of outliers.

## 4.10 Detection Algorithm Implementation

ESMOD was implemented using python. The four detection techniques in ESMOD’s ensemble follow a similar class structure. The dataset and any thresholds are required are passed to the detection methods and a ‘pandas DataFrame’ containing the outliers detected is returned [65]. For real time detection, a window of data points is passed to the detector along with the next data value. The detector returns a confidence score.

The ‘Pycaaret Anomaly Detection’ module was implemented early in the development of this project. It contains an array of unsupervised traditional detection techniques such as SVM and KNN. The detection techniques can be applied to a dataset and outliers are predicted [66]. The techniques in this library were used to build the simple graphing and scoring mechanisms behind the prototype of this application, they were also used as a benchmark to test the ensemble technique proposed in this dissertation.

The ‘Sklearn’ library is used for machine learning in python [67]. In the experimental space of the software, there is a section for supervised learning. The supervised learning methods in this section were implemented using this library.

## 4.11 Architecture

Clean architecture was implemented by applying ‘The Dependency Rule’ to the structure of the application. This rule states that “source code dependencies can only point inwards” [51]. Therefore, the classes implemented with the software solution fall into a hierarchical pattern. This can be observed as a ‘cone’ of the clean architecture diagram described in Fig. 3.3.1.

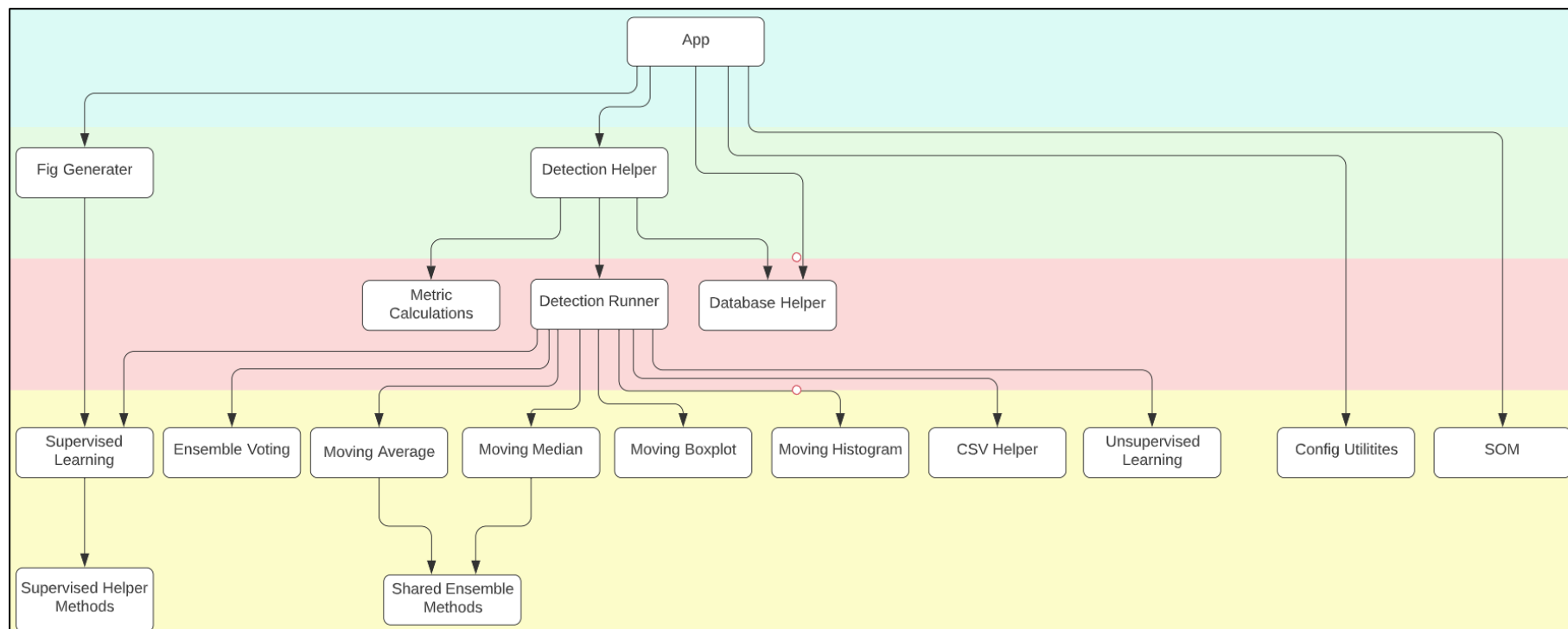


Fig. 4.2.2. Architecture Diagram of Implemented Classes.

Fig. 4.2.2 shows the architecture of the entire software solution (areas are shaded to match those labelled in Fig. 3.5.1). It follows ‘The Dependency Rule’ described as part of ‘Clean Architecture’. Classes in the lower levels do not depend on classes higher up. With the classes implemented this way, the software is effortlessly maintained, and new features are easily implemented.

## 4.12 Deployment to The Cloud

A 'Dockerfile' is used to generate a docker image for Outlier Detection Dashboard which is pushed to an Azure Container Registry. The application runs on an Azure Container Instance (ACI). The ACI does not require the use of VMs, can be auto-scaled and is a cost effective-solution for making the application publicly available.

```

Dockerfile
1 FROM ubuntu:latest
2 ENV DEBIAN_FRONTEND noninteractive
3 RUN apt-get update && apt-get install -y python3 python3-pip
4
5 WORKDIR /app
6 COPY . /app
7
8 RUN pip install -r requirements.txt
9
10 ENTRYPOINT [ "python3" ]
11
12 CMD [ "app.py" ]

```

Fig. 4.12.1 Dockerfile used to generate Outlier Detection Dashboard docker image.

As per the design described in Fig. 3.3.2, two services are needed for the application to function properly: a dashboard presenting the detection data and a server for periodically updating CPU usage for real time detection. Fig. 4.12.2 shows the dashboard and server containers running.



<input type="checkbox"/> Name ↑↓	Resource group ↑↓	Location ↑↓	Status ↑↓	OS type ↑↓	Total c... ↑↓	Subscription ↑↓
<input type="checkbox"/>  cpu-usage-server-container	cpu_usage_server_resources	East US	Running	Linux	1	<a href="#">Azure subscription 1</a>
<input type="checkbox"/>  outlier-detection-dashboard-container	outlier_detection_dashboard_resources	East US	Running	Linux	1	<a href="#">Azure subscription 1</a>

Fig. 4.12.2 Screenshot of Azure dashboard showing running containers.

As mentioned, the application is publicly available and can be accessed using this link [outlier-detection-dashboard.eastus.azurecontainer.io](https://outlier-detection-dashboard.eastus.azurecontainer.io). With the application using remote resources to perform detection, Outlier Detection Dashboard can be accessed from any device including smart phones.

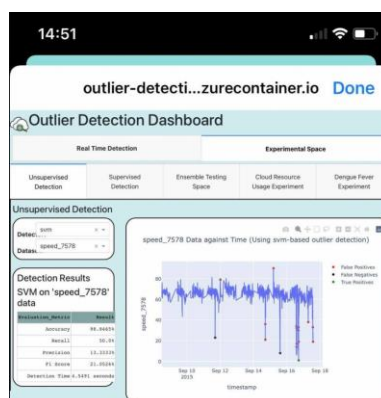


Fig. 4.12.3 Screenshot of Outlier Detection Dashboard accessed using a mobile device.



## 5.0 Testing

### 5.1 Continuous Integration – Automated Test Suite

Continuous integration was paired the ‘Agile’ development lifecycle, this allowed changes to be tested against the entire repository within minutes. New features being implemented were committed to the main branch of the repository (often multiple times a day) for fast feedback on code changes. This was made possible with the implementation of an automated test suite [68].

The implementation of the test suite came in the form of unit tests written in python. White box testing techniques were used to design the automated test suite with the goal of achieving high code coverage. Unit testing is a method of individually testing components of software to validate code acts as expected, unit tests are designed to run quickly, thus new code changes can be validated against the code base within seconds [69].

Before Continuous Integration was implemented with this project, a testing plan was developed. This plan contains the methods/techniques considered when designing and implementing the unit tests.

### 5.2 Test Plan for Design and Implementation of the Automated Test Suite

#### Introduction

The test plan follows white-box testing principles. Test case designs are based on the flow of code of the methods implemented. Test adequacy is determined by the portion of the code that is exercised. For newly implemented features, black-box testing techniques are used to derive initial test cases. Then, an iterative approach is used to improve code coverage by analysing lines and branches missed by the initial test cases. This testing approach verifies functionality exists and works as intended.

#### Scope

To achieve good code coverage, unit tests are written for all methods of a feature implemented. In some instances, unit tests may be inefficient for some of the features (i.e. graph plotting functions). These methods are not included in the automated test suite and are tested a different way.

#### Quality Objective

The ultimate objective of the automated test suite is to ensure methods act as expected and do not break any functionality in the rest of the code base.

Good coverage will guarantee the code is robust and acts as intended. Therefore, a threshold of 90% code coverage over the entire application will be used to verify the code base is well exercised in automated tests.

For maintainability purposes, conventions are used for test code architecture.

- Test code is modularised with respect to the implemented code (unit tests are grouped and ordered).
- Test scripts are named by prefixing 'test\_' to the name of the class being tested.
- The unit test method name refers to the purpose of what the test is attempting to achieve/assert.

Conventions are included as part of the quality objective because it makes the test code maintainable. Maintainability is important when writing code in any form since other developers can review, edit or understand failures with a given test case without having to dissect code line by line.

### **Test Methodology**

White box testing techniques are used to enhance the automated test suite, but initial commits of features include tests written with black box testing techniques. The requirements of a method are used as the test oracle for the initially implemented test cases.

- Black box testing techniques
  - Boundary Value Analysis
 

This black box testing technique ensures there are no defects with a method when extreme or boundary values are used as input.
  - Equivalence-Class Partitioning
 

This method of testing aims to use test code to produce a full set of function outputs with the minimum amount of test cases.
- White box testing techniques
  - Control-Flow Coverage
 

Specifically 'Path Coverage' is used to design test cases. Path coverage ensures all nodes in a method are traversed i.e. every possible condition for a conditional branch is passed to the function.

'Coverage.py' - <https://coverage.readthedocs.io/en/6.3.2/>, is a python test coverage tool. Coverage.py will be used to generate reports on code coverage and determine test adequacy.

## Test Deliverables

There are 2 phases involved with the unit test implementation.

- During feature implementation
  - Black box testing techniques are used to implement an initial set of test cases.
- After feature implementation
  - Coverage.py report is generated to determine code coverage using the black box testing techniques.
  - If coverage < 90%, white box testing techniques are applied to improve test adequacy.
    - Coverage report generated again. Iterate until coverage  $\geq 90\%$ .

At each phase of implementation, testing is performed and coverage can be determined.

## 5.3 Test Environment – Docker

The automated test suite runs in a Docker container, the test suite is triggered when a commit is made to the Gitlab repository. Fig. 5.3.1 shows this pipeline and fig 5.3.2 shows the ‘Gitlab-CI’ configuration. Personal hardware is used to host the Docker container. Without the budget constraint of this project, the repository would be configured to spin up a Docker instance in the cloud, run the tests, and then tear it down. This would remove the need for personal hardware used as the runner.

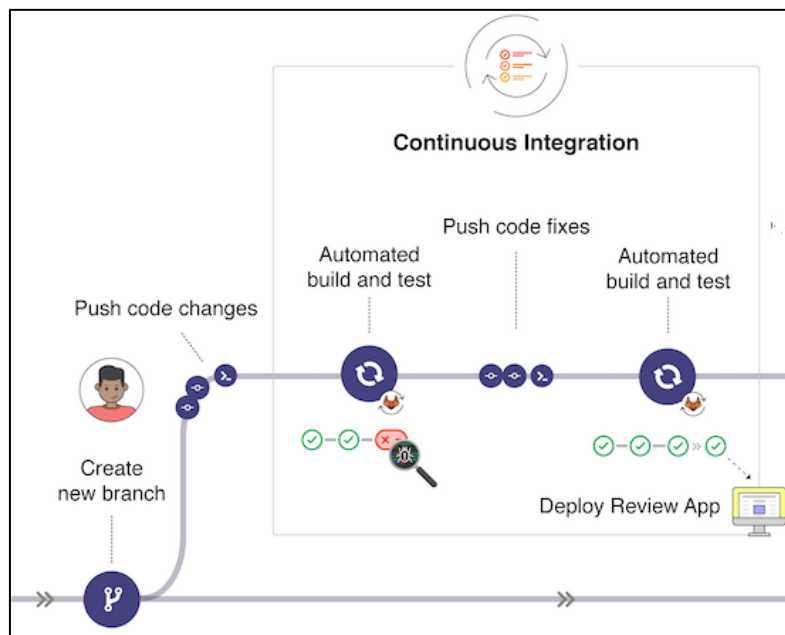


Fig. 5.3.1 Continuous Integration Pipeline [70]

```

1  image: "python:3.8"
2
3  before_script:
4  | - python --version
5  | - pip install -r requirements.txt
6
7  unit_test:
8  | stage: test
9  | script:
10 | - coverage run -m unittest discover
11 | - coverage report

```

Fig. 5.3.2 Gitlab-CI Configuration.

## 5.4 Automated Test Suite Results

The output of the automated test suite executed on the Docker container is shown in Fig. 5.4.1. A large number of unit tests run in a very short amount of time. The coverage report shows good test adequacy ( $\geq 90\%$  total) over the code so the quality objective of the test plan has been achieved.

```

647  Ran 123 tests in 2.243s
648  OK

```

```

701  $ coverage report --omit="*/test*", "*/__init__.py"
702  Name                                                    Stmts   Miss  Cover
703  -----
704  app_helper_scripts/app_detection.py                     150     18    88%
705  app_helper_scripts/app_exceptions.py                     15      3    80%
706  app_helper_scripts/config_utilities.py                   32      0   100%
707  app_helper_scripts/csv_helper.py                         28      0   100%
708  app_helper_scripts/detector_evaluation.py                 39      0   100%
709  app_helper_scripts/metric_calculations.py                 24      1    96%
710  database_scripts/database_helper.py                      116      8    93%
711  ensemble_detectors/ensemble_shared_methods.py            20      0   100%
712  ensemble_detectors/ensemble_voting.py                    52      2    96%
713  ensemble_detectors/moving_average_detection.py            88      3    97%
714  ensemble_detectors/moving_boxplot.py                     106      4    96%
715  ensemble_detectors/moving_histogram_detection.py          139     15    89%
716  ensemble_detectors/moving_median_detection.py             83      2    98%
717  supervised_learning_detectors/data_splitter.py            73      0   100%
718  supervised_learning_detectors/isolation_forest.py         47      0   100%
719  unsupervised_detectors/pycaret_detection.py              23      0   100%
720  -----
721  TOTAL                                                    1035     56    95%
722  Cleaning up project directory and file based variables
723  Job succeeded

```

Fig. 5.4.1 Output of Automated Test Suite Execution

## 6.0 Detector Evaluation

This chapter details the experiments performed to evaluate ESMOD. Two experiments were conducted to determine the performance of ESMOD on labelled and unlabelled datasets. Additionally, an ROC curve was generated to aid in evaluating the detector.

### 6.1 ROC Curve

The Receiver Operating Characteristic (ROC) curve is a method of evaluating a detection algorithm's performance using a confusion matrix [71]. It can be visualised by plotting the detection rate against the false alarm rate, both values can be calculated from the confusion matrix using the following equations.

$$\text{Detection rate} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{False alarm rate} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

Equations used to calculate metrics for ROC curve [72].

The desired ROC curve would have “0% false alarm rate, while having 100% detection rate” [72]. Furthermore, Area Under Curve (AUC) can be calculated from the ROC curve and is used to evaluate the performance of a detector. A detector with an AUC less than 1 is considered imperfect, however, a paper by Alex J. Bowers outlines how classification accuracy can be described for imperfect detectors [73].

- $\text{AUC} \geq 0.85$  = High classification accuracy
- $0.75 < \text{AUC} < 0.85$  = Moderate classification accuracy
- $\text{AUC} \leq 0.75$  = Low classification accuracy

Fig. 6.1.1 represents an ROC curve plotted for ESMOD with changing outlier threshold on the ‘speed\_7578’ dataset described in chapter 1. AUC is 0.824, meaning ESMOD can be described as ‘moderately accurate’.

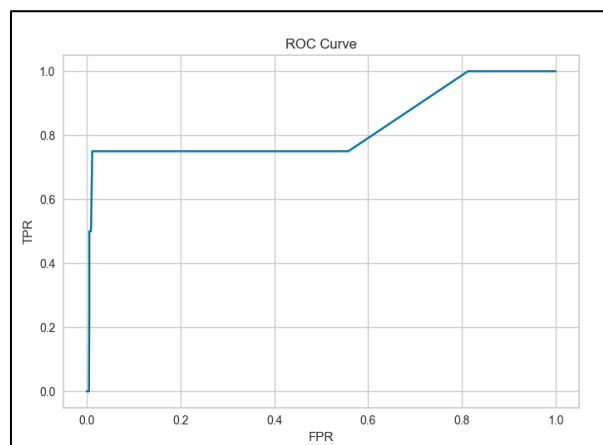


Fig. 6.1.1 ROC Curve for ESMOD on ‘speed\_7578’.

## 6.2 Labelled Data - CPU Utilization Outlier Detection Experiment

### Abstract

This experiment tests the effectiveness of the newly implemented technique of detecting outliers, ESMOD. The experiment is performed on Cloud platform CPU usage. Techniques are evaluated using accuracy, recall, precision and f1. This experiment determines which voting system works best out of two implemented. Results show that the ESMOD can detect outliers in the CPU usage data. It often produces good scores but does not perform well against unstable data. This experiment determines that a ‘Combined Confidence’ voting system produces the best scores.

### Introduction to the Experiment

#### Background Information

The data being analyzed is Amazon Elastic Compute Cloud (EC2) CPU usage. EC2 is a service provided by Amazon used for on-demand cloud infrastructure. Customers use the platform for its compute power, running multiple different kinds of operating systems for various applications [74].

Amazon Web Services (AWS) provides many tools for analyzing the metrics of an EC2 instance. CPU Utilization is arguably the most important metric, since it “identifies the processing power required run an application on a selected instance” [75]. Problems with an EC2 instance, or an application running on one, can often be identified by a discrepancy in CPU usage [76].

This CPU utilization data is labelled, meaning the outlier detection technique applied to the data can be evaluated based on accuracy, precision, recall and f1 score [24].

#### How is the experiment carried out?

This experiment is carried using ‘Outlier Detection Dashboard’, the application developed as part of this project (shown in Fig. 6.2.1). The purpose of this application is to apply outlier detection algorithms to datasets defined by the user.

#### Hypothesis

The implemented outlier detection method (ensemble) is an effective outlier detector for labelled datasets and will generate good scores for accuracy, precision, recall and f1.

The proposed scoring method for the ensemble of detectors will generate better scores than the previously implemented solution (Combined confidence will outperform majority classification).

## Methods

The detection algorithm, ESMOD is used to perform outlier detection on the datasets. Detailed descriptions of this algorithms design are described in chapter 3.1.

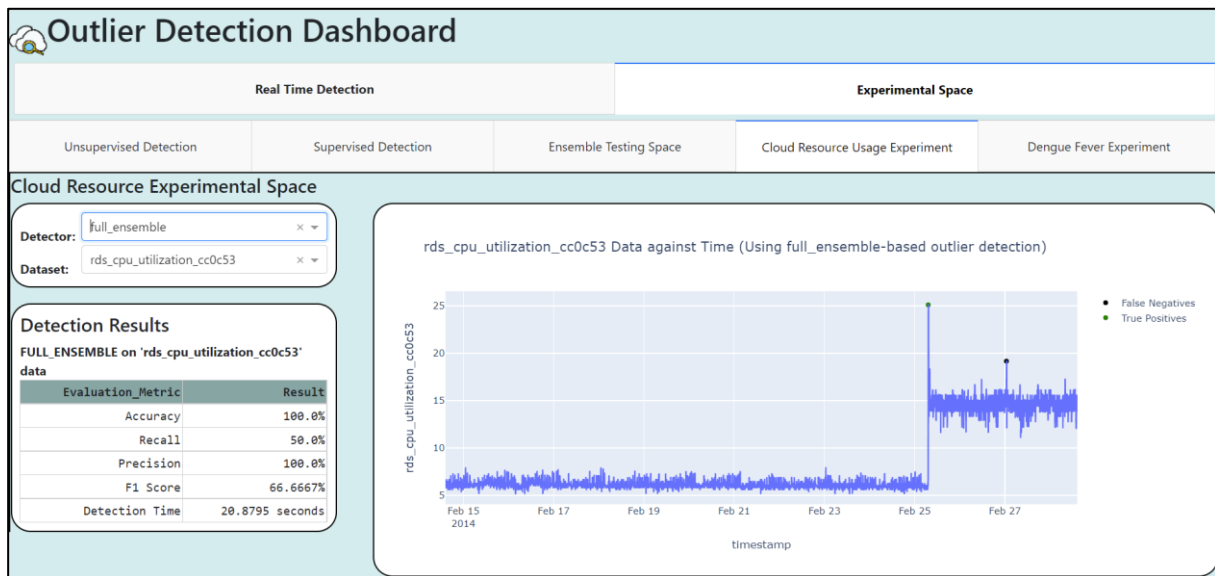


Fig. 6.2.1 Screenshot showing application used to apply outlier detection.

## Method of Scoring the Detector

The datasets being analyzed have been labelled by the Numenta Anomaly Benchmark (NAB) [35]. The software used to perform the experiments uses the labels provided by NAB to plot the outliers. Fig. 6.2.3 shows a sample of time series data with classifications. The key for this data is represented by Fig. 6.2.2.

- False Positives
- False Negatives
- True Positives

Fig. 6.2.2 Outlier detection classification key

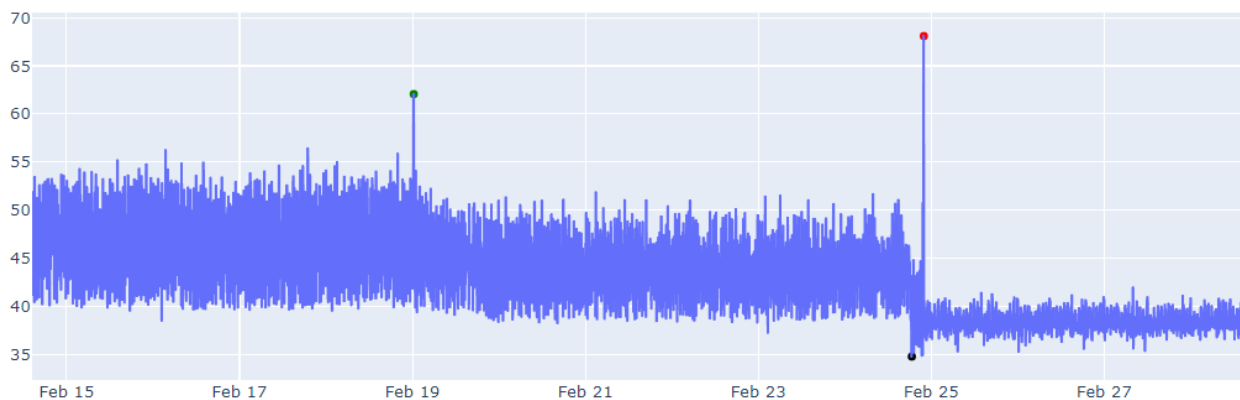


Fig. 6.2.3 Graph demonstrating how the application represents the detection data.

## Evaluating Results

The software calculates accuracy, precision, recall and f1 score to determine the overall performance of the detector. The results of these calculations and graphs generated by the software can be found in Appendix 7.2.

## Evaluating the Voting Methods

The ‘Combined Confidence’ voting system produces better scores than the ‘Majority Classification’ voting system concluding that the former is a better solution, as hypothesized. Tables 1 and 2 in Appendix 7.2 provide detailed scores for each dataset and table 4 provides a side-by-side comparison. ‘Combined Confidence’ produces better scores for precision, recall and f1. Table 3 shows that the ‘Majority Classification’ system cannot outperform moving average (one of the detectors in the ensemble) proving this method is ineffective. ‘Majority Classification’ has a higher average accuracy, but false negatives are crucial when analyzing CPU usage [76], therefore this metric is less useful than precision, recall and f1.

‘Majority Classification’ is more efficient than ‘Combined Confidence’, the simplicity of its voting system results in a lower execution time (less than half the time taken compared to the confidence technique). Some improvements may be required to improve the detection time so that this technique can work for real time outlier detection.

## Evaluating the Effectiveness of ESMOD

Occasionally, ESMOD is very effective. Appendix 7.2, Fig. 7, graphs I, V, VIII and IX show that the detection is working and good scores for recall, precision and f1 are generated.

This technique is especially ineffective against unstable datasets. Graphs II, III and VII in appendix 7.2, Fig. 1, show that the detection has failed, and the ensemble of detectors are ineffective. Although the ensemble produces weak scores in these datasets, table 3 shows that the ‘moving histogram’ detector produces good scores. Fig. 6.2.4 shows a side-by-side comparison of ESMOD performing well and poorly. An improvement to the voting system, by potentially adding weighted confidences, could produce better detection in these graphs.

Observations of Appendix 7.2, Fig. 7 show that the detector is very nearly producing perfect scores for some datasets. Graph VI shows that a false positive detection was made 1 data point in the time series away from the false negative. This would have produced a perfect score for this dataset. Similarly, in graphs VIII and IX, the detectors would have produced perfect scores if they had correctly classified the second false negatives.



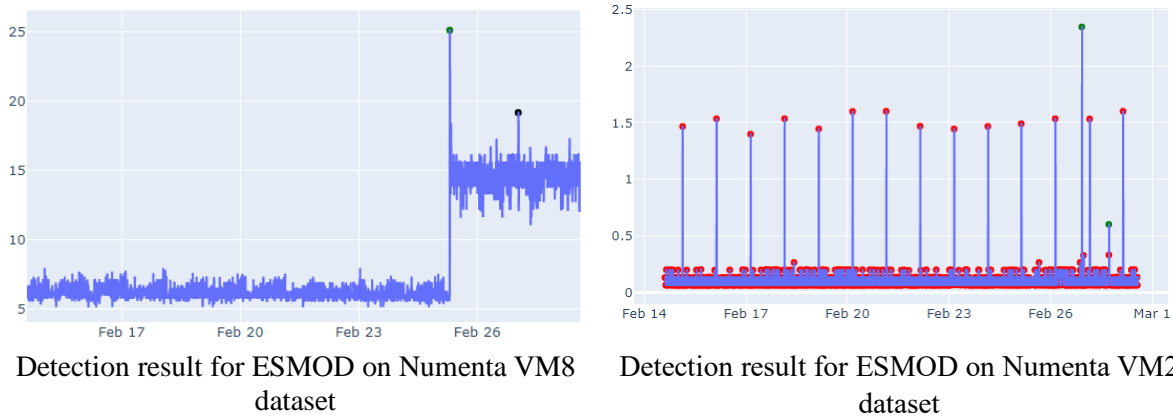


Fig. 6.2.4 Comparison of ESMOD performing well vs. ESMOD performing poorly.

## Conclusion

This experiment evaluated ESMOD and determined that it is effective in detecting outliers and producing good scores in some datasets. It produced an average f1 score of 0.2446.

It was found that this technique is ineffective in detecting outliers in unstable datasets, but some detectors within the ensemble are more effective in unstable datasets than others. Meaning improvements to the voting system could improve upon this issue. In other datasets, perfect scores are almost achieved.

Results show that ‘Combined Confidence’ produces better scores than ‘Majority Classification’ as hypothesized. The intricacy of the voting system means that it takes longer to perform detection, but some optimization techniques could improve this score.

## 6.3 Unlabeled Data - Dengue Fever Rate Experiment

### Abstract

The aim of this experiment is to test the effectiveness of a newly implemented technique of detecting outliers, ESMOD. This technique is tested on Dengue Fever rates in regions of Vietnam. Observations of generated graphs show that this technique is effective in detecting outliers. Obvious outliers and some subtle outliers can be detected using this method but on rare occasions an outlier is missed and there are many false alarms. A comparison with a traditional classifier (KNN) proves that this method of detecting outliers is of good standard.

### Introduction

#### Background Information

This experiment determines the performance of ESMOD on unlabelled datasets. The datasets used concern Dengue fever rates in the Bac Lieu and An Giang regions of Vietnam.

The Dengue Fever is a viral disease that is carried by mosquitos. It is widespread throughout tropical regions. Local environmental factors have an impact of the spread and severity of this virus and it is leading cause of hospitalisation and deaths in the areas that it affects [78].

The data consists of details about the climate of these regions over a 20-year period. Data such as average temperature, rainfall, humidity and the fever rate itself will be considered in this experiment. The data is unlabelled. The techniques used will be unsupervised and will not require any training. The ultimate goal of this experiment is to run outlier detection on these datasets and see if the outliers correlate between different datasets, if they do, then the outlier detection is working.

### **How is the experiment carried out?**

This experiment is carried using the application developed as part of this project, ‘Outlier Detection Dashboard’. The purpose of this application is to apply outlier detection algorithms to datasets defined by the user.

The application plots the data and generate scores for the chosen method. A score is not generated for the detection methods on these datasets since they are unlabelled.

The outlier detection methods in the ensemble are implemented using python. These methods work individually first to make a prediction with a confidence score. A voting system, also implemented using python, determines the final classification.

### **Hypothesis**

The implemented outlier detection method (ESMOD) is an effective outlier detector for unlabelled datasets.

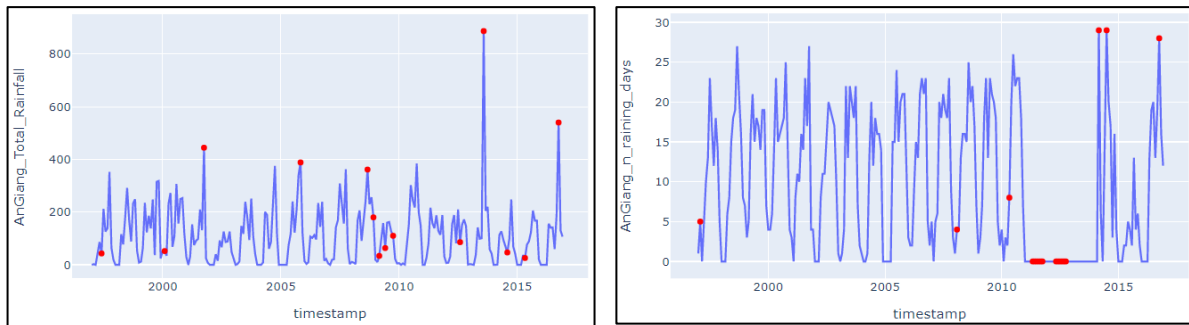
### **Methods**

The methods used to perform detection are described in the previous experiment.

### **Evaluation of An Giang Data**

Graphs showing results generated by the software can be found in Appendix 7.3, Fig. 1.

The graphs above show that outliers have been detected in peaks and troughs throughout the time series, indicating that (possible) actual outliers have been detected. Most notably, a cluster of detections were made in graph I in the spike between 2000 and 2005. This correlates with the outlier detected in graph III indicating rainfall caused a spike in the fever rate. More notably, outliers were indicated in graph IV between 2010 and 2015 where there seems to be no raining days. In graph III, a spike in rainfall is detected, which correlates with the spike in evaporation detected in graph VI around the same time. This correlation can be seen in Fig. 6.3.1.

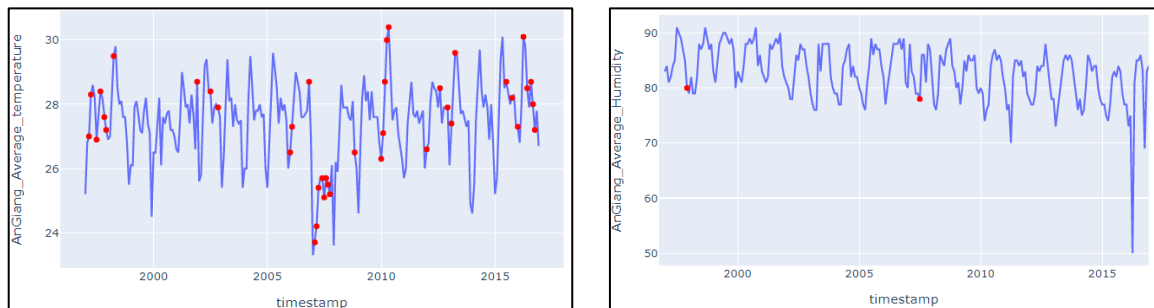


Detection result for ESMOD on An Giang Total  
Rainfall dataset

Detection result for ESMOD on An Giang No.  
Raining Day dataset

Fig. 6.3.1 Correlation in detections made by ESMOD between different An Giang datasets.

A great number of detections have been made in graph V compared with the rest, the detection technique may be ineffective against such unstable data. But a cluster of detections are shown in the 2005 to 2010 period. There is a trough in the time series which correlates with the detection made in graph II where an outlier is detected in average humidity. The outlier detected here does not look irregular when plotted since it is not major peak or a trough but could be expected to be an actual outlier because of the irregular temperature. Fig. 6.3.2 shows ESMOD detecting the subtle outlier.



Detection result for ESMOD on An Giang Average  
Temperature dataset

Detection result for ESMOD on An Giang Average  
Humidity dataset

Fig. 6.3.2 Correlation between abnormal temperature and the outlier detected in humidity in  
An Giang datasets.

### Evaluation of Bac Lieu Data

Graphs showing results generated by the software can be found in Appendix 7.3, Fig. 2.

The graphs show that more outliers are detected here compared with the An Giang data, especially in graph V. Besides this, the outliers detected are in the peaks and troughs of the data. Detections were made around the spikes in fever rates in graph I, but the detector is failing to correctly classify the top of some peaks.

Similarly to graph V in the An Giang Data, the detector has marked many data points as outliers, solidifying the fact that this detector may be ineffective against unstable data. Besides graph V

there are a number of correlations between the graphs. The initial spike in dengue fever rates in graph I correlate with the average humidity in graph II and the drop in no. raining days in graph IV. The fall in fever rates and humidity is shown in Fig. 6.3.3. Like in the An Giang detections, the trough in graph IV is picked up by the detector as well as a sudden spike in total rainfall in graph III.

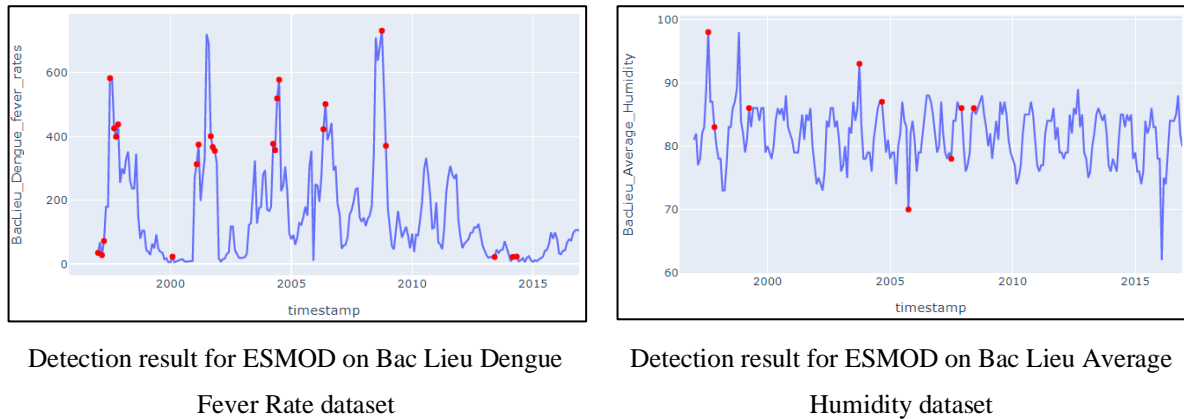


Fig 6.3.3 Comparison of ESMOD detection on Bac Lieu datasets showing outlier in humidity where fever rates fell.

### Comparison with a Traditional Classification Technique – KNN

To test the effectiveness of this newly implemented ensemble, KNN outlier detection has been applied to the datasets [79].

### Comparison of Results

KNN failed to detect some of the obvious outliers (peaks/troughs) that are detected by ESMOD. In graph IV of the An Giang data, ESMOD correctly identifies outliers in the 3-4 year stretch where there were no raining days, but KNN fails to detect this data as anomalous. A comparison of this observation is shown in Fig. 6.3.4. Similarly, KNN detects two spikes in dengue fever rates in the An Giang region and misses a major peak in around 2007, ESMOD detects this spike.

KNN performs better for graph V in both regions. ESMOD detects many outliers whereas KNN detects a few in areas of the graphs that (appear to be) actual outliers. KNN is better at detecting datapoints at the top of peaks.

Something important to note when comparing these results is that the ensemble method takes much less time to run than KNN. It took KNN ~6 seconds to process each dataset and it took the ensemble ~1 second using the same hardware.

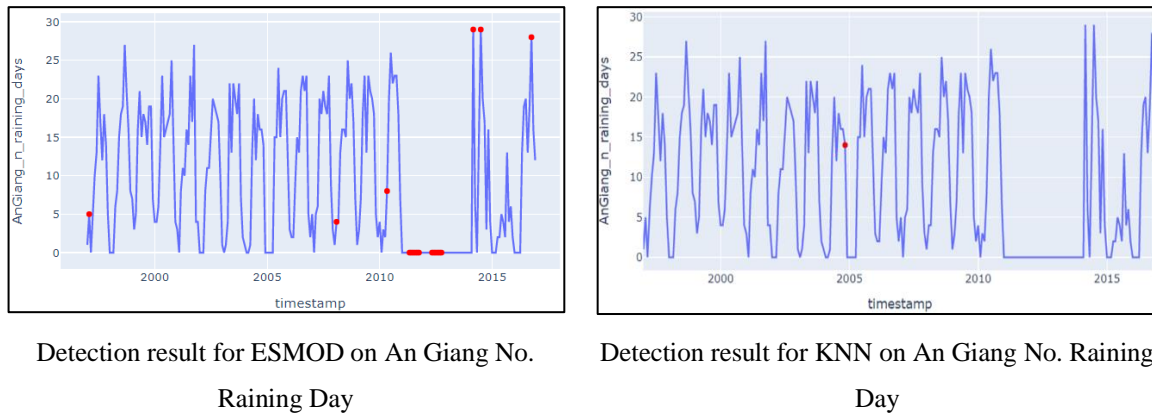


Fig. 6.3.4 Comparison of ESMOD and KNN detection on An Giang No. Raining Day dataset.

## Conclusion

ESMOD is effective in detecting outliers in unlabelled datasets. However, it is difficult to say to what extent it is effective since accuracy, precision, recall and f1 are impossible to calculate without labels. Further research is needed in techniques of evaluating outlier detection in unlabelled datasets. But, by comparing this method with a traditional classifier, and observing peaks and troughs within the datasets, it can be said that the ensemble is detecting outliers in the correct places. For most datasets, ESMOD performs the same or even better than the traditional KNN detector but it evident that the ESMOD needs more work around unstable datasets. Another important thing to note is that with similar results, the ensemble performs detection up to 6x faster than KNN.

## 6.4 Future Work

This dissertation proposed ESMOD, an outlier detection technique designed to be effective in detecting abnormal behaviour in VM resource usage. Experiments proved that ESMOD outperforms other traditional techniques in detecting outliers in non-stationary univariate time series data and that it produces good scores for accuracy, precision, recall and f1 when used on cloud resource data (CPU usage).

Evidence from experiments show that although ESMOD is an effective outlier detector, it can fail when used on highly unstable datasets. A desirable component of this detection technique would be the ability to adjust thresholds and window sizes accordingly when faced with these kinds of datasets. Potentially, an analysis of periodic ‘peaks’ or ‘troughs’ would be used to determine thresholds meaning hour or minute of day is used a feature for the classifier. This could reduce ESMOD’s false alarm rate.

Outlier Detection Dashboard was proposed as part of this dissertation and has proved to be effective in visualising real time outlier detection and performing experiments using various outlier detection techniques. Chapter 3 discussed a design that did not require various VMs running complex applications to test the adequacy of the software, however this solution could be expanded. Options to allow the user to configure cloud resources from various vendors could be implemented. This would allow Cloud Systems Administrators to observe the performance of newly deployed VMs hosted on multiple different platforms. Comparisons of abnormal behaviour between different vendors could influence where an organisation choose to build their cloud infrastructure and deploy their services.

## **6.5 Closing Thoughts**

This dissertation discussed the research and implementation of methods to improve the uptime and availability of cloud based services. This has been achieved by developing ESMOD and Outlier Detection Dashboard. The increased use of the applications and services deployed on cloud based platforms will inevitably warrant the development of multiple tools and detection algorithms for effective monitoring of Virtual Machine resources. But, with the completion of the suggested future work, Outlier Detection Dashboard could be a free, publicly available tool that serves Cloud Systems Administrators for years to come.

## References

Gitlab repository - [40231992 / Outlier Detection in Virtual Machines · GitLab \(qub.ac.uk\)](#)

- [1] D. M. Hawkins, "Introduction," in Identification of Outliers, Dordrecht: Springer Netherlands, 1980, pp. 1–12. doi: 10.1007/978-94-015-3994-4\_1.
- [2] P.K. Paul, A. Kumar. and M. Ghosh. "Cloud Computing: the 21st Century Friend for Virtualization." *About Mewar University* (2012). Available: [Paper Title \(use style: paper title\) \(researchgate.net\)](#)
- [3] Cast AI. (2022, Jan. 11) "The cloud in 2021: 21 game-changing outages, security issues, and highlights" [Online]. Available: <https://cast.ai/blog/the-cloud-in-2021-21-game-changing-outages-security-issues-and-highlights/>
- [4] N. M. Shousha, L. R. Abdelgawad, and others, "Down In Minutes, Out For Six Hours: A Brief Report on Feelings during the Outage of Whatsapp, Instagram, and Facebook," *British Journal of Psychology Research*, vol. 9, no. 2, pp. 38–44, 2021. Available: <https://tudr.org/id/eprint/43/>
- [5] The Guardian. (2021, Oct. 10) Impact of WhatsApp, Facebook and Instagram outage on businesses [Online]. Available: <https://guardian.ng/technology/social-media/impact-of-whatsapp-facebook-and-instagram-outage-on-businesses/>
- [6] J. Frank. (2019, Jul. 29) Your Next Move: Cloud Systems Administrator [Online]. Available: <https://www.comptia.org/blog/your-next-move-cloud-systems-administrator>
- [7] 451 Research. (2017) Voice of the Enterprise: Cloud Transformation survey [Online] Available: [https://451research.com/images/Marketing/press\\_releases/Pre\\_Re-Invent\\_2018\\_press\\_release\\_final\\_11\\_22.pdf](https://451research.com/images/Marketing/press_releases/Pre_Re-Invent_2018_press_release_final_11_22.pdf)
- [8] AWS. (n.d.) Who is using cloud computing? [Online]. Available: <https://aws.amazon.com/what-is-cloud-computing/>
- [9] S. Soares, F. Bonnet, and J. Berg. (2021, Apr. 25) Working from home during the COVID-19 pandemic: Updating global estimates using household survey data [Online]. Available: <https://voxeu.org/article/working-home-during-covid-19-pandemic-updated-estimates>
- [10] D. Anderson, C. Kelliher (2020), "Enforced remote working and the work-life interface during lockdown", *Gender in Management*, Vol. 35 No. 7/8, pp. 677-683. Available: <https://doi.org/10.1108/GM-07-2020-0224>

- [12] P.J. Cho, et al. "Demographic Imbalances Resulting from Bring-Your-Own-Device Study Design." *JMIR mHealth and uHealth* (2021). Available: <https://s3.ca-central-1.amazonaws.com/assets.jmir.org/assets/preprints/preprint-29510-accepted.pdf>
- [12] T. Mangan (2020, Mar. 30) The Tech Supporting Remote Workers [Online]. Available: <https://www.nutanix.com/theforecastbynutanix/technology/modern-remote-workforce-is-powered-by-virtual-desktop-technology>
- [13] H. Zhang, S. Chen, J. Liu, Z. Zhou, and T. Wu, "An incremental anomaly detection model for virtual machines" (2017). Available: <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0187488>
- [14] A. Chaudhuri. (2015). "Best Practices for Mitigating Risks in Virtualized Environments" - *Cloud Security Alliance, USA*. Available: [https://downloads.cloudsecurityalliance.org/whitepapers/Best\\_Practices\\_for%20Mitigating\\_Risks\\_Virtual\\_Environments\\_April2015\\_4-1-15\\_GLM5.pdf](https://downloads.cloudsecurityalliance.org/whitepapers/Best_Practices_for%20Mitigating_Risks_Virtual_Environments_April2015_4-1-15_GLM5.pdf)
- [15] EBA/GL. (2017) Guidelines on ICT Risk Assessment under the Supervisory Review and Evaluation process (SREP) (Page 11). Final Report on ICT Risk Assessment Guidelines. [Online] Available: EBA BS 2017 131 (Final Guidelines on ICT Risk Assessment under SREP) (europa.eu) (Accessed: 4 January 2022).
- [16] N. Arya., M Gidwani., S.K. Gupta. "Hypervisor Security - A Major Concern" *International Journal of Information and Computation Technology. Volume 3, Number 6, pp. 533-538*. Aug. 2013. Available: [https://www.ripublication.com/irph/ijict\\_spl/08\\_ijictv3n6spl.pdf](https://www.ripublication.com/irph/ijict_spl/08_ijictv3n6spl.pdf)
- [17] T. Morrow. (2018) 12 Risks, Threats, & Vulnerabilities in Moving to the Cloud. *Carnegie Mellon University's Software Engineering Institute Blog* [Online]. Available: <http://insights.sei.cmu.edu/blog/12-risks-threats-vulnerabilities-in-moving-to-the-cloud/> (Accessed March 28, 2022)
- [18] AWS. (2022) Amazon CloudWatch Features [Online]. Available: <https://aws.amazon.com/cloudwatch/features/>
- [19] AWS. (n.d.) Using CloudWatch anomaly detection – *AWS Documentation* [Online]. Available: [https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch\\_Anomaly\\_Detection.html](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch_Anomaly_Detection.html)
- [20] H.P. Kriegel., P Kroger., A. Zimek. (2010) "Outlier Detection Techniques" *The 2010 SIAM International Conference on Data Mining Slide 6-7*. Available:



- <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.183.1847&rep=rep1&type=pdf>
- [21] V. Chandola. A. Banerjee., V Kumar. “Anomaly Detection: A Survey.” *ACM Computing Surveys, Vol. 41, No. 3, Article 15.5*. Jul, 2019. Available: <https://dl.acm.org/doi/epdf/10.1145/1541880.1541882>
  - [22] J.-R. Jiang, J.-B. Kao, and Y.-L. Li, “Semi-Supervised Time Series Anomaly Detection Based on Statistics and Deep Learning,” *Applied Sciences*, vol. 11, no. 15, 2021. Available: <https://doi.org/10.3390/app11156698>
  - [23] J. Brownlee. (2016, Nov. 18) What is a Confusion Matrix in Machine Learning [Online]. Available: <https://machinelearningmastery.com/confusion-matrix-machine-learning/> (Accessed 2022, Apr. 17)
  - [24] Y. Sasaki, “The truth of the F-measure,” *Teach Tutor Mater*, Jan. 2007. Available: [https://www.researchgate.net/publication/268185911\\_The\\_truth\\_of\\_the\\_F-measure](https://www.researchgate.net/publication/268185911_The_truth_of_the_F-measure)
  - [25] T. Wood. (n.d.) Machine Learning Glossary and Terms – F-Score [Online]. Available: <https://deeptai.org/machine-learning-glossary-and-terms/f-score>
  - [26] H.-S. Wu, “A survey of research on anomaly detection for time series,” in *2016 13th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, 2016, pp. 426–431. Available: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8079887>
  - [27] R. E. Emanuel., et. Al. (n.d.) Stationary and Nonstationary Behaviour [Online]. Available: <https://serc.carleton.edu/hydromodules/steps/236435.html>
  - [28] A. A. Cook, G. Misirlı, and Z. Fan, “Anomaly Detection for IoT Time-Series Data: A Survey,” *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6481–6494, 2020 Available: [Anomaly Detection for IoT Time-Series Data: A Survey | IEEE Journals & Magazine | IEEE Xplore](#)
  - [29] J. Gama, I. Žliobaitundefined, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A Survey on Concept Drift Adaptation,” *ACM Comput. Surv.*, vol. 46, no. 4, Mar. 2014. Available: <https://doi.org/10.1145/2523813>
  - [30] T. G. Dietterich and others, “Ensemble learning,” *The handbook of brain theory and neural networks*, vol. 2, no. 1, pp. 110–125, 2002. Available: <https://courses.cs.washington.edu/courses/cse446/12wi/tgd-ensembles.pdf>
  - [31] T. T. Dang, H. Y. T. Ngan and W. Liu, "Distance-based k-nearest neighbors outlier detection method in large-scale traffic data," 2015 IEEE International Conference on

- Digital Signal Processing (DSP), 2015, pp. 507-510. Available: <https://ieeexplore.ieee.org/document/7251924>
- [32] M. Goldstein and A. Dengel, "Histogram-based Outlier Score (HBOS): A fast Unsupervised Anomaly Detection Algorithm," Sep. 2012. Available: <https://www.goldiges.de/publications/HBOS-KI-2012.pdf>
- [33] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. SIGMOD Rec. 29, 2 (June 2000), 93–104. Available: <https://dl.acm.org/doi/10.1145/335191.335388>
- [34] F. T. Liu, K. M. Ting and Z. Zhou, "Isolation Forest," 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 413-422. Available: [Isolation Forest | IEEE Conference Publication | IEEE Xplore](#)
- [35] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying Density-Based Local Outliers," in Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, 2000, pp. 93–104. Available: [Outlier Detection with One-Class SVMs: An Application to Melanoma Prognosis - PMC \(nih.gov\)](#)
- [36] Numenta. (2015). *The Numenta Anomaly Benchmark – White Paper*. 1. Available: <https://numenta.com/assets/pdf/numenta-anomaly-benchmark/NAB-Business-Paper.pdf#:~:text=The%20Numenta%20Anomaly%20Benchmark%20%28NAB%29%20is%20an%20open,NAB%3A%20the%20labeled%20dataset%20and%20the%20scoring%20system.>
- [37] Cloudfactory. (n.d.). *The Ultimate Guide to Data Labelling for Machine Learning* [Online] Available: <https://www.cloudfactory.com/data-labeling-guide>
- [38] Q. Nguyen, P. Eades, and S.-H. Hong, "On the faithfulness of graph visualizations," in 2013 IEEE Pacific Visualization Symposium (PacificVis), 2013, pp. 209–216. Available: [https://link.springer.com/content/pdf/10.1007/978-3-642-36763-2\\_55.pdf](https://link.springer.com/content/pdf/10.1007/978-3-642-36763-2_55.pdf)
- [39] A. Laurent, and M. Z. Hauschild. "Normalisation." *Life cycle impact assessment*. Springer, Dordrecht, 2015. 271-300. Available: [http://nozdr.ru/data/media/biblio/kolxoz/Cs/CsDb/Stephens%20R.%20Beginning%20database%20design%20solutions%20\(Wiley,%202009\)\(ISBN%200470385499\)\(O\)\(552s\)\\_CsDb\\_.pdf](http://nozdr.ru/data/media/biblio/kolxoz/Cs/CsDb/Stephens%20R.%20Beginning%20database%20design%20solutions%20(Wiley,%202009)(ISBN%200470385499)(O)(552s)_CsDb_.pdf)
- [40] N. Bevan, J. Carter, and S. Harker, "ISO 9241-11 Revised: What Have We Learnt About Usability Since 1998?," Aug. 2015, pp. 143–151. Available: [https://link.springer.com/chapter/10.1007/978-3-319-20901-2\\_13](https://link.springer.com/chapter/10.1007/978-3-319-20901-2_13)

- [41] D. Stone, C. Jarrett, M. Woodroffe, and S. Minocha, User Interface Design and Evaluation. Elsevier Science, 2005. Available: <https://books.google.co.uk/books?id=VvSoyqPBPbMC>
- [42] M. Olan. "Unit testing: test early, test often." *Journal of Computing Sciences in Colleges* 19.2 (2003): 319-328. Available: [https://scholar.google.com/scholar?hl=en&as\\_sdt=0%2C5&q=unit+testing&btnG=](https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=unit+testing&btnG=)
- [43] T. Hamilton. (2022, Feb. 19) Test Plan Template: Sample Document with Web Application Example. *Guru99* [Online]. Available: <https://www.guru99.com/test-plan-for-project.html>
- [44] Dr. Dataman. (2021, Apr, 18). *Anomaly Detection for Time Series. (1) Simple Moving Average*. [Online]. Available: <https://medium.com/dataman-in-ai/anomaly-detection-for-time-series-a87f8bc8d22e>
- [45] Anomaly. (2016, Jan. 12). *Detecting Anomalies with Moving Median Decompsition*. [Online]. Available: <https://anomaly.io/anomaly-detection-moving-median-decomposition/index.html>
- [46] K. Andrea, G. Shevlyakov, and P. Smirnov, "Detection of outliers with boxplots," Jan. 2013, pp. 141–144. [https://www.researchgate.net/publication/261173084\\_Detection\\_of\\_outliers\\_with\\_boxplots](https://www.researchgate.net/publication/261173084_Detection_of_outliers_with_boxplots)
- [47] A. Tsymbal, "The Problem of Concept Drift: Definitions and Related Work," May 2004. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.58.9085&rep=rep1&type=pdf>
- [48] J. L. Harrington, Relational Database Design and Implementation. Elsevier Science, 2016. Available: <https://books.google.co.uk/books?id=yQgFCgAAQBAJ&printsec=frontcover#v=onepage&q&f=false>
- [49] AWS. (n.d.) How Amazon CloudWatch works [Online]. Available: [https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch\\_architecture.html](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch_architecture.html)
- [50] A. Ajiboye, R. Abdullah-Arshah, and Q. Hongwu, "Evaluating the effect of dataset size on predictive model using supervised learning technique," 2015. Available: <https://uilspace.unilorin.edu.ng/handle/20.500.12484/1306>

- [51] P. Ivanics. (n.d.) An Introduction to Clean Code Architecture. *Department of Computer Science – University of Helsinki* [Online]. Available: [http://pivanics.users.cs.helsinki.fi/portfolio/docs/publications/Peter\\_Ivanics-Clean\\_Software\\_Architecture.pdf](http://pivanics.users.cs.helsinki.fi/portfolio/docs/publications/Peter_Ivanics-Clean_Software_Architecture.pdf)
- [52] R. C. Martin. (2012, Aug. 13) The Clean Architecture. *The Clean Code Blog*. [Online]. Available: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>
- [53] J. Shore and S. Warden, The Art of Agile Development. O'Reilly Media, 2021. Available: [https://books.google.com/books?hl=en&lr=&id=i3ZIEAAAQBAJ&oi=fnd&pg=PT4&dq=agile+development&ots=VCWTBZ\\_I4Z&sig=ruhYeT7IqwxD1\\_HXacwqBCLYvoE](https://books.google.com/books?hl=en&lr=&id=i3ZIEAAAQBAJ&oi=fnd&pg=PT4&dq=agile+development&ots=VCWTBZ_I4Z&sig=ruhYeT7IqwxD1_HXacwqBCLYvoE)
- [54] Coursera. (2022, Mar. 21) What Is Python Used For? A Beginner's Guide [Online]. Available: <https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python>
- [55] Tiobe. (2022, April) TIOBE Index for April 2022 [Online]. Available: <https://www.tiobe.com/tiobe-index/>
- [56] D. Goodger., G. Van Rossum. (2001, Jul. 5) PEP 8 – Style Guide for Python Code [Online]. Available: [PEP 8 – Style Guide for Python Code | peps.python.org](https://peps.python.org/pep-0008/) (Accessed 18/4/2022)
- [57] G. V. Rossum, B. Warsaw, and N. Coghlan. (2001, May. 29) PEP 257 – Docstring Conventions [Online]. Available: [PEP 257 – Docstring Conventions | peps.python.org](https://peps.python.org/pep-0257/) (Accessed 18/4/2022)
- [58] B. Latte, S. Henning, and M. Wojcieszak, “Clean Code: On the Use of Practices and Tools to Produce Maintainable Code for Long-Living,” in Proceedings of the Workshops of the Software Engineering Conference 2019, Feb. 2019, vol. Vol-2308, pp. 96–99. [Online]. Available: <http://CEUR-WS.org>
- [59] Visual Studio Code. (n.d.) Why did we build Visual Studio Code? [Online]. Available: <https://code.visualstudio.com/docs/editor/whyvscode>
- [60] Insights – Stackoverflow. (2021) Stack Overflow Developer Survey 2021 – Most Popular Technologies. [Online]. Available: <https://insights.stackoverflow.com/survey/2021#technology>
- [61] SQLite. (n.d.) What Is SQLite? [Online]. Available: <https://www.sqlite.org/index.html>

- [62] Python-Requests. (n.d.) Requests: HTTP for Humans. [Online]. Available: <https://docs.python-requests.org/en/latest/>
- [63] A. Tomar. (2021, Mar. 17) Dash for Beginners: Create Interactive Python Dashboards. [Online]. Available: <https://towardsdatascience.com/dash-for-beginners-create-interactive-python-dashboards-338bfc66ffa4>
- [64] Plotly. (n.d.) Plotly Python Open Source Graphing Library. [Online]. Available: <https://plotly.com/python/>
- [65] Pandas. (2022) Pandas Documentation. [Online]. Available: <https://pandas.pydata.org/docs/>
- [66] Pycaret. (n.d.) Anomaly Detection – *Pycaret Quickstart* [Online]. Available: <https://pycaret.gitbook.io/docs/get-started/quickstart#anomaly-detection>
- [67] Scikit-learn (n.d.) Scikit-learn – Getting Started [Online]. Available: [https://scikit-learn.org/stable/getting\\_started.html](https://scikit-learn.org/stable/getting_started.html)
- [68] S. M. Mohammad. “Continuous Integration and Automation.” *International Journal of Creative Research Thoughts (IJCRT)*, ISSN:2320-2882, Volume.4, Issue 3, pp.938-945, July 2016. Available: <https://ssrn.com/abstract=3655567>
- [69] D. Sale, *Testing Python: Applying Unit Testing, TDD, BDD, and Acceptance Testing*. Wiley, 2014. [Online]. Available: <https://books.google.co.uk/books?id=pp25wgEACAAJ>
- [70] Gitlab Docs. (n.d.) CI/CD Concepts [Online]. Available: <https://docs.gitlab.com/ee/ci/introduction/index.html#continuous-integration>
- [71] A. P. Bradley, “The use of the area under the ROC curve in the evaluation of machine learning algorithms,” *Pattern Recognition*, vol. 30, no. 7, pp. 1145–1159, 1997. <https://www.sciencedirect.com/science/article/pii/S0031320396001422>
- [72] A. Lazarevic and V. Kumar, “Feature Bagging for Outlier Detection,” in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, 2005, pp. 157–166.
- [73] A. J. Bowers and X. Zhou, “Receiver Operating Characteristic (ROC) Area Under the Curve (AUC): A Diagnostic Measure for Evaluating the Accuracy of Predictors of Education Outcomes,” *Journal of Education for Students Placed at Risk (JESPAR)*, vol. 24, no. 1, pp. 20–46, 2019. <https://academiccommons.columbia.edu/doi/10.7916/d8-nc5k-3m53>
- [74] AWS Amazon. (2022). *Amazon EC2 – Secure and resizable compute capacity for virtually any workload* [Online]. Available: <https://aws.amazon.com/ec2/>

- [75] AWS Amazon (2022). *Monitor Amazon EC2* [Online]. Available: [https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring\\_ec2.html](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring_ec2.html)
- [76] Ionos (2020, Feb. 24) *High CPU usage: What does this mean?* [Online]. Available: <https://www.ionos.com/digitalguide/server/know-how/cpu-usage/>
- [77] P. Huilgol. (2019, Aug. 24) *Accuracy vs. F1-Score* [Online]. Available: <https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2>
- [78] World Health Organization. (2022, Jan. 10). Dengue and Severe Dengue [Online]. Available: <https://www.who.int/en/news-room/fact-sheets/detail/dengue-and-severe-dengue>
- [79] Harrison, O. (2018, Sep. 10). Machine Learning Basics with the K-Nearest Neighbours Algorithm. [Online]. Available: <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>

## 7.0 Appendices

### 7.1 Initial Outlier Detection Results

Table 1 Experimental Evaluation Metrics for ESMOD

Dataset	Accuracy	Recall	Precision	f1	Time to execute
Dataset 1	90.2	77.8	87.5	82.4	4.6247
Dataset 2	90.1	66.7	100.0	80.0	20.0145
Dataset 3	89.4	66.7	10.0	17.4	8.6202
Dataset 4	90.1	100.0	62.5	76.9	7.8199
Dataset 5	89.3	94.7	19.8	32.7	20.6700
Average	89.82	81.18	55.96	57.88	12.3498

Table 2 Experimental Evaluation Metrics for KNN

Dataset	Accuracy	Recall	Precision	f1	Time to execute
Dataset 1	89.4	71.4	38.5	50.0	0.4956
Dataset 2	89.1	87.5	13.7	23.7	0.6939
Dataset 3	89.4	88.9	25.8	40.0	0.4729
Dataset 4	89.7	100.0	38.7	55.8	0.5251
Dataset 5	89.2	92.5	18.7	31.1	1.6133
Average	89.36	88.06	27.08	40.12	0.7602

Table 3 Experimental Evaluation Metrics for SVM

Dataset	Accuracy	Recall	Precision	f1	Time to execute
Dataset 1	89.8	80.0	53.3	64.0	2.3065
Dataset 2	89.2	90.0	17.6	29.5	2.0780
Dataset 3	89.2	85.7	18.8	30.8	1.4419
Dataset 4	89.9	100.0	46.9	63.8	1.1291
Dataset 5	89.2	92.7	19.1	31.7	20.2167
Average	89.46	89.68	31.14	43.96	5.4344

Table 4 Experimental Evaluation Metrics for iForest

Dataset	Accuracy	Recall	Precision	f1	Time to execute
Dataset 1	88.6	25.0	6.7	10.5	1.0115

<b>Dataset 2</b>	89	83.3	9.8	17.5	1.3546
<b>Dataset 3</b>	88.8	33.3	3.1	5.7	1.2097
<b>Dataset 4</b>	89.5	100.0	31.2	47.6	1.2103
<b>Dataset 5</b>	89.0	84.6	11.1	19.6	2.9436
<b>Average</b>	88.98	65.24	12.38	20.18	1.5450

Table 5 Experimental Evaluation Metrics for LOF

<b>Dataset</b>	<b>Accuracy</b>	<b>Recall</b>	<b>Precision</b>	<b>f1</b>	<b>Time to execute</b>
<b>Dataset 1</b>	88.4	0.0	0.0	0.0	0.4495
<b>Dataset 2</b>	89.1	100.0	11.8	21.1	0.6609
<b>Dataset 3</b>	89.0	60.0	9.4	16.2	0.8000
<b>Dataset 4</b>	89.8	100.0	40.6	57.8	0.7193
<b>Dataset 5</b>	89.1	84.4	13.6	23.4	1.5037
<b>Average</b>	89.08	68.88	15.08	23.7	0.8267

Table 6 Experimental Evaluation Metrics for Histogram

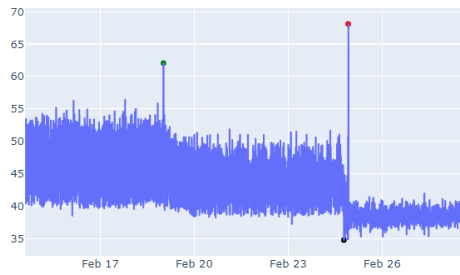
<b>Dataset</b>	<b>Accuracy</b>	<b>Recall</b>	<b>Precision</b>	<b>f1</b>	<b>Time to execute</b>
<b>Dataset 1</b>	88.6	0	0	0	0.4460
<b>Dataset 2</b>	88.8	0	0	0	0.6569
<b>Dataset 3</b>	88.9	0	0	0	0.6308
<b>Dataset 4</b>	89.2	0	0	0	3.0579
<b>Dataset 5</b>	89.5	0	0	0	0.8865
<b>Average</b>	89.0	0	0	0	1.1356

Table 7 Dataset Naming Convention

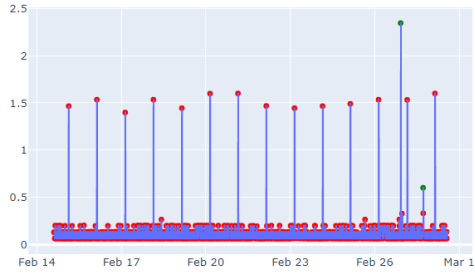
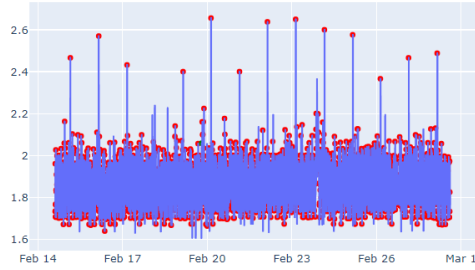
<b>Dataset Name</b>	<b>Dataset CSV File Name</b>
Dataset 1	speed_7578
Dataset 2	ec2_cpu_utilization_5f5533
Dataset 3	TravelTime_387
Dataset 4	occupancy_t4013
Dataset 5	Twitter_volume_UPS



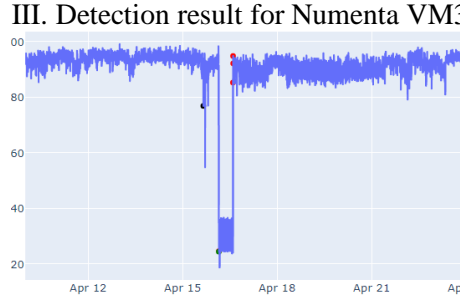
## 7.2 EC2 CPU Experiment Results



I. Detection result for Numenta VM1

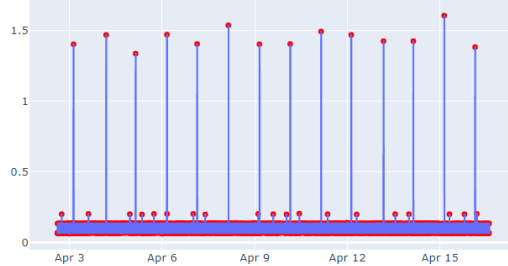


II. Detection result for Numenta VM2



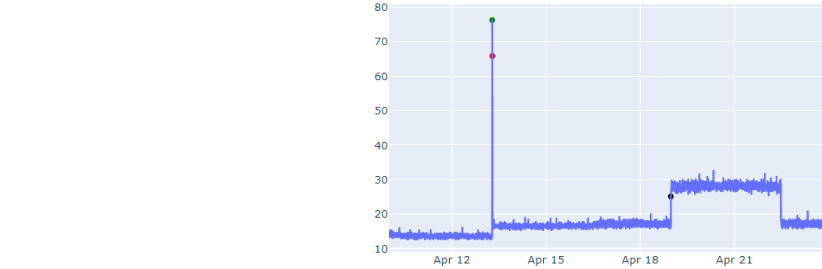
III. Detection result for Numenta VM3

IV. Detection result for Numenta VM4



V. Detection result for Numenta VM5

VI. Detection result for Numenta VM6



VII. Detection result for Numenta VM7

VIII. Detection result for Numenta VM8

IX. Detection result for Numenta VM9

Fig. 1 Graphs showing CPU utilization over time with outliers detecting using the Ensemble and 'Combined Confidence' voting.

Table 1 Results for ensemble detection with ‘Combined Confidence’ voting.

VM NAME	Accuracy	Recall	Precision	f1	Time to execute
<b>Numenta VM1</b>	99.97	50.00	50.00	50.00	14.3014
<b>Numenta VM2</b>	66.10	100.00	0.15	0.29	21.6323
<b>Numenta VM3</b>	43.23	100.00	0.09	0.17	25.6607
<b>Numenta VM4</b>	99.20	0.00	0.00	0.00	14.0828
<b>Numenta VM5</b>	99.90	50.00	20.00	28.57	14.4061
<b>Numenta VM6</b>	99.98	0.00	0.00	0.00	14.2985
<b>Numenta VM7</b>	70.26	0.00	0.00	0.00	18.5497
<b>Numenta VM8</b>	100.00	50.00	100.00	66.67	13.734
<b>Numenta VM9</b>	99.98	50.00	50.00	50.00	13.8773
<b>Average</b>	86.51	50.00	27.53	24.46	16.7270

\* Numenta VM7 is excluded from the average since there are no true positives and an f1 score cannot be calculated \*

Table 2 Results for ensemble detection with ‘Majority Classification’

VM NAME	Accuracy	Recall	Precision	f1	Time to execute
<b>Numenta VM1</b>	99.98	50.00	50.00	50.00	5.3486
<b>Numenta VM2</b>	77.75	100.00	0.22	0.44	7.2271
<b>Numenta VM3</b>	78.17	100.00	0.23	0.45	8.7147
<b>Numenta VM4</b>	99.93	0.00	0.00	0.00	5.4163
<b>Numenta VM5</b>	99.88	50.00	16.67	25.00	5.7832
<b>Numenta VM6</b>	100.00	0.00	0.00	0.00	4.8424
<b>Numenta VM7</b>	76.81	0.00	0.00	0.00	7.2695
<b>Numenta VM8</b>	100.00	50.00	100.00	66.67	5.0816
<b>Numenta VM9</b>	99.85	50.00	14.29	22.22	6.8909
<b>Average</b>	92.49	44.44	20.16	18.31	6.2860

\* Numenta VM7 is excluded from the average since there are no true positives and an f1 score cannot be calculated \*

Table 3 F1 scores (%) of Individual Detectors

VM Name	Average	Median	Boxplot	Histogram
<b>Numenta VM1</b>	50.0	50.0	0.0	0.0
<b>Numenta VM2</b>	0.1	0.2	0.5	66.7
<b>Numenta VM3</b>	0.1	0.1	0.0	0.0
<b>Numenta VM4</b>	0.0	0.0	0.3	0.0
<b>Numenta VM5</b>	25	25.0	8.2	0.0
<b>Numenta VM6</b>	0.0	0.0	4.3	0.0
<b>Numenta VM7</b>	0.0	0.0	0.0	100.0
<b>Numenta VM8</b>	66.7	50.0	1.8	0.0
<b>Numenta VM9</b>	50.0	50.0	9.1	0.0
<b>Average</b>	24.0	22.0	3.0	8.3

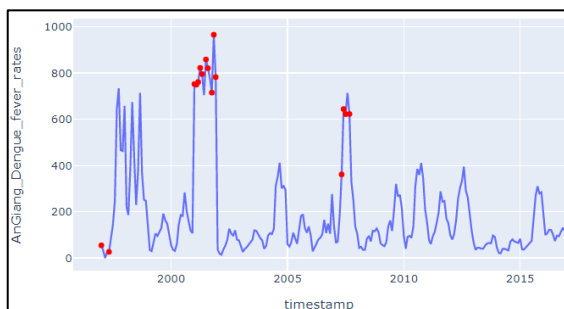
Table 4 Comparison of Voting Systems

Voting System	Accuracy	Recall	Precision	f1
<b>Combined Confidence</b>	86.51	50.00	27.53	24.46
<b>Majority Classification</b>	92.49	44.44	20.16	18.31

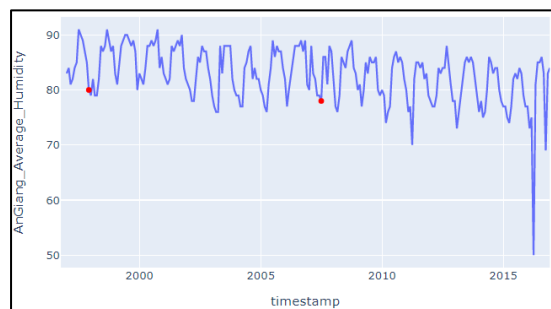
Table 5 VM Data Naming Convention

S/No	VM ID	VM NAME
<b>1</b>	ec2_cpu_utilization_5f5533	Numenta VM1
<b>2</b>	ec2_cpu_utilization_24ae8d	Numenta VM2
<b>3</b>	ec2_cpu_utilization_53ea38	Numenta VM3
<b>4</b>	ec2_cpu_utilization_77c1ca	Numenta VM4
<b>5</b>	ec2_cpu_utilization_825cc2	Numenta VM5
<b>6</b>	ec2_cpu_utilization_ac20cd	Numenta VM6
<b>7</b>	ec2_cpu_utilization_c6585a	Numenta VM7
<b>8</b>	rds_cpu_utilization_cc0c53	Numenta VM8
<b>9</b>	rds_cpu_utilization_e47b3b	Numenta VM9

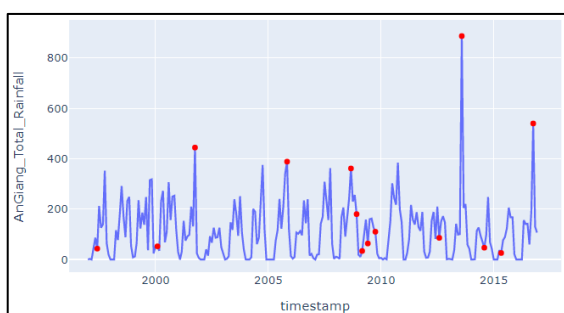
### 7.3 Dengue Fever Rates Experiment



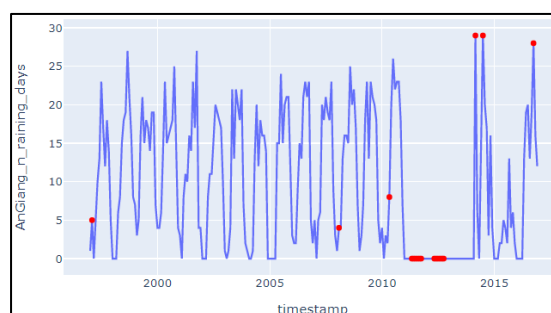
I. Detection result for An Giang Dengue Fever Rate



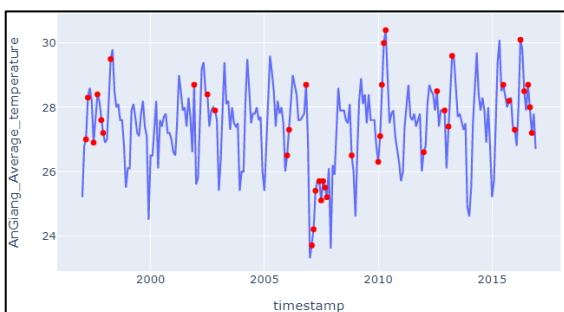
II. Detection result for An Giang Average Humidity



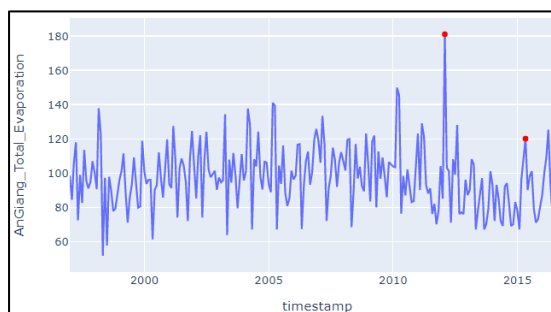
III. Detection result for An Giang Total Rainfall



IV. Detection result for An Giang No. Raining Day

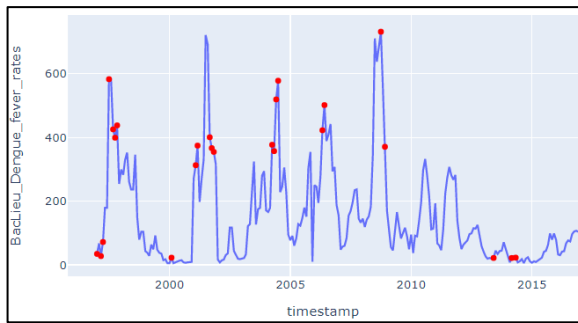


V. Detection result for An Giang Average Temperature

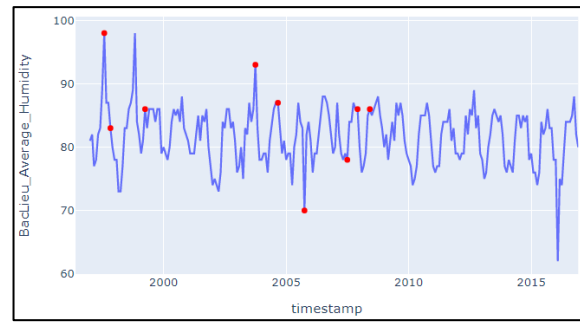


VI. Detection result for An Giang Total Evaporation

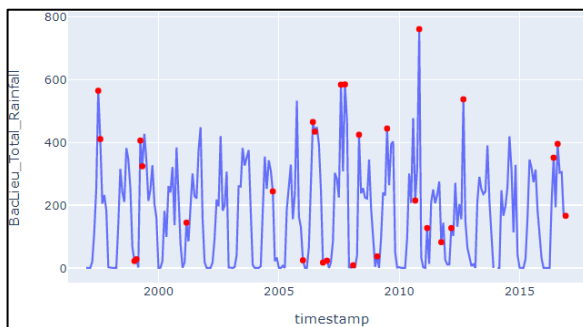
Fig. 1 ESMOD Detection on An Giang Dengue Fever Data



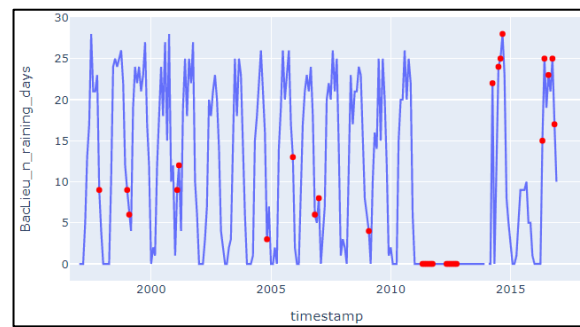
I. Detection result for Bac Lieu Dengue Fever Rate



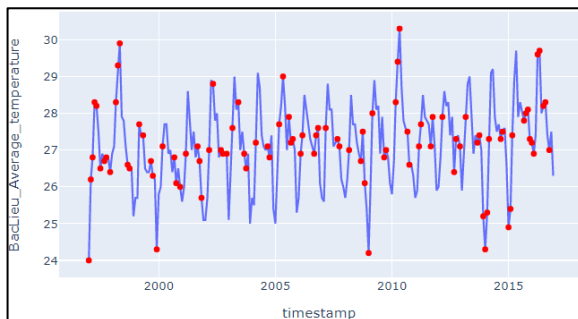
II. Detection result for Bac Lieu Average Humidity



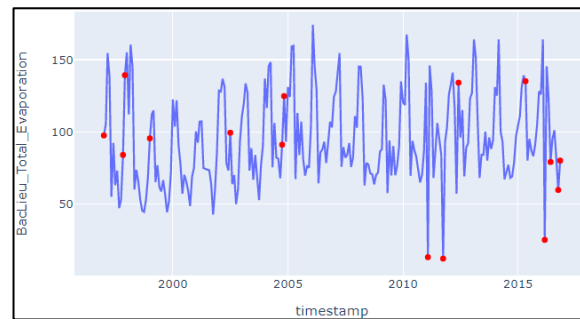
III. Detection result for Bac Lieu Total Rainfall



IV. Detection result for Bac Lieu No. Raining Day

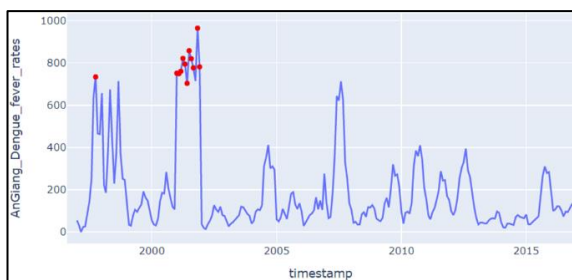


V. Detection result for Bac Lieu Average  
Temperature

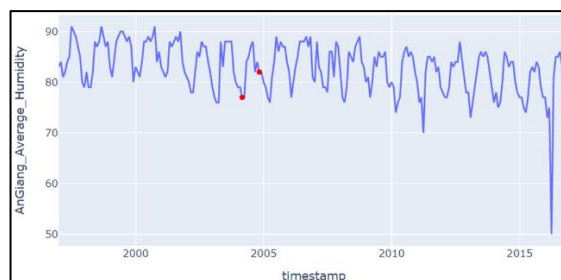


VI. Detection result for Bac Lieu Total Evaporation

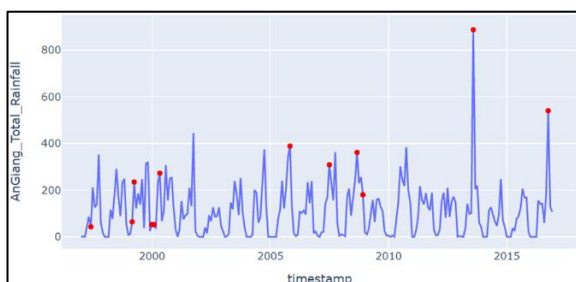
Fig. 2 ESMOD Detection on Bac Lieu Dengue Fever Data



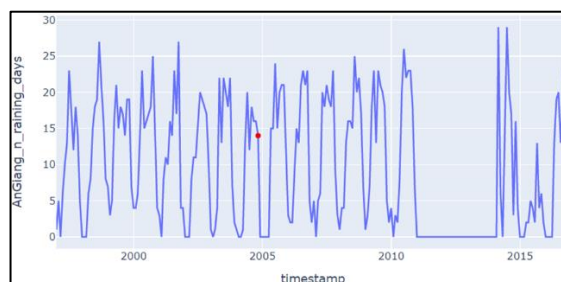
I. Detection result for An Giang Dengue Fever Rate



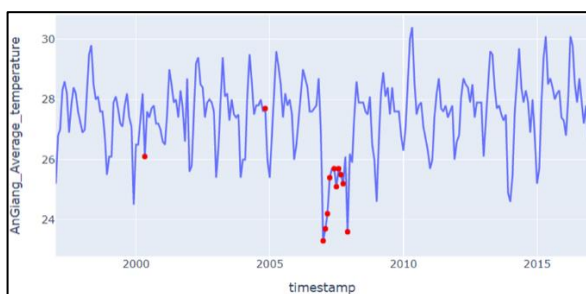
II. Detection result for An Giang Average Humidity



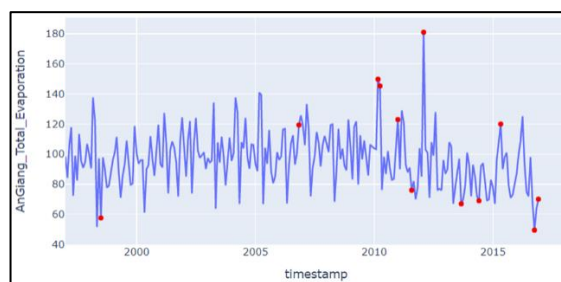
III. Detection result for An Giang Total Rainfall



IV. Detection result for An Giang No. Raining Day

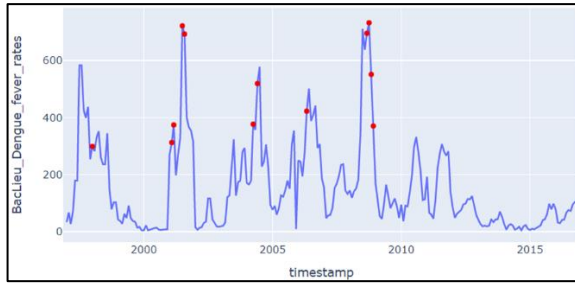


V. Detection result for An Giang Average Temperature

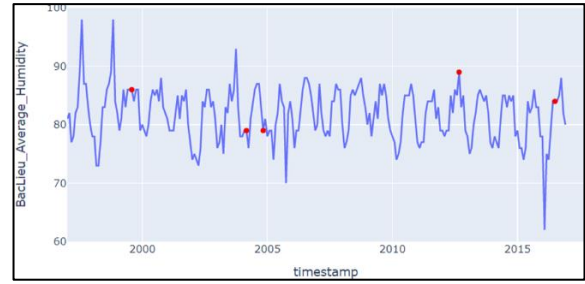


VI. Detection result for An Giang Total Evaporation

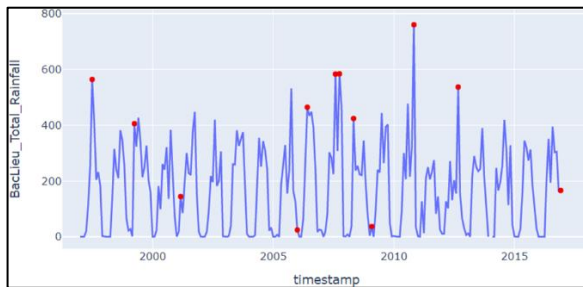
Fig. 3 KNN Outlier Detection on An Giang Dengue Fever Data



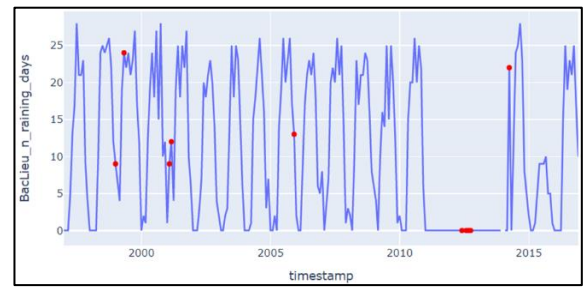
I. Detection result for Bac Lieu Dengue Fever Rate



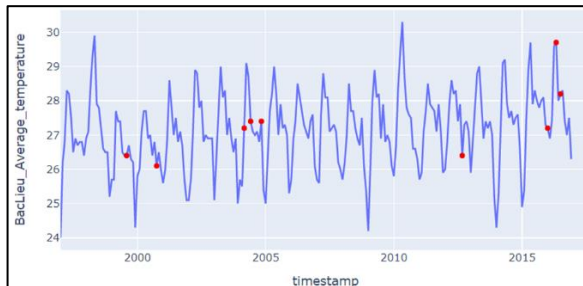
II. Detection result for Bac Lieu Average Humidity



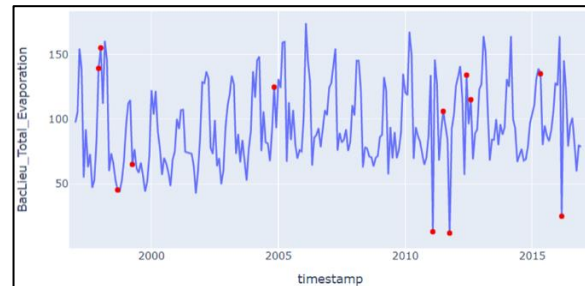
III. Detection result for Bac Lieu Total Rainfall



IV. Detection result for Bac Lieu No. Raining Day



V. Detection result for Bac Lieu Average Temperature



VI. Detection result for Bac Lieu Total Evaporation

Fig. 4 KNN Outlier Detection on Bac Lieu Dengue Fever Data