

Marshall University

Marshall Digital Scholar

Theses, Dissertations and Capstones

2023

Leveraging Explainable Artificial Intelligence (XAI) to Understand Performance Deviations in Load Tests of Large Software Systems

Eric Shoemaker

ericjamesshoemaker@gmail.com

Follow this and additional works at: <https://mds.marshall.edu/etd>



Part of the [Artificial Intelligence and Robotics Commons](#)

Recommended Citation

Shoemaker, Eric, "Leveraging Explainable Artificial Intelligence (XAI) to Understand Performance Deviations in Load Tests of Large Software Systems" (2023). *Theses, Dissertations and Capstones*. 1822. <https://mds.marshall.edu/etd/1822>

This Thesis is brought to you for free and open access by Marshall Digital Scholar. It has been accepted for inclusion in Theses, Dissertations and Capstones by an authorized administrator of Marshall Digital Scholar. For more information, please contact beachgr@marshall.edu.

**LEVERAGING EXPLAINABLE ARTIFICIAL INTELLIGENCE (XAI) TO UNDER-
STAND PERFORMANCE DEVIATIONS IN LOAD TESTS OF LARGE SOFTWARE
SYSTEMS**

A thesis submitted to
Marshall University
in partial fulfillment of
the requirements for the degree of
Master of Science
in
Computer Science

by

Eric Shoemaker

Approved by

Dr. Jamil Chaudri, Committee Chairperson

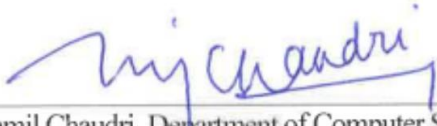
Dr. Husnu Narman

Dr. Haroon Malik

Marshall University
August 2023

Approval of Thesis

We, the faculty supervising the work of Eric Shoemaker, affirm that the thesis, *Leveraging Explainable Artificial Intelligence (XAI) to Understand Performance Deviations in Load Tests of Large Software Systems*, meets the high academic standards for original scholarship and creative work established by the Department of Computer Sciences and Electrical Engineering and the College of Engineering and Computer Sciences. This work also conforms to the formatting guidelines of Marshall University. With our signatures, we approve the manuscript for publication.



Dr. Jamil Chaudri, Department of Computer Sciences and Electrical Engineering Committee Chairperson

2023 May 04

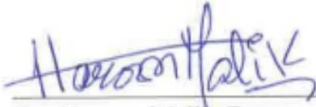
Date



Dr. Hushu Narman, Department of Computer Sciences and Electrical Engineering Committee Member

May 1, 2023

Date



Dr. Haroon Malik, Department of Computer Sciences and Electrical Engineering Committee Member

01-MAY-2023

Date

Acknowledgments

My thanks to Dr. Haroon Malik, my thesis supervisor, for his excellent guidance and contributions to this thesis.

Table of Contents

List of Tables.....	vi
List of Figures.....	vii
Abstract.....	ix
Chapter 1: Introduction.....	1
1.1 Problem Statement.....	2
1.2 Solution Overview.....	3
1.3 Contribution of the Thesis.....	4
1.4 Organization of Thesis.....	5
Chapter 2: Background Knowledge.....	6
2.1 Artificial Intelligence.....	6
2.2 Machine Learning.....	7
2.3 Black-Box Models.....	8
2.4 Explainable Artificial Intelligence.....	14
2.4.1 SHAP Framework.....	16
2.4.2 Shapley Values.....	17
2.4.3 Advantages and Disadvantages of Shapley Values.....	21
Chapter 3: Load Testing of Large-Scale Systems.....	24
3.1 Performance Monitoring of Large-Scale Systems.....	24
3.2 Load Testing.....	26
3.2.1 Steps Involved in Load Testing.....	27
3.2.2 The Current Practice of Load Test Analysis in LSS.....	28
Chapter 4: Explaining Performance Monitoring Classifiers With SHAP.....	34

4.1 Subject of Study and Environment Setup	34
4.1.1 The Open-Source System	34
4.1.2 Testbed	35
4.1.3 Description of Load	37
4.1.4 Fault Injection	37
4.1.5 Experiment Design.....	39
4.1.6 Organizing Load Test Results	44
4.2 Classifier Models	47
4.2.1 Classifier Model Training and Testing.....	49
4.2.2 SHAP Values for the Classifier Models.....	51
4.2.3 Updating the Classifier Models	55
4.2.4 Resulting Explanation of the Classifier Models	65
Chapter 5: Limitations of Work	67
Chapter 6: Conclusions and Future Work.....	68
6.1 Conclusion	68
6.2 Future Work	68
References.....	70
Appendix A: Approval Letter.....	74
Appendix B: Acronyms.....	75

List of Tables

Table 1	Examples of Common ML Algorithms	13
Table 2	Hardware Specifications of the Ecommerce Service System.....	35
Table 3	Baseline Load Test Configuration for Dell DVD Store (DS2).....	36
Table 4	Faults Injected into the Load Test Experiments	40
Table 5	Features of the Experiment Data Sets.....	46

List of Figures

Figure 1	Learning Process of Supervised ML Algorithms	8
Figure 2	Black-Box Model	9
Figure 3	Example of an Artificial Neural Network Architecture	11
Figure 4	Calculating the Contribution of the Total CPU Time Feature	19
Figure 5	Replacing Feature Values in x with Feature Values from r	20
Figure 6	Possible Coalitions When Calculating the Shapley Value for Total CPU Time.....	21
Figure 7	Steps Involved in Load Testing.....	27
Figure 8	A Sample Test Report	32
Figure 9	Timeline of Experiment #2.....	41
Figure 10	Timeline of Experiment #3.....	42
Figure 11	Timeline of Experiment #4.....	42
Figure 12	Timeline of Experiment #5.....	43
Figure 13	Timeline of Experiment #6.....	44
Figure 14	Architecture of the ANN classifier.....	48
Figure 15	Two Decision Trees from the Random Forest Classifier	49
Figure 16	Mean SHAP Values for ANN Model.....	52
Figure 17	Mean SHAP Values for Random Forest Model.....	52
Figure 18	SHAP Values Towards Predictions of Fault Identification.....	54
Figure 19	Correlation Heatmap of Testing Set.....	55
Figure 20	Mean SHAP Values Over Testing Set for Updated ANN Model	57
Figure 21	Mean SHAP Values Over Testing Set for Updated Random Forest Model	57
Figure 22	SHAP Values Towards Fault Identification During CPU Exhaustion.....	58

Figure 23	SHAP Values Towards Fault Identification During Memory Exhaustion	59
Figure 24	SHAP Values Towards Fault Identification During Unscheduled Replication	60
Figure 25	SHAP Values Towards Fault Identification During Interfering Workload.....	61
Figure 26	SHAP Values for a Random Instance During Interfering Workload	63
Figure 27	Boxplots of Network Traffic During Experiment 6	64

Abstract

Performance testing generates vast amounts of data, making it challenging for human analysts to process within a reasonable timeframe. Therefore, black-box machine learning models are often used to determine pass/fail status, but these models lack transparency and cannot explain why a test has failed, leading to a time-consuming manual analysis process. To address this issue, this thesis proposes using Explainable Artificial Intelligence (XAI) to improve the trustworthiness of black-box and interpretable models in performance testing. The proposed approach leverages the Shapley Additive Explanation (SHAP) algorithm as a surrogate model to help performance analysts understand the decision-making process of black-box machine learning models. By wrapping SHAP around black-box models, analysts can gain explainability on why a model predicted a test's pass or fail status and identify the relative importance of performance data to machine learning models. The proposed approach was evaluated through several load text experiments on a real testbed, using industry-standard performance benchmarks, manually injecting performance bugs into the system to synthesize ground truth and building machine learning models using a black box learner (Artificial Neural Network) and an interpretable learner (Random Forest) to predict the test's pass or fail status. The results demonstrate that classical performance measures such as precision, recall, F-measure, and accuracy are not sufficient to gauge the reliability and trustworthiness of machine learning models. Instead, the proposed approach stands out by providing the explanations behind the decisions made by learning algorithms and enhancing their trustworthiness. The proposed approach, though evaluated through load testing can be generalized to other domains and require little or no effort to operate.

Chapter 1: Introduction

Companies are faced with immensely large volumes of data which outpace humans' ability to absorb and interpret the data, making it increasingly difficult to make informed business decisions from such data. Therefore, being able to efficiently utilize big data to draw insights and inform decisions can give companies a competitive edge [1]. This has caused massive growth in the field of artificial intelligence (AI), which is able to recognize patterns, forecast events, classify data, and much more even on a large scale. This growth in AI development has led towards the usage of AI in nearly every domain, and we see this in the form of social media engines, chatbots, navigational devices, etc.

One domain which can greatly benefit from the usage of AI is load testing analysis of large-scale systems (LSS). Due to the rapid growth in size, complexity, and demand for practical performance monitoring solutions, the performance monitoring of LSS has become increasingly critical and challenging. Due to the predictive power of modern-day AI, applying AI to analyze the performance of large-scale systems has become increasingly useful. Such AI systems can monitor systems 24/7 to identify the performance of the system, and this can also be useful for the analysis of load testing, which is assessing the performance of a system under a given load. The load refers to the rate at which transactions are submitted to the system.

Despite the usefulness of AI in solutions towards load testing and other domains, there has been rising concern regarding the trustworthiness of AI. As AI continues to improve and evolve, the underlying machine learning (ML) algorithms which make up AI applications have grown in complexity. Some modern ML algorithms, such as the artificial neural network (ANN) and the random forest, are able to learn patterns in specific data sets. Although this has increased the predictive power of these ML algorithms, they have become much less interpretable. This leaves users

with only a few methods of ensuring the quality of AI model. One method is to hire a domain specialist to analyze the AI model. Although this method is useful, it is a costly and time-consuming process, and depending on the domain, domain-specialists may be hard to find. Another method is to compare model predictions to the ground truth, i.e., comparing current predictions to past historical data. However, this method requires much domain knowledge from the user and can require the analysis of vast quantities of data. The third method is evaluating the AI with performance metrics, such as the prediction accuracy, the amount of error present in predictions, etc. This is often how users evaluate the quality of AI. However, this method is not case specific, and it does not explain what went wrong when AI provides an unsatisfactory prediction.

1.1 Problem Statement

Performance testing, particularly load testing of large-scale systems, can generate a significant volume of performance data, including CPU utilization, memory consumption, network utilization, disk Input/Output operations per second, and packet latency from hundreds of machines. With such a vast amount of data, it becomes impossible for human performance analysts to skim through thousands of metrics. Besides, performance testing is often the last step in an already delayed release schedule, leaving analysts with minimal time to complete their work. Consequently, analysts typically use black-box machine learning models to quickly determine whether hundreds of performance tests pass or fail. However, if the test fails, it is crucial to understand the reason for the failure to repeat it successfully. While the black-box model can determine with a certain degree of accuracy and precision whether a test passes or fails, it cannot provide an explanation for why it failed or passed. Therefore, analysts must manually analyze the failed load test to understand the root cause of its failure, which is a tedious task. As a result, there is an immense

and utmost need for intelligent techniques that can explain the opaque black-box model's decision to analysts and speed up the load testing process.

1.2 Solution Overview

To address the problem identified in Section 1.1, this thesis aims to explore the concept of using XAI to improve the trustworthiness of both black-box and interpretable models, particularly in the context of load testing analysis of numeric performance data—such as percent processor time, available megabytes of memory, milliseconds of latency, etc.—and for most of the black-box machine learning models. The thesis proposes using the Shapley Additive Explanation (SHAP), which is an XAI technique based on cooperative game theory, as a surrogate model to help analysts understand the decision-making process of their underlying complex black-box machine learners, such as Neural Networks, Support Vector Machines, and Naïve Bayes.

By using the proposed approach, i.e., SHAP as a wrapper around black-box models, the performance analysts can gain explainability on why their black-box model predicted the passing or failing of a test. Moreover, it facilitates the performance analysts in explaining the reasoning behind the decisions made by the underlying machine learning algorithms by pinpointing the relative importance of the performance data to the results and prediction of their machine learning opaque models. The proposed approach is expected to enhance the trustworthiness of the black-box models used for performance testing and help analysts quickly identify the root cause of the failure, thereby speeding up the load testing process. Chapter 4 provides the details of the proposed solution.

1.3 Contribution of the Thesis

The aim of this thesis is to demonstrate how the XAI algorithm SHAP can improve the trustworthiness of black-box classifiers that identify failures in load tests of large-scale systems.

This thesis makes several contributions to the field of XAI and performance testing:

- 1) The thesis contributes to the XAI literature by demonstrating the successful application of SHAP in the performance testing domain.
- 2) The thesis is the first to demonstrate that using SHAP as a surrogate model can enable performance analysts to explain the reasoning behind the decisions made by underlying machine learning algorithms regarding the passing or failing of load tests.
- 3) Unlike previous studies that used synthetic data or log traces from the internet, this thesis conducts experiments on a testbed that uses industry-standard performance benchmarks.
- 4) Rather than simulating failures to match real-world performance problems, we use open-source tools and manually inject faults into the system to evaluate the trustworthiness of proposed techniques.
- 5) To ensure the replicability and fair comparison of our results, each experiment is presented as a Jupyter Notebook that is accessible to the research community upon request.

Overall, this thesis provides practical insights into how XAI can be used to improve the transparency and reliability of black-box classifiers in the performance testing domain, which can benefit both the research and industrial communities.

1.4 Organization of Thesis

This thesis is organized into a total of 6 chapters:

- I. **Chapter 1** introduces the thesis topic and discusses the objective and contributions of this work. Along with this, it outlines the organization of this thesis.
- II. **Chapter 2** provides background knowledge on the topics of artificial intelligence, machine learning, and black-box models. Along with this, it provides an overview of XAI and discusses its importance. One XAI method, SHAP, is selected to use in the experiment performed in Chapter 4, so an overview of the workings of SHAP is also provided.
- III. **Chapter 3** discusses the load testing of large-scale systems. This includes details of how performance monitoring of large-scale systems is completed, and the many difficulties faced.
- IV. **Chapter 4** details an experiment which demonstrates the usage of SHAP for explaining black-box classifier predictions. The classifier models are used to identify faults during the load testing of large-scale systems.
- V. **Chapter 5** discusses several limitations of this work, such as those presented by the test environment utilized in the experiment in Chapter 4.
- VI. **Chapter 6** briefly summarizes the thesis work and conclusions.

Chapter 2: Background Knowledge

2.1 Artificial Intelligence

Companies are currently faced with a wider variety and larger volume of data than ever before, and being able to efficiently manage and utilize big data can give a company a great competitive edge [1]. Analyzing big data allows companies to draw insights from past events and records, and these insights can be used to inform better future investments and business decisions of any kind. However, properly utilizing and learning from such enormous amounts of data is no simple task. With modern demands, the immense quantities of data far outpace humans' ability to absorb interpret such vast data quantities [1].

Therefore, the usage of artificial intelligence has become foundational in nearly every industry. Artificial intelligence (AI) is the ability of a computer to replicate, enhance, or even surpass human intellect [2], and it can be used to process large amounts of data to accomplish many different tasks such as recognizing patterns, forecasting events and probabilities, and even classifying data.

There are several factors which have driven the adoption and development of AI across different industries. AI's ability to provide insights on vast quantities of data to inform business objectives has been the driving factor of its popularity. With the modern amounts of processing power and storage that is widely available, AI can help businesses to make better decisions faster through observations made on immense amounts of data. Also, AI has become useful in increasing business efficiency in other contexts as well. AI is being used to amplify the worker's capabilities, to automate mundane tasks, and to provide 24/7 monitoring of activities.

AI has seen such rapid development, and it now sees usage not only in the workplace but also in many day-to-day activities. Many applications such as social media engines, text editors,

chatbots, navigational applications, web searching applications, self-driving vehicles, etc. have taken advantage of an AI component. Because of this, AI has become a prevalent part of the everyday life of many individuals.

2.2 Machine Learning

Despite the widespread usage of AI in many domains and in everyday life, knowledge of the underlying algorithms that make up AI are not often understood by the average end users of AI systems. These underlying algorithms are part of the field of machine learning. Machine learning (ML) is a subcategory of AI that uses statistical algorithms to imitate the human learning process [2]. This learning process of humans typically consists of several steps. When faced with a problem and a choice, humans will use their own knowledge in an attempt to choose the best possible solution. Sometimes they are successful, and sometimes they are not. After seeing the outcome of their decision, humans can determine whether or not that decision was correct in solving the problem. If not, they must adjust their approach to the problem using their newfound knowledge from the failed attempt.

Similar to the human learning process, ML algorithms consist of several parts [3], referred to as the (1) decision algorithm, (2) error function, (3) model optimization process, as shown in Figure 1.

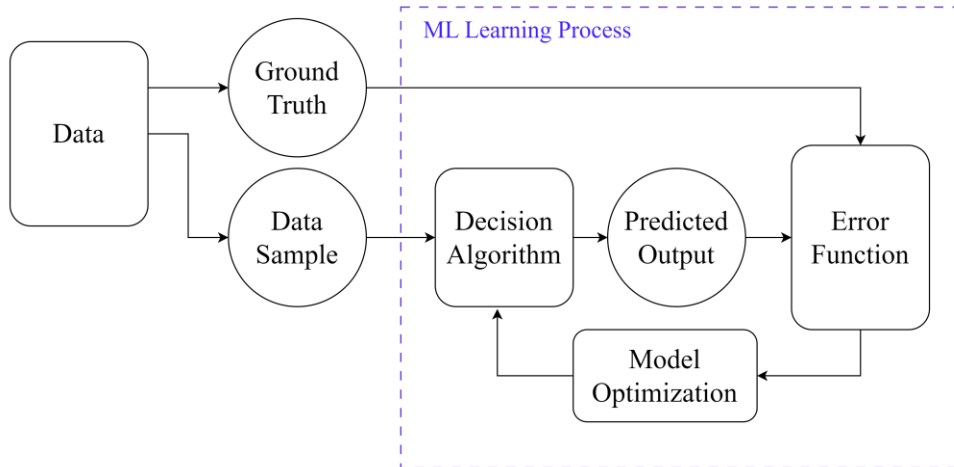


Figure 1: Learning Process of Supervised ML Algorithms

1. **Decision Algorithm:** Given a data sample, the decision algorithm attempts to make a correct decision through utilizing a statistical algorithm. This algorithm views the problem and produces its best solution (the predicted output).
2. **Error Function:** The error function evaluates the solution proposed by the decision algorithm by determining the difference between the ground truth and the proposed solution.
3. **Model Optimization Process:** The model optimization process adjusts the decision algorithm based on the result of the error function. This process aims to reduce the amount of error present in the decision algorithm.

Through these steps, the ML algorithm can learn how to find the best solution to the problem through a series of trial-and-error.

2.3 Black-Box Models

Although the general learning process of ML algorithms is a simple concept to understand, the specifics of each step in the process can become increasingly complex depending on the type

of ML algorithm being used. Since ML algorithms make up the AI being used in data analytics, decision making, and automation in many industries, the boundaries of what these ML algorithms can accomplish are constantly being pushed to new heights, and developing the most efficient and powerful ML algorithms has become a primary focus of data analytics research. This has resulted in many positives. Modern ML algorithms can produce predictions with high accuracy, and they can do so even with little or no human interference. This is due to highly complex decision algorithms that are able to adjust themselves to learn specific data sets.

Despite the overall gains achieved from these complex decision algorithms, they have also produced an issue. The complexity of a decision algorithm can exceed explainability once it has learned a specific set of data. Most people—even those with high-levels of knowledge of machine learning algorithms—cannot always explain the rationale, i.e., why the decision algorithm makes a specific prediction. AI models which utilize unexplainable ML algorithms are referred to as black-box models. These models take input(s) and provide output(s). However, the inner mechanisms used to reach to the output (i.e., decision) are not seen or understood. Figure 2 illustrates this concept of a black-box model.

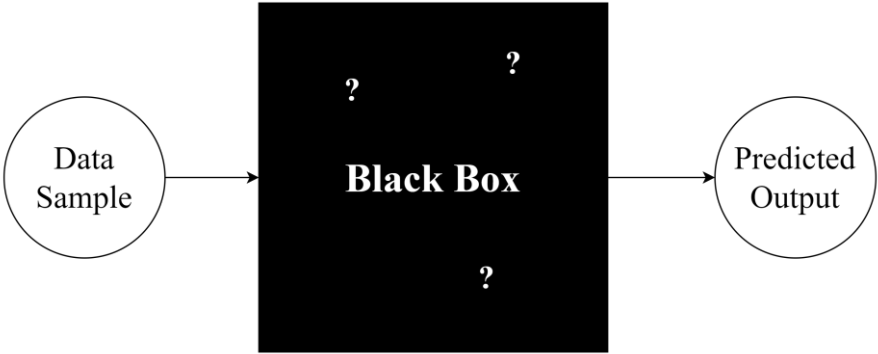


Figure 2: Black-Box Model

The term "black box" suggests that the model's internal processes are opaque, and the output is solely dependent on the input. In the context of machine learning, a black box model refers to a model whose internal workings are not transparent or easily interpretable by humans. Especially when the black box model is composed of several machine learning techniques or surrogate models that work with some degree of interdependency among them to produce output, i.e., result or prediction. Even if some of these complex models can be interpreted it will take quite a human effort which some time is infeasible in large environments where these models are continuously evolving over time.

In other words, we know what goes into the model and what comes out of it, but we do not know what happens inside the model. This makes it difficult to understand how the model makes decisions, and to analyze and interpret the reasons behind its predictions or recommendations.

To better understand what makes a black-box model unexplainable, consider an example. One type of black-box ML model is an artificial neural network (ANN). ANNs are designed to mimic the human brain, and they are composed of many neurons. These neurons are arranged into several layers, and each neuron has connections—each represented by a numeric value called a weight—to neurons in the previous layer. An example of an ANN architecture with 3 layers is shown in Figure 3.

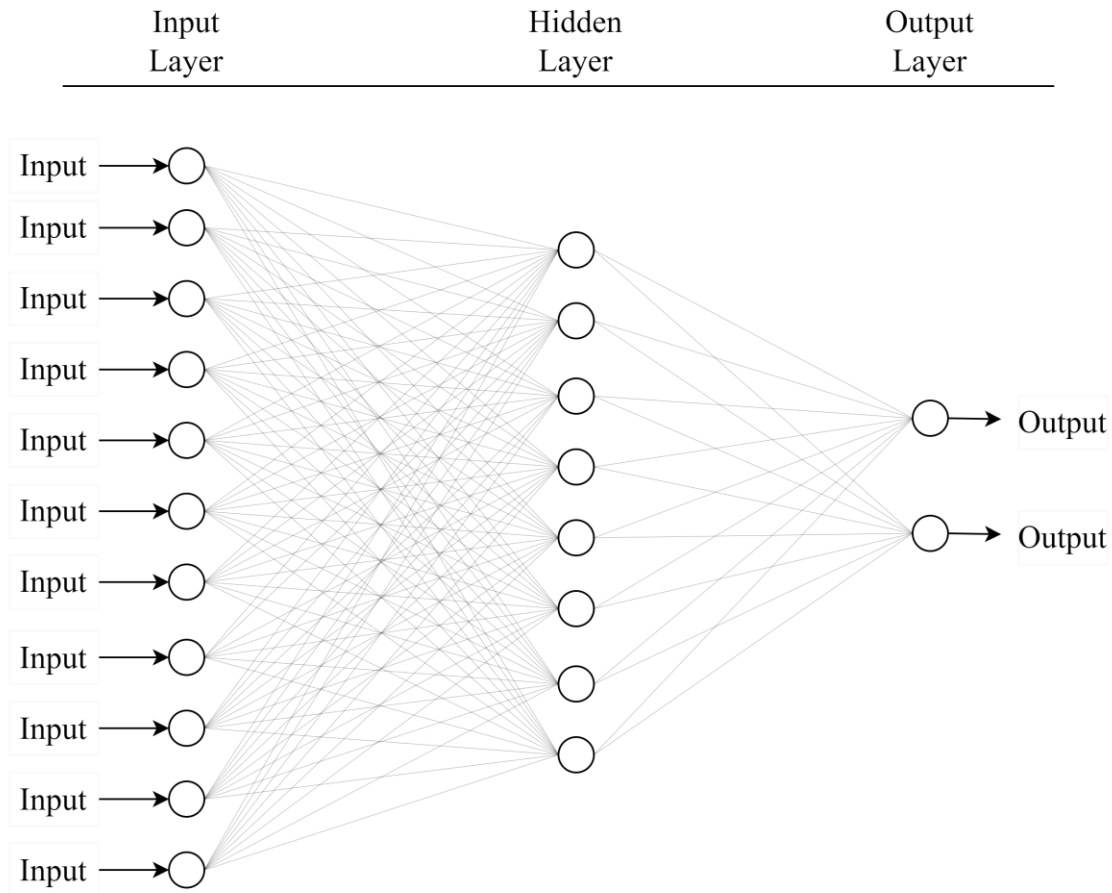


Figure 3: Example of an Artificial Neural Network Architecture

Inputs are given to the neurons in the input layer. The data is processed with a mathematical function within each neuron, and the results are passed onto the next layer through the connections. The weight of each connection transforms the results as they are passed to the next layer. When the next layer receives the data, the process repeats. The neurons in this layer perform a mathematical function and pass the results to the next layer through the connections, and once again, the weights affect the value of the data being passed. This process repeats at each layer until reaching the end of the network, and the values output by the neurons in the final layer are considered to be the predicted output of the ANN's decision algorithm.

This predicted output is then analyzed with an error function, which determines the degree

of error between the predicted output and the ground truth. Once the error is calculated, the model optimization process alters the numeric values of the weights such that the predicted output becomes closer to the ground truth. In this way, we can understand the workings of an ANN.

However, consider a trained ANN that has updated its weights many times through the process of learning on a specific data set. Although humans may understand the learning process that the ANN utilizes, humans are not able to explain the reasoning behind the numeric weight values of the trained ANN. During training, the network has identified patterns which relate inputs to outputs in the data, making it challenging to understand how neurons interact with input data patterns. Therefore, ANNs are considered black-box models.

There are many ML algorithms that exhibit this opaqueness like the ANN. To provide further examples, a few commonly used ML algorithms are listed and classified as black-box or not in Table 1.

ML Algorithm	Description	Black-Box (Yes/No)?
Linear Regression	Linear regression predicts an output value using a linear equation that relates input independent variables to output-dependent variables.	No. Linear regression identifies the strength of independent variables' effect on the predicted dependent variable through coefficients in the linear equation.
Logistic Regression	Logistic regression predicts an output using a logistic equation that relates input independent variables to the output-dependent variables.	No. Logistic regression identifies the strength of independent variables' effect on the predicted dependent variable through coefficients in the logistic equation.
Decision Tree	A decision tree predicts a value by splitting all the data across features to minimize a cost function. This is accomplished through a graph data structure that represents choices and their results as a tree; nodes represent attributes in a group, and branches represent values that the node can take [4].	No. The decision-making process is outlined in the graph-structure of a decision tree. One can follow the exact process through the graph to understand why specific decisions are made.
Neural Network (ANN, CNN, RNN)	A neural network is a model that attempts to identify underlying relationships in a data set through a process mimicking how the human brain learns information [4].	Yes. The weights of a neural network cannot be understood after training. The values were set from data patterns identified by the network.
Weighted Random Forest	A weighted random forest is a model that generates a number of decision trees (each with weight(s)), and each tree produces a prediction on input data; the result is a combination of all predictions (considering weighting) into a final prediction [3].	Yes. After a weighted random forest has been trained, the logic behind the weight values is not understandable.

Table 1: Examples of Common ML Algorithms

2.4 Explainable Artificial Intelligence

Due to the predictive power of black-box models, ML algorithms such as the ANN continue to see more widespread use and implementation into AI applications. Since AI has become more instrumental in many industries and more prevalent in everyday life through its usage in applications (such as social media platforms, web-searching applications, email services, advertisements, etc. [5]), this has led to a major concern. Although many people are accustomed to using applications with AI, many users have limited understanding of the decision-making process of this AI that they interact with each day. Some users may lack knowledge of AI implementation entirely, but even domain specialists and programmers alike cannot explain the decision-making process of black-box models. This can hamper the trust between the AI and the user.

Although this issue is a cause for concern in all cases involving AI decisions, the severity can vary depending on the specific situation. In many common cases of end-user to AI interaction—such as allowing AI from Netflix to decide which movie recommendations should be shown—lacking an understanding of the decision-making process behind-the-scenes has little importance to the end user. This is because an incorrect decision by the AI will have little or no consequence or impact on the end user.

However, there are many scenarios where AI decisions impact an individual's life much more. Consider an AI decision made in the field of healthcare. A doctor may utilize AI to help determine an important disease diagnosis. In this case, an incorrect diagnosis by the AI could incorrectly inform the doctor, which can result in improper treatment of the patient. An event such as this could majorly impact the patient's life for the worse. In this scenario, the doctor will likely exhibit concern towards the AI's decision-making process, and he/she may doubt the trustworthiness of the AI system.

Typically, the doctor could take several approaches towards evaluating the AI system to ensure that it is trustworthy. One approach is to hire a domain specialist to analyze the model and the incorrect prediction in question. The domain specialist may examine past predictions and patient diagnoses to determine what may have caused the AI to give the incorrect prediction in this case. However, this is a lengthy and costly process, and domain specialists may be in high demand depending on the field.

Another method is to utilize evaluation metrics such as accuracy or loss. The accuracy of tells the ratio of correct to incorrect decisions made, and the loss gives the magnitude of error present in predictions. These two metrics are often used to show the precision of an AI model, and showing high precision intends to help users trust the predictions.

However, these approaches may not be sufficient at ensuring user trust in an AI system. Many AI users are currently reassessing the basis for trust in AI, as there is growing concern about incorrect AI decisions and their potential consequences. Even seemingly correct decisions are now being scrutinized for potential model bias and ethical concerns, particularly in opaque models where discrimination and other issues may go unnoticed. For instance, decision-making processes that have a history of prejudice, such as the determination of bail [6], have been the focus of extensive efforts to eliminate racial discrimination through AI. However, there is a concern that these systems may unintentionally perpetuate discrimination due to biased data used for their training.

Due to these rising concerns about trust in AI, there has been much growth in interest in explainable artificial intelligence throughout academia, industry, and government. Explainable artificial intelligence (XAI) proposes creating ML techniques that can produce more explainable and transparent models while maintaining high performance, and it also aims to provide diverse methods of post-hoc explainability to existing ML.

2.4.1 SHAP Framework

Regarding post-hoc explanations of the decision-making process of black-box models, the field of XAI considers many different approaches. Some approaches utilize specific examples from the overall data to try and explain the behavior of an ML model. Other methods can try to explain the model’s decisions by perturbing input data to see how the model reacts. Among these methods, two popular approaches are sensitivity analysis (analyzing how ANN output is affected by perturbations of inputs or model weights [7]) and feature importance (permuting inputs to determine the contribution of different features of data towards predictions [7]). The approach of feature importance has seen rising popularity, as this approach often produces visual explanations that can be easily understood by humans. Within this approach, there are several different XAI methods which can be applied, such as Local Interpretable Model-agnostic Explanations (LIME), and SHAP.

This thesis will utilize **SH**apley **A**dditive **eX**planations (SHAP), which is both a post-hoc and model-agnostic method of explaining individual predictions. SHAP aims to provide a linear model which explains the marginal contribution of each feature towards a model prediction. SHAP proposes that the SHAP values (representing the marginal contribution) for each feature of a data instance x can be represented as a linear model g [8]. The linear model is shown in Equation 1. ϕ_j represents the SHAP value for feature j , and M represents the total number of different features in the feature space.

$$g(x') = \phi_0 + \sum_{j=1}^M \phi_j$$

Equation 1: Linear Explanation Model for SHAP

Although explanations from the SHAP framework are in the form of a linear model of Shapley values—as shown in Equation 1—the SHAP framework considers that the decision-making process of many modern ML algorithms will identify complex patterns and utilize non-linear statistical methods. For explaining these models, the KernelSHAP technique is used. This technique is based on both Shapley values and LIME. The goal is to weight the perturbed instances by their proximity to the original instance being explained and then to fit a weighted linear model through linear regression to arrive at the final explanation.

2.4.2 Shapley Values

The logic utilized by SHAP is based on the concept of Shapley values, which were introduced by Lloyd Shapley in 1951 [9]. Shapley values were proposed as a solution in cooperative game theory in order to find the marginal contribution of different team members towards an outcome, and we can apply the concepts employed by Shapley to find the mean marginal contribution of each feature in the feature space towards a model's predictions [10].

The calculation of a Shapley values can be a complex process which involves multiple integrations [8]. By definition, a Shapley value is the mean marginal contribution of a feature towards the predicted outcome, weighted and summed over all possible combinations of features [8]. To better illustrate the concept of Shapley values, consider a simplified example similar to the experiment discussed in Chapter 3. In this simplified example, a random forest classifier, denoted by f , is used to identify abnormal computer behavior, such as a memory leak or excessive CPU stress. The random forest classifier monitors the computer by receiving 4 metrics as inputs: (1) total CPU time, (2) disk idle time, (3) amount of memory consumption, (4) internal temperature.

At one point in time—we will refer to this as instance x —the computer logs the metrics:

- CPU Time: 84%
- Memory Consumption: 2378 MB
- Disk Idle Time: 94%
- Internal Temperature: 82° F

The random forest predicts a probability of 0.87 towards identifying abnormal behavior for instance x , and it alerts the user of the computer that abnormal behavior is present. In this instance, the user may not notice any abnormal behavior from the computer, so he/she wants to know why the random forest classifier came to its conclusion by seeing the marginal contribution of each feature.

For this example, we can demonstrate the steps finding a Shapley value for one feature—total CPU time. We start by demonstrating how the contribution of the total CPU time feature is calculated. Consider Figure 4, where the total CPU time is removed from the coalition. We can see the classifier's prediction changes from 0.87 to 0.74 when the total CPU time is removed, so for this specific coalition, the contribution of the total CPU time is the difference between predictions, $0.87 - 0.74 = 0.13$.

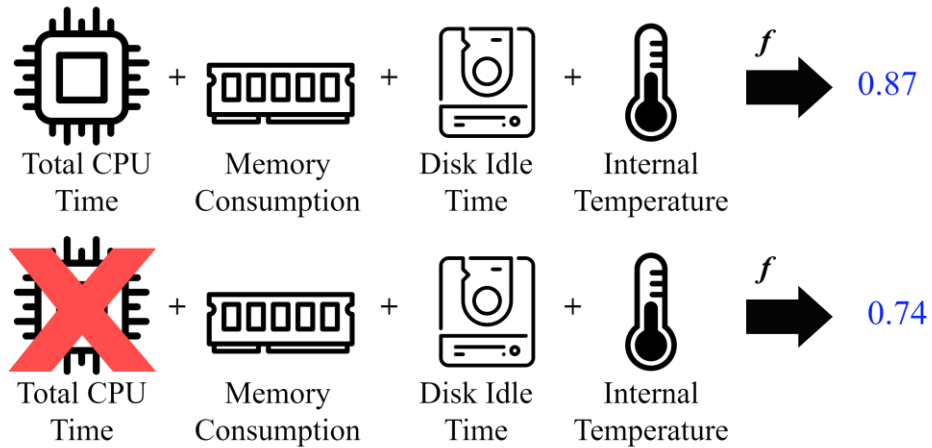










Figure 4: Calculating the Contribution of the Total CPU Time Feature

The process of removing a feature value is not simply setting the value to 0; rather, the feature value for the total CPU time was replaced by the feature value of another random sample in the sample space. If we represent the random instance as r , examples of replacing some feature values in x is shown in Figure 5.

Instance x		Instance r	
	= 84%		= 16%
	= 2378 MB		= 2190 MB
	= 94%		= 98%
	= 82° F		= 52° F

$$\begin{aligned}
 & \text{CPU} + \text{RAM} + \text{Hard Drive} + \text{Temp} = 84\% + 2378\text{MB} + 94\% + 82^\circ\text{F} \\
 & \text{CPU} \times + \text{RAM} + \text{Hard Drive} + \text{Temp} = 16\% + 2378\text{MB} + 94\% + 82^\circ\text{F} \\
 & \text{CPU} \times + \text{RAM} \times + \text{Hard Drive} + \text{Temp} = 16\% + 2190\text{MB} + 94\% + 82^\circ\text{F} \\
 & \text{CPU} \times + \text{RAM} + \text{Hard Drive} \times + \text{Temp} = 16\% + 2378\text{MB} + 94\% + 82^\circ\text{F} \\
 & \dots \\
 & \text{CPU} \times + \text{RAM} \times + \text{Hard Drive} \times + \text{Temp} \times = 16\% + 2190\text{MB} + 98\% + 52^\circ\text{F}
 \end{aligned}$$

Figure 5: Replacing Feature Values in x with Feature Values from r

In this way, we can replace features values in x with feature values from r . Now, reconsider the definition of Shapley values. To find a Shapley value, all possible combinations need to be considered, so we need to calculate the contribution of total CPU time for every possible coalition. This is accomplished by taking the difference between predictions with the total CPU time feature value present and replaced, and this difference is calculated for each possible coalition. The possible coalitions are shown in Figure 6.

























 +  + 	 +  + 
 +  + 	 +  + 
 +  + 	 +  + 
 +  + 	 +  + 

Figure 6: Possible Coalitions When Calculating the Shapley Value for Total CPU Time

Now, we have seen the process of calculating the contribution of total CPU time for every coalition between x and a random instance r , but for the Shapley value to be representative of the entire data space, this process should be repeated with every other data instance in the data space sampled for r . Then, the Shapley value is simply found as the mean of all the contributions.

2.4.3 Advantages and Disadvantages of Shapley Values

The Shapley value has several advantages which distinguish it from other explainable AI methods, such as Local Interpretable Model-Agnostic Explanations (LIME). One primary advantage is the efficiency property of Shapley values, which guarantees that the difference between a model prediction and the average prediction is fairly distributed among all the features in the instance [8]. The efficiency property is shown in Equation 2.

$$\sum_{j=1}^M \phi_j = f(x) - E_X(f(X))$$

Equation 2: Efficiency Property of Shapley Values

Here, $f(x)$ is the model prediction for instance x , and $E_X(f(X))$ is the mean model prediction over all instances in the instance space X . Therefore, the efficiency property of Shapley values ensures that the feature contributions for x add up to the difference of predictions between x and the average prediction overall. Due to this property, explanations utilizing Shapley values have developed a reputation for being the most trustworthy explanation methods, and they have become the focus point of using XAI in legal situations—such as providing trustworthiness for AI in compliance with EU’s Artificial Intelligence Act [11].

Despite the usefulness and advantages of Shapley values, there are several disadvantages of using Shapley values. The most important disadvantage to consider is the computing time and resources necessary for calculating Shapley values. In Section 1.4.2, we saw the calculation of the contributions of one feature using only one random sample. This process would have to be completed for all features and considering all instances for sampling. In many real-world situations with many features and instances to consider, calculating Shapley values takes too much time, Shapley values can be approximated to save computing time and resources [8]. This helps improve runtime but can compromise the accuracy of Shapley values. For Shapley values to be precise, as many random instances from the data space should be sampled and used as possible.

Another disadvantage of Shapley values is that they cannot provide selective explanations [8]. To uphold the efficiency property, Shapley values always consider all features in the feature space. In scenarios with many features, this can affect how interpretable Shapley values are to humans. Many humans will prefer a selective explanation, showing the most influential features rather than all features. In these cases, it may be best to use another XAI method such as LIME or SHAP (based on Shapley values but can allow for explanations with few features).

Another disadvantage is the necessity of access to the data. When calculating Shapley values for an instance, the instance is compared to other coalitions using feature values from other random instances. Some methods of estimating Shapley values may include creating instances similar to those in the data if the data is not directly available. For example, one may use Monte Carlo Sampling to create random instances whose features can be used for substitution.

Chapter 3: Load Testing of Large-Scale Systems

3.1 Performance Monitoring of Large-Scale Systems

Today's large-scale systems (LSS), which consist of data centers and server farms, have undergone a significant increase in size and complexity. For instance, Google alone has over a million servers in its data center, while Facebook has doubled the size of its data center within a year, from 60,000 to 120,000 servers. Similarly, Microsoft, eBay, Yahoo, and Amazon collectively have over 550,000 servers.

LSS must be monitored continuously to ensure high availability and peak performance. These systems represent large capital investments for service providers, and any performance issues can result in significant monetary losses. For example, a PayPal outage of just one hour may prevent up to \$7.2 million in customer transactions [12]. Therefore, operators of LSS closely monitor their systems to detect performance problems before they violate service level agreements (SLAs) or cause unplanned system downtime that can cost as much as \$550,000 per hour [13], [14].

Performance problems in LSS can manifest as high response times, increased latency, low throughput underload, or not meeting desired SLAs. The occurrence of performance bugs in these complex systems has become a common issue, rather than an exception. Service providers, like AT&T and Research in Motion, have also reported concerns about performance degradation and resource saturation. For example, a robust performance monitoring system could have alerted the operators of Skype before a system overload caused a 48-hour service disruption for millions of users.

Availability or performance issues can have a significant impact on business-customer re-

relationships. In the growing market of Software as a Service (SaaS), competitors can quickly capitalize on dissatisfaction and lure customers away. In some cases, costly lawsuits or even the breakdown of the business model can result from a lack of trust in the quality of service (QoS). To maintain customer trust, these LSS must be continuously monitored and actions must be taken immediately when low availability or poor performance threatens QoS.

Due to the rapid growth in size, complexity, and demand for practical performance monitoring solutions, the performance monitoring of LSS has become increasingly critical and challenging. Despite limited progress in research on automatic performance monitoring and analysis, load testing remains the most important part of performance testing for LSS. Research on large industrial projects has shown that the primary problems in the field are performance-related. Performance analysts use load testing to detect early performance problems in the system before they become critical field problems.

During a load test, which may last several days, one or more load generators simulate thousands of concurrent transactions. The application under test is closely monitored, and a large volume of performance counter data is logged. This information, such as CPU utilization, disk I/O, queues, and network traffic, is used to observe the system's behavior under load. This behavior is then compared to documented behavior from past similar tests, or it is used to inform decisions on expected behavior based on domain knowledge and experience. However, current load test analysis practices are labor-intensive, time-consuming, and prone to errors. While there has been some research on automatically generating load test suites, little attention has been given to effectively analyzing the results of load tests.

3.2 Load Testing

Load testing is a crucial process in assessing the performance of a system under a given load. The load refers to the rate at which transactions are submitted to the system. It helps to identify any residual functional and performance issues that may have escaped the functional testing process.

Functional problems can be bugs such as deadlocks and memory leaks, whereas performance problems can result in the system freezing, crashing, or becoming unresponsive during a load test. Symptoms of performance problems include high response time and low throughput underload. Performance analysts and developers spend significant time fixing these issues, which can have a major impact on the overall functioning of the system.

For example, post-release problems such as performance degradation and resource saturation are a common concern among companies such as AT&T and Research In Motion. These problems can result in a subsystem of the system failing, potentially leading to monetary losses. For instance, a PayPal outage that lasted an hour during periodic maintenance could have prevented up to \$7.2 million in customer transactions.

Load testing is an effective tool to uncover such functional and performance problems by simulating thousands of concurrent transactions to an application under test using one or more load generators. The application is closely monitored during the load test, which may span several days, and a vast amount of performance counter data is recorded. This data includes important information such as CPU utilization, disk I/O, queues, and network traffic, which is analyzed to observe the system's behavior underload. The performance data is compared to the documented behavior of the application/system or to the expected behavior, providing invaluable insights for the performance analysts.

3.2.1 Steps Involved in Load Testing

The typical process of load testing consists of four phases, including environment setup, load test execution, load test analysis, and report generation as shown in Figure 7.

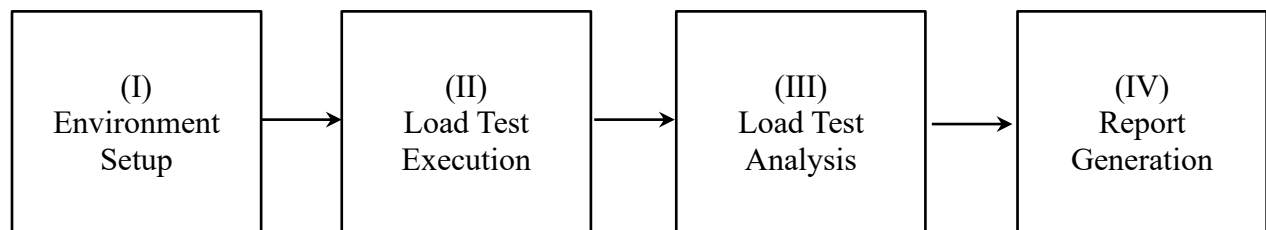


Figure 7: Steps Involved in Load Testing

Environment Setup

Environment setup is a crucial phase in load testing as it is where most failures occur due to improper setup. The environment setup involves installing the application and load testing tools, including configuring the load generators to emulate user interactions with the system, and matching the real workload.

Load Test Execution

The load test execution phase involves starting the system components under load test, including the services, hardware resources, and tools, such as load generators and performance monitors. The application or system under test is monitored closely, and performance counters are recorded in performance logs.

Load Test Analysis

Load test analysis involves comparing the results of a load test to another load test result or pre-defined thresholds. Unlike functional and unit testing, load testing requires additional quantitative metrics, such as response time, throughput, and hardware resource utilization to summarize

results. The performance analyst selects a few important performance counters among the thousands collected, compares them with those of past runs, and looks for evidence of performance deviation using plots and correlation tests.

Report Generation

The report generation phase involves filing any performance deviations found based on the personal judgment of the analyst. The results in the performance report are usually verified by an experienced analyst. Based on the extent of performance deviation and its relevance to the subsystem handling team, such as the database, application, or web system, a performance report is generated.

3.2.2 The Current Practice of Load Test Analysis in LSS

In LSS, it is challenging to find an analyst who has complete and up-to-date knowledge of the underlying systems involved in a load test. To compensate for this, analysts use a variety of methods to assist them in a timely analysis.

Rule of Thumb

Analysts often rely on "rules of thumb," which are important performance counters recommended by domain experts and gurus, to quickly identify performance problems in a load test. However, in large projects, it is rare to have gurus who can provide these "rules of thumb" for every type of load test [15] [16].

History

In the absence of domain experts, performance analysts often use historical information stored in performance test repositories to uncover the test signature, or important performance counters, for a quick load test analysis [17]. However, this method requires well-maintained performance repositories, systematic retrieval techniques, and established traceability between bugs

and test cases.

Self-Judgment

When other options are not available, performance analysts often rely on their personal judgment to analyze load test results. Based on their individual judgment, they determine whether changes compared to the baseline or past runs are significant enough to file defect reports.

2.2.3 Challenges of the Current Practices for Analyzing Load Tests

Large Volume of Data

The results of the load test conducted on LSS encompass a vast amount of performance data obtained from various subsystems that were tested, including numerous applications, databases, web servers, as well as a variety of network components like routers, switches, firewalls, and multiple interface controllers and adapters. This presents several challenges related to organizing and presenting the data in a methodical manner, as well as dealing with time constraints.

Methodological Challenge

Within the domain of load testing, the amount of performance data gathered during a load test can be quite substantial. This data can vary in size from a few hundred megabytes to multiple terabytes, as seen in examples such as Yahoo and Twitter's testing, where they collected 5 petabytes and over 12 terabytes of logs respectively over the course of several days [18]. It is often impractical to repeatedly read such massive amounts of data for each load test in order to evaluate it against a model, such as a baseline load test.

Presentation Challenge

Load tests can range in duration from a few hours to several days and tend to generate a significant amount of performance counter data, often in terabytes. Performance analysts frequently utilize graphs to detect performance deviations or anomalies, particularly when plotting

performance counters to identify fluctuations. For instance, an upward trend in memory usage during a load test may indicate a memory leak. However, an excessive number of performance counters can lead to graphs that are overcrowded and challenging to evaluate. Consequently, analyzing such a substantial amount of performance data represents a significant challenge in the load testing process [19].

Time Pressure Challenge

Typically, load testing occurs at the conclusion of a development cycle and often serves as the final step in a release schedule that has already experienced delays. As a result, load testers typically feel immense pressure to complete testing and certify the software for release. Although load tests can span many hours or days, there is often limited time allocated for analysis. Given the extensive volume of performance counter data and the limited amount of time that testers and analysts have to report their findings from a load test, even experienced analysts can find conducting a comprehensive analysis to detect performance deviations challenging.

Information Extraction Challenge

When analyzing a load test, practitioners often seek out similar historical load tests that were conducted with comparable workloads to the current test. These past load tests can provide practitioners with several benefits, including the ability to quickly identify performance counters used in previous tests, point them towards potential performance issues (such as those experienced in the past), assist in understanding the root cause of any issues, and occasionally even offer a solution that saves time and effort otherwise required to analyze a load test. Nevertheless, there are several challenges associated with extracting the necessary information from past test results due to:

Unstructured Performance Repositories

Regrettably, performance repositories are primarily designed to archive performance tests for record-keeping purposes, rather than for conducting analysis. Therefore, the retrieval of pertinent information regarding similar load test operations and signature profiles is not extensively explored. The process of locating the most relevant test case is manual and laborious due to the lack of systematic techniques for retrieving and utilizing the information stored in performance repositories to analyze the outcomes of future tests [20].

Consistency Challenge

The terminology used by analysts/testers to describe a performance problem in a load test performance report can vary. For instance, heavy memory usage may be reported as 'memory bug,' 'memory leak,' 'memory overload,' 'out of memory,' or 'low mem.' Analysts usually perform a string search to find past tests with similar performance problems to those observed in the current load test. For example, they may use keywords related to performance problem symptoms, such as "memory leak," to search for related test reports from the past. However, inconsistent documentation of performance problems can result in many false positives [21], which can waste the time and effort of analysts.

Inadequate Test Report Information

Analysts in LSS may vary in the level of granularity they use to report load test findings. Due to their busy schedules, they often report only the minimum required information in their reports, which is enough for the report to be deemed complete and placed in a performance repository but may not be sufficient for conducting root cause analysis. Figure 8 provides an example of a performance report obtained from a large LSS's performance repository. The original de-

scriptors, machine names, and counter names have been redacted due to a Non-Disclosure Agreement (NDA). All the mandatory fields of the report (shown in bold) are completed by the analyst, making the report appear complete. However, the information provided may not be sufficient for conducting a root cause analysis.

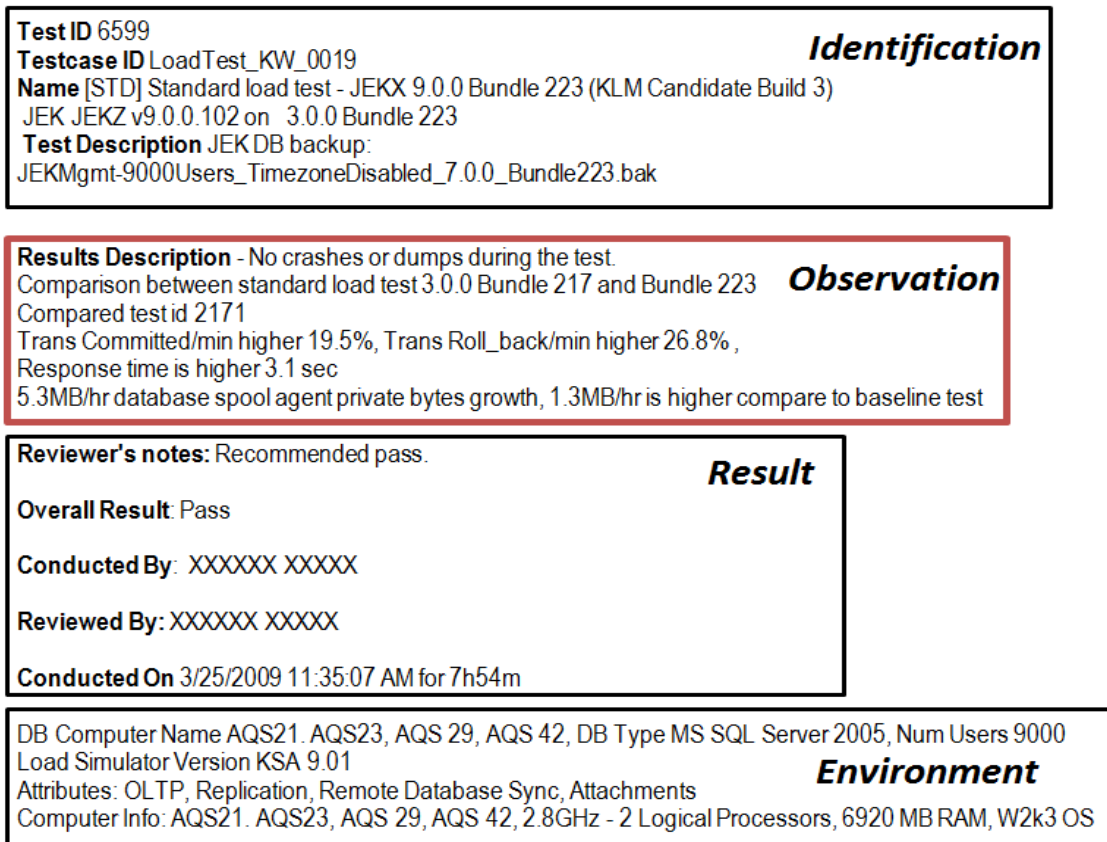


Figure 8: A Sample Test Report

In the result description section of the performance report in Figure 8, the analyst reports that certain performance counters, such as 'Trans committed/min', 'Trans Roll_Back/min', 'Response time', and 'Database spool agent private bytes growth', were significantly higher compared to the baseline test, which is a similar test conducted on the previous bundle of the enterprise software. However, the analyst does not provide any explanation for why the load test passed despite these deviations. It is unclear whether:

These differences were expected or normal for the new version under test.

- The SLAs were maintained or satisfied even though the reported performance counters fluctuated.
- These were the only four performance counters collected and analyzed for this load test, or whether there are other counters that showed deviations.

Performance Fix Traceability Challenge

The load test analysis activity involves not only detecting performance problems during the course of a load test but also identifying the root cause of the problem. If the identified problem is similar to a previously seen problem, a known solution can be applied, which can prevent time-consuming escalations. However, performance test reports of previously solved problems rarely reflect the fix because there is a traceability gap in LSS between the performance problem identified in a performance report and the documented fix in the bug or defect repositories. Therefore, past tests with a similar problem may not necessarily expedite the resolution of a current problem.

Judgement Bias

The current method of relying on personal judgment in load test analysis is prone to errors since it is heavily dependent on an analyst's knowledge and past experience. For instance, one analyst may consider a 5% rise in the number of database transactions per second as a cause for concern, while another may dismiss the increase, citing experimental measurement error.

Chapter 4: Explaining Performance Monitoring Classifiers With SHAP

For monitoring the performance of large-scale systems (LSS), many organizations have begun to integrate AI into their software application performance monitoring environment. To demonstrate an application of XAI, an experiment which utilizes SHAP to explain AI which monitors an LSS was completed. In this experiment, a simulation of an e-commerce software system was constructed, and AI was used to monitor the software system during load tests to signal if any faults—behavior interfering and/or differing with the normal operation of the system—were present. Since the underlying machine learning algorithms used to develop the AI for this task were black-box algorithms, the objectives of this experiment are to: (1) use SHAP to explain the black-box models' decision making, (2) determine how the explanations can be used to help improve the models, (3) demonstrate how the explanations can increase user trust in the system.

4.1 Subject of Study and Environment Setup

The section describes the studied system to perform the load test experiments as well as the environment of the system under study.

4.1.1 The Open-Source System

The Dell DVD Store (DS2) application [22] is the system being examined in this study. It's an open-source simulation of an e-commerce website that's used for benchmarking Dell hardware. DS2 includes standard e-commerce features like user registration, login, product search, and purchase. The application is made up of a web application component, a back-end database component, and driver programs (load generator). DS2 has different versions to support various programming languages such as PHP, JSP, or ASP, and databases like MySQL, Microsoft SQL server, and

Oracle. For this study, the PHP distribution and a MySQL database were used. A mix of transactions, including user registration, product search, and purchases were used as the load.

4.1.2 Testbed

To replicate the load testing process of large-scale software systems, a system of three machines was used as a testbed for the open-source system, i.e., Dell DVD Store. One machine was used as a web server running Apache web service to host the web application for browsing and purchasing DVDs. The second machine was the database server; it hosted a MySQL database to hold the store inventory and customer account information. The third machine acted as a load generator, i.e., generating concurrent load on the servers mimicking real-world users. The machines were all connected to one another physically in a LAN. The hardware specifications of the machines are shown in Table 2.

Machine	CPU Cores	RAM	Disk Type	Disk Size
Web Server	2	4 GB	HDD	300 GB
Database Server	2	4 GB	HDD	300 GB
Load Generator	4	8 GB	HDD	500 GB

Table 2: Hardware Specifications of the Ecommerce Service System

Set-up Process of Testbed

To ensure a sterile testing environment without interference from other software applications, the disks on both the database and web servers were reformatted to NTFS format before

any software installation was completed. Windows 7 Home Premium along with the .NET Framework 4.5.2 and Visual C++ Redistributable 2015-2022 were installed on both the servers.

For the web server, XAMPP was installed to serve the PHP 5 e-commerce website. The server was configured to allow many concurrent user connections, and the site was served using Apache Web Service. For the database server, MySQL Community Edition 5.7 was installed and configured to allow many concurrent connections. The installed database stored all data regarding the DVD store inventory, customer account information, and purchase history. The initial database held 20,000 customers, 1,000 orders per month, and 10,000 DVDs in the store inventory.

Parameter	Value
Testing duration	1 hour
Number of driver (load generator) threads	15
Startup request rate (load ramp-up rate)	25
Think time (time it takes a customer to complete an order)	30 sec
Database size	100 MB
Percentage of new customers	10%
Average number of searches per order	5
Average number of items returned in each search	5
Average number of items per order	5

Table 3: Baseline Load Test Configuration for Dell DVD Store (DS2)

4.1.3 Description of Load

A load generation script from the DS2 package was configured and executed from the load generator machine to simulate many users accessing the e-commerce web service to browse inventory, purchase DVDs, and access their accounts. Table 3 lists the configuration of the DS2 load generator for the baseline load test in the experiments.

4.1.4 Fault Injection

To test our methodologies in practical scenarios, we need to assess them in the presence of typical faults. This involves selecting a fault category, such as software failures, hardware failures, or human/operator errors, and determining the corresponding failure triggers, such as resource depletion, logical errors, or system overload for each category. In a study conducted by Pretet et al. [23] on an enterprise web service system, it was found that 80% of the failures were due to software and human errors. Therefore, in our experiments, we use these categories of faults. Pretet et al. also identified the seven most common triggers for software and human errors. We selected four failure triggers from their list that were suitable for our load testing experiments and are listed in Table 4. In the following paragraphs, we will explain why we chose these specific failure triggers and how they relate to load testing.

For each release or build of an application, performance analysts typically need to conduct a multitude of tests with specific workloads under particular hardware and software conditions [24]. It is crucial for them to meticulously analyze each test to ensure that the system is not being overwhelmed and is meeting its intended service level agreement (SLA), such as response time or latency.

- I. **System Overload:** Load test failures can often be attributed to a misconfiguration of either the system under test or its execution environment. In some cases, this may result from time pressure or the complexity of the configuration itself. For instance, a database or web server could be misconfigured due to a rushed setup or incomplete understanding of the required settings. Similarly, load generators may not be correctly configured, resulting in inaccurate or unreliable test results. It's crucial to ensure that each component is set up and configured correctly to obtain accurate and reliable results from load testing.
- II. **Configuration Errors:** According to Pretet et al., the second most common reason for failures in the operator error category is procedural errors. These types of errors occur when analysts or testers do not follow the established guidelines and processes for conducting a load test. In load testing, procedural errors can take various forms. For instance, a tester may forget to restart a web service or initialize database tables, leading to inaccurate or unreliable test results.
- III. **Procedural errors:** Procedural errors are also prevalent in large-scale systems where multiple processes may be running in the background [23]. In such cases, a tester may overlook or forget to adjust the schedule of interfering loads. For example, starting an antivirus or database replication during the course of a load test can cause procedural errors and lead to invalid test results. To minimize procedural errors in load testing, it's crucial to establish clear guidelines and procedures for each step of the testing process. These guidelines should cover not only the technical aspects of the testing but also the documentation and communication requirements. Testers should also receive adequate training and resources to ensure they understand the guidelines and can execute the tests correctly.

Additionally, it's important to have a standardized process for capturing and reporting procedural errors when they occur. This can help identify trends and patterns in the types of errors that are happening, and allow for targeted training or process improvement efforts to address the underlying causes. By addressing procedural errors and minimizing their occurrence, load testing can be conducted more efficiently and effectively, providing accurate and reliable performance insights for the system under test.

4.1.5 Experiment Design

We designed a total of 5 different performance tests experiments. Each test lasted for a duration of 1 hour and was repeated 3 times to ensure consistency among our findings and to gain statistical confidence on the results. The ramp-up and ramp-down (warm-up and cool-down) periods were excluded from the load test analysis, as the system usually is not stable at these periods. We used perfmon [25], a native windows operating system tool to collect the performance data periodically every 3seconds (sampling interval). This means that all the experiments conducted on the DS2 benchmark application are one hour long and contain 1,200 observations per performance counter. In total 90,000 observations of counters were collected across all experiments (including repetitions).

For both the webserver and the database server, the following performance data was collected:

- Available Megabytes of Memory
- Network Packets Received per Second
- Network Packets Sent per Second
- Logical Disk Idle Time
- Percent Processor Time

In Table 4, we outline the types of faults which were injected during the different experiments.

Category	Failure Trigger	Faults	Exp. Id
Software Failures	Resource Exhaustion	CPU Stress	3
		Memory Stress	4
	Procedural Errors	Unscheduled Replication	5
		Interfering Workload	6

Table 4: Faults Injected into the Load Test Experiments

For the injection of these faults, two different software applications were used: HeavyLoad and Packet Sender. HeavyLoad [26] is a system utilities benchmarking application created by Jam Software. It provides several utilities for resource utilization. In these experiments, HeavyLoad was used to purposefully consume different resources such as CPU, memory, and disk.

Packet Sender [27] is an open-source application for creating and handling network requests, and it was created by NagleCode LLC. In these experiments, the intense traffic generator feature of Packet Sender was used. This feature creates packets at a defined rate to flood the network port of a specified machine.

Description of Each Experiment

- I. **Experiment #1:** The experiment's purpose was to simulate the system's performance before any load was applied. This was done to capture a baseline state for the system. Typically, during this pre-load state, the system performance counters show low utilization of resources such as CPU, memory, and network.

II. **Experiment #2:** The second experiment’s purpose was to measure the performance of the software system under a normal operating load. Many user’s may utilizing the software system to create accounts, browse inventory, purchase DVDs, etc. To simulate these actions, the load generation script from DS2 was used to push the load defined by Table 3.

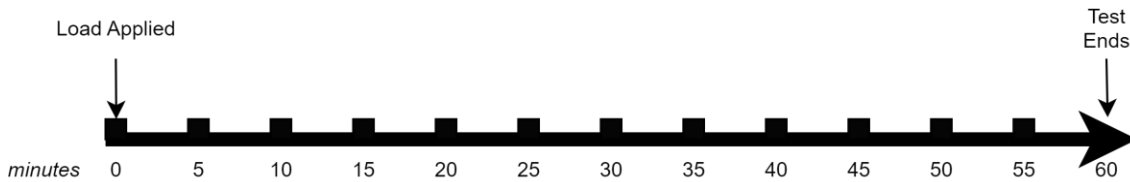


Figure 9: Timeline of Experiment #2

III. **Experiment #3:** Servers may experience resource exhaustion when too many processes are running or if a process requires many resources for execution. In a production environment, there may be additional processes being executed by other departments and users. This can greatly affect the software system’s performance. Users sending requests to the DVD store website may experience delays, or in extreme cases, they may experience a failure of the system to process the request. Simulating scenarios such as this is the purpose of experiment 3. In this experiment, the load generation script from DS2 was used to push to load defined by Table 3 for 1 hour; however, the HeavyLoad application was run on the web server over two 15-minute intervals to utilize the web server’s CPU. During the first 15-minute interval, HeavyLoad was executed using one core of the CPU; during the second 15-minute interval, HeavyLoad was executed using both cores of the CPU. The timeline of experiment 3 is shown in Figure 10.

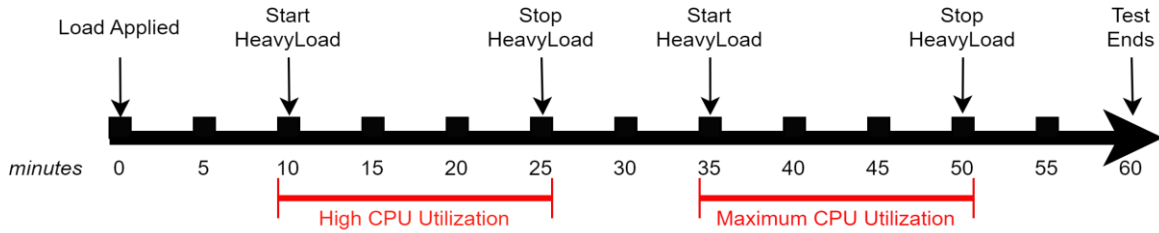


Figure 10: Timeline of Experiment #3

IV. Experiment #4: As discussed in experiment 3, resource exhaustion can greatly affect the performance of a software system and inhibit its ability to function properly. The purpose of experiment 4 is to simulate scenarios where many concurrent applications interfere with the system performance by over-utilizing the available memory, leaving only a limited amount for normal operation. The load generation script from DS2 was used to push the load defined by Table 3 for 1 hour. During this time, the HeavyLoad application was executed on the web server over two 15-minute intervals to consume any remaining memory on the web server. Within each 15-minute interval, it took 7.3 minutes on average for HeavyLoad to consume all unused megabytes of memory. At the end of each 15-minute interval, HeavyLoad was stopped, and the memory it was consuming was freed immediately. The timeline of experiment 4 is shown in Figure 11.

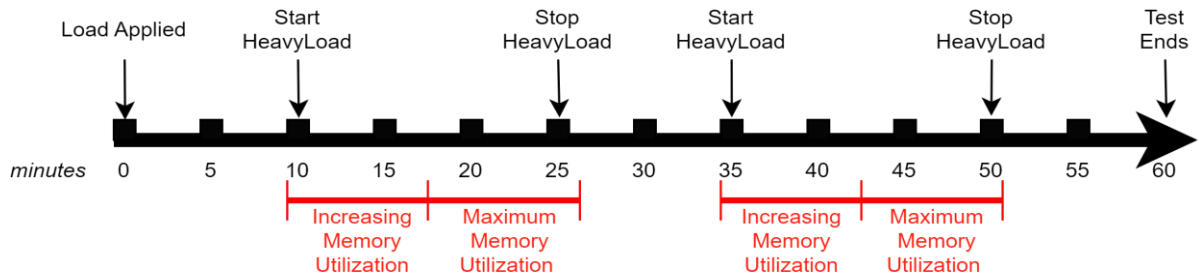


Figure 11: Timeline of Experiment #4

V. **Experiment #5:** Resource exhaustion is not the only factor which can affect the performance of a software system. Sometimes other users and applications can affect the performance even when not directly utilizing the computing resources of CPU or RAM. One way this occurs is through disk utilization. If many users are accessing data stored on the same disk, each may need to wait for the previous transaction to complete. Experiment 5 simulates this scenario. The load generation script of DS2 was used to push the load defined by Table 3 for 1 hour. During this time, the HeavyLoad application was used to continuously write a temporary file to the disk on the database server over two 15-minute intervals. Within each 15-minute interval, transactions from the DVD store web site were delayed, causing slower responses to users even when resource exhaustion of the web server was not present. A timeline of experiment 5 is shown in Figure 12.

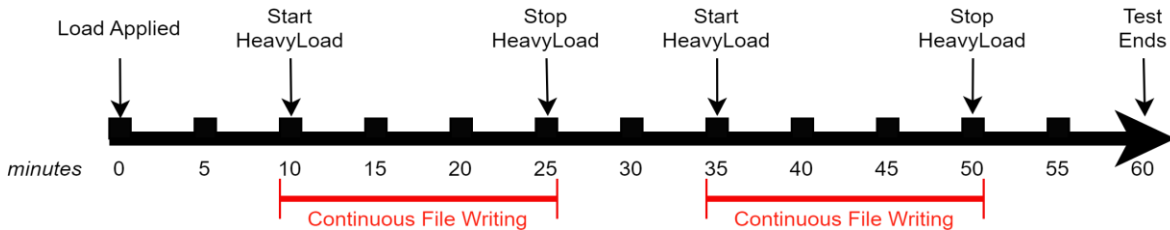


Figure 12: Timeline of Experiment #5

VI. **Experiment #6:** Another scenario that can affect the performance of the software system is an interfering workload. These workloads can utilize the resources of the software system to interfere with its performance as simulated in the other experiments. However, sometimes the interfering workloads may not over-utilize system resources. This does not mean that performance is unaffected though. These workloads may still slow the system down by adding additional congestion to the network due to a high amount of requests.

Experiment 6 simulates this scenario. The load generation script from DS2 was used to push the load defined by Table 3 for 1 hour. During the two 15-minute intervals, the Packet Sender application was used to continuously generate and send requests to the software system. In the first 15-minute interval, requests are sent to port 80 of the web server. In the second 15-minute interval, requests are sent to port 3306 of the database server. A timeline of experiment 6 is shown in Figure 13.

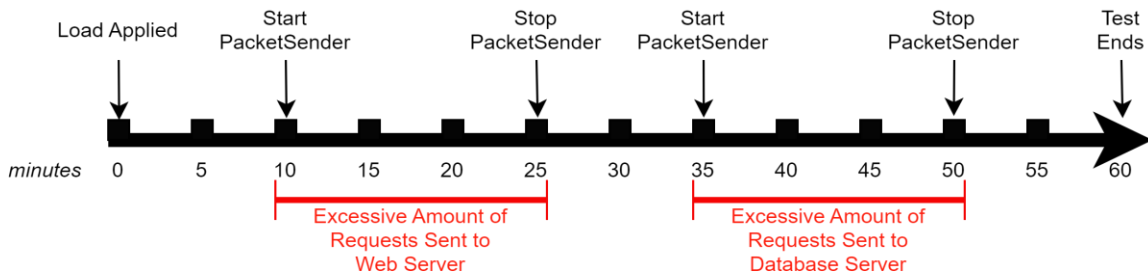


Figure 13: Timeline of Experiment #6

4.1.6 Organizing Load Test Results

After each load test was executed, the resulting log files from the test were saved to an external disk. For each test, there are a total of 2 log files. For the database and web server, there is one log generated by perfmon to record the performance of each server during the tests. The perfmon log contains timestamped data which was logged every 3 seconds. The second log file was the result of printing the command line output from the load generation script; this command line output logged the average amount of latency that the user’s would experience over the last 10-second interval.

It is not always possible to have the clock of all machines synced together. Many times,

even synced, over some time a natural drift occurs resulting in some difference or deviations between the 2 timestamps of a same event on different machines. To eliminate and difference or deviations between the time stamps between the 2 logs per each test, a Python script is produced to conjoin all files into one comma separated file. This script read the perfmon logs, synced both such that the timestamps matched, and then added the latency from the load generation log file to each row.

The recorded latency was applied to each perfmon recording which occurred within each 10-second time interval, so each 3-4 perfmon instances, falling within the 1—second interval, would receive the same value.

To better understand the behavior of the software system, we added an additional column called "Fault Presence" to the dataset. This column contains binary values, with '0' indicating normal system behavior and '1' indicating the presence of a fault at the time of logging. Since we knew the specific times when faults were being injected during the experiments, we manually inserted the values of 0 or 1 into the column based on the corresponding timestamp.

Once the log files were merged, and the additional features were added, we saved 11 columns from the conjoined log into a comma-separated file for each test. These 11 features are summarized in Table 5, providing a comprehensive overview of the recorded instances.

Feature Name	Domain	Description
WS Available Megabytes	(0, 3370)	Amount of memory (in Megabytes) in the web-server not being utilized by any application
WS Packets Received per Second	(0, 160)	Amount of network packets received each second by the network adapter in the web server
WS Packets Sent per Second	(0, 160)	Amount of network packets sent each second by the network adapter in the web server
WS Percent Processor Time Total	(0, 100)	The percentage of elapsed time that the CPU of the web server spends executing threads
WS Logical Disk Percent Idle Time	(0, 100)	The percentage of elapsed time that the disk of the web server is not executing read or write commands
DB Available Megabytes	(0, 2850)	Amount of memory (in Megabytes) in the database server not being utilized by any application
DB Packets Received per Second	(0, 215)	Amount of network packets received each second by the network adapter in the database server
DB Packets Sent per Second	(0, 215)	Amount of network packets sent each second by the network adapter in the database server
DB Percent Processor Time Total	(0, 25)	The percentage of elapsed time that the CPU of the database server spends executing threads
DB Logical Disk Percent Idle Time	(0, 100)	The percentage of elapsed time that the disk of the database server is not executing read or write commands
Latency	(0, 3060)	The average amount of time (in milliseconds) users must wait for the software system to respond to a request
Fault Presence	(0, 1)	A binary number representing whether a fault was being injected into the software system at the point in time that this instance was logged; 1 represents a fault being present; 0 indicates no fault

Table 5: Features of the Experiment Data Sets

4.2 Classifier Models

After the completion of all load tests, two classifier models were developed to monitor the system and identify the presence of faults—specifically the 4 types of faults, typical to software systems, listed in Table 4.

Since the tests were conducted in a controlled environment and the times when faults were present in the system were known, we decided to use supervised learning algorithms to analyze the data. The machine learning algorithms chosen for this purpose were artificial neural networks (ANN) and random forest. Both algorithms are considered black-box machine learning models and have gained significant popularity in recent years. The ANN and random forest models are powerful tools for identifying patterns in complex datasets and have been widely adopted in various fields. With their ability to handle high-dimensional datasets and perform nonlinear transformations, they were deemed suitable for this study.

For the ANN model, it was decided to use TensorFlow for much of the implementation; TensorFlow is a popular open-source library that implements machine learning algorithms [28]. It is particularly well-known for its implementation of deep learning models. The Sequential model from TensorFlow Keras API was used to compile a simple classification ANN with a 3-layer architecture:

1. *Input Layer*: brings the input data into the ANN system for further processing by subsequent layers
2. *Hidden Layer*: utilizes an activation function and learned weights to produce outputs
3. *Output Layer*: receives outputs from the hidden layer and chooses which classification to output as the final prediction

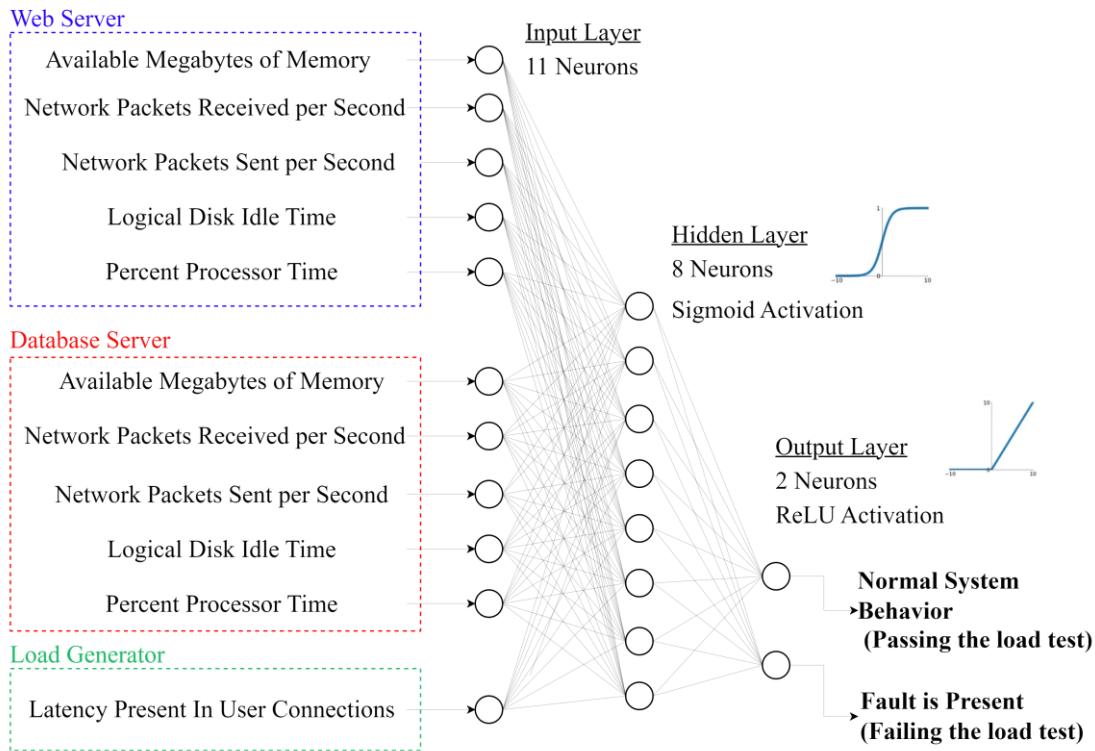


Figure 14: Architecture of the ANN classifier

This architecture is shown in Figure 14. This implementation utilized an S-shaped curve for the activation function of the hidden layer. This allows for the ANN to learn non-linear relationships between the input features. For the output layer, a linear activation function was chosen. This means that the output layer chooses between the two possible classifications—normal system behavior or the presence of a fault—depending on which receives the highest output from the hidden layer.

For the other classifier model, a random forest algorithm was chosen. Since decision tree algorithms are not well supported by TensorFlow for a Windows 7 operating system, another library—Scikit Learn—was chosen; Scikit Learn is another popular library implemented in Python to provide machine learning and data analysis tools [29]. This model utilized an ensemble of 100 decision trees. To give a visual representation of the random forest classifier, 2 random trees from

the ensemble were selected and displayed in Figure 15.

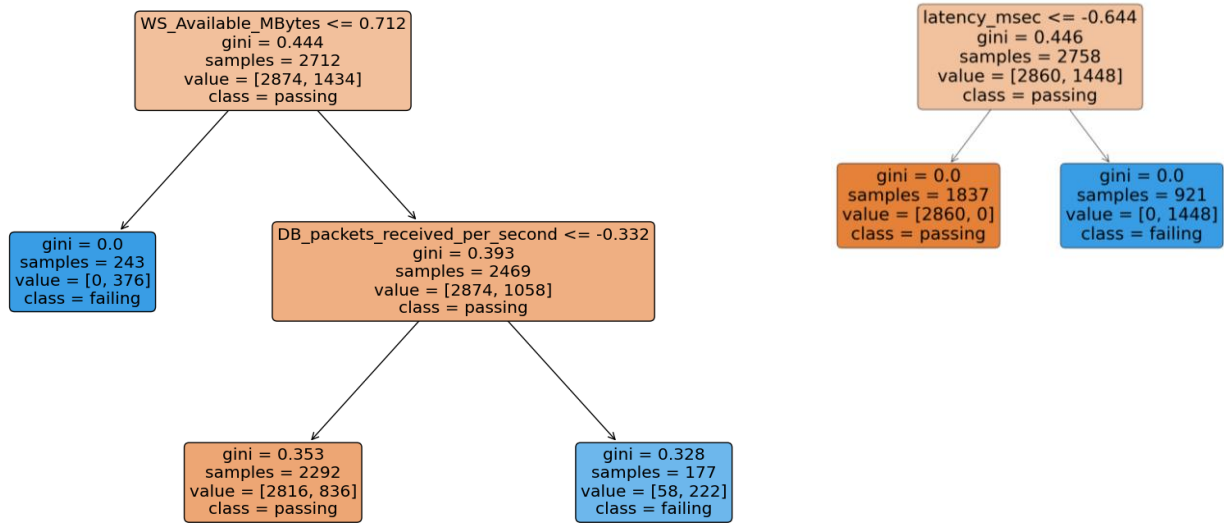


Figure 15: Two Decision Trees from the Random Forest Classifier

4.2.1 Classifier Model Training and Testing

Once the machine learning models were developed, it was time to see if they could identify faults in the software system’s performance. To accomplish this, both models must first learn how to identify faults, so they needed to be trained on the data gathered from all of the load test experiments. This training process occurred in several steps.

First, the feature values of the data were all scaled into the range [0, 1] using the approximate measured ranges for each feature shown in Table 5. Machine learning models, such as ANN, can experience difficulty in learning data where features all occur on a different scale, so to ensure that the model treats all features with the same importance, this scaling is necessary. The scaling method used was min-max scaling, shown in Equation 3.

$$\frac{s(j) - \min(j)}{\max(j) - \min(j)}$$

Equation 3: Feature Scaling Into Range [0, 1]

In Equation 3, j represents one feature of the data set—ex. logical disk idle time, latency, etc. $s(j)$ represents the value of the feature in sample s . $\min(j)$ represents the minimum value that feature j experiences over the entire dataset; $\max(j)$ represents the maximum value that feature j experiences over the entire dataset.

Second, all the data was divided into two different groups: a training group and a testing group. The training group consisted of a random 70% of the instances, and the remaining 30% were used for the testing group. This split was made to ensure that the classifier models do not get to see all of the data when being trained; this way, unseen data will be left over after the training which can be used to validate how well the models have learned from training.

Third, the models were fit on the training data. For the ANN model, the training process utilized Adam, a popular algorithm for optimizing a neural network, to update the ANN weights based on the amount of loss. The loss was defined as the mean squared error (MSE) present between the model prediction and ground truth. For the random forest model, the training process utilized bagging. Bagging is a process that randomly selects a random subset of samples from the input data and creates a decision tree model to be trained on the subset. After training each individual decision tree, the trees are all joined into one ensemble, which is the overall random forest model. This model can then make predictions on data by allowing all trees in the ensemble to vote on their own predicted classification.

After the training of each model was completed, several metrics were calculated to evaluate the results of the training. For the ANN model, these metrics are 'Accuracy' and 'MSE Loss'. Accuracy tells you how many times the ANN model was correct overall. TensorFlow defines accuracy as the mean accuracy of predictions of all instances in a data set. The accuracy function in TensorFlow creates two local variables, total and count that are used to compute the frequency

with which predictions matches labels. This frequency is ultimately returned as accuracy: an idempotent operation that simply divides the total by count.

Mean Squared Error loss (MSE) is calculated as the average of the squared differences between the predicted and actual values. In our training case, its average MSE present over all the predictions in the training set. For the random forest, mean accuracy was used. When predicting the class of all instances in the training set, the ANN model resulted in a mean MSE loss of 0.322 and a mean accuracy of 99%; similarly, the random forest classifier also achieved a high accuracy of 100%.

The models' predictions were then evaluated on the testing set—a set of data instances that the models have not yet seen, i.e., the remaining 30% split. The ANN model resulted in a mean MSE loss of 0.325 and a mean accuracy of 99%. The random forest classifier achieved a mean accuracy of 99%.

4.2.2 SHAP Values for the Classifier Models

Using only the metrics of accuracy and loss recorded from the results of training and testing, the classifier models appear to have learned the data well. Both models exhibited mean prediction accuracies of 99% or greater even when making predictions on data not yet seen. However, high values of accuracy and loss are not sufficient to justify the trustworthiness of a model. When accuracy is very high, there is a possibility of model overfitting. Overfitting is when a machine learning model memorizes specific patterns in data rather than learning to generalize. This results in the network correctly memorizing a data set, but it often fails to make correct predictions on unseen data. However, the testing accuracy of the ANN is also very high, so identifying overfitting here is not an easy problem.

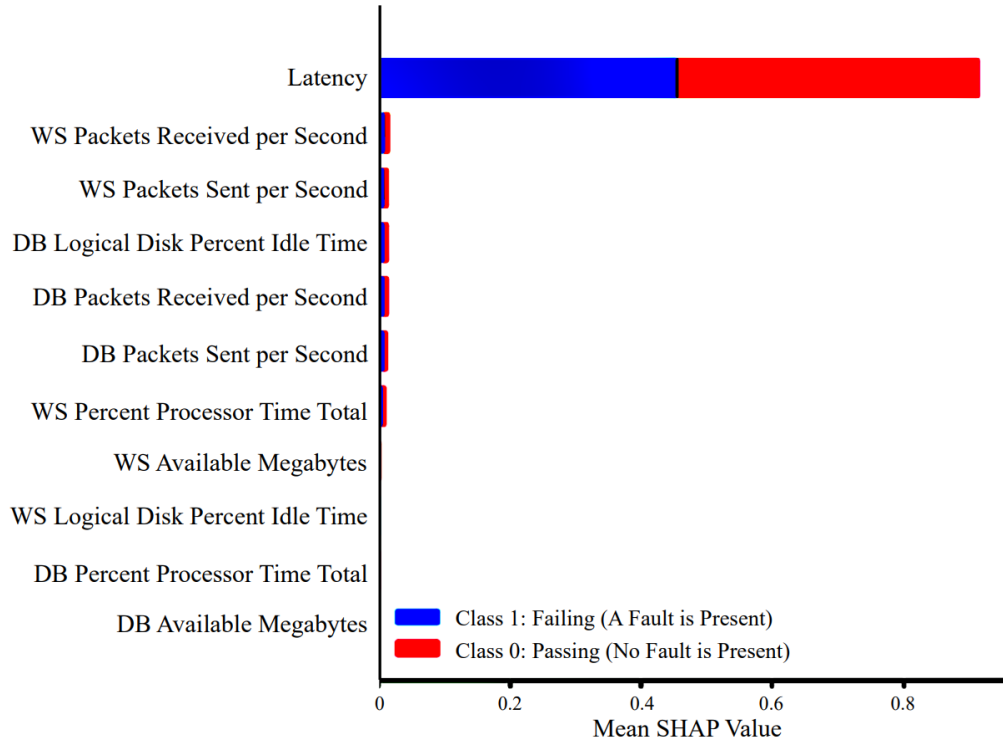


Figure 16: Mean SHAP Values for ANN Model

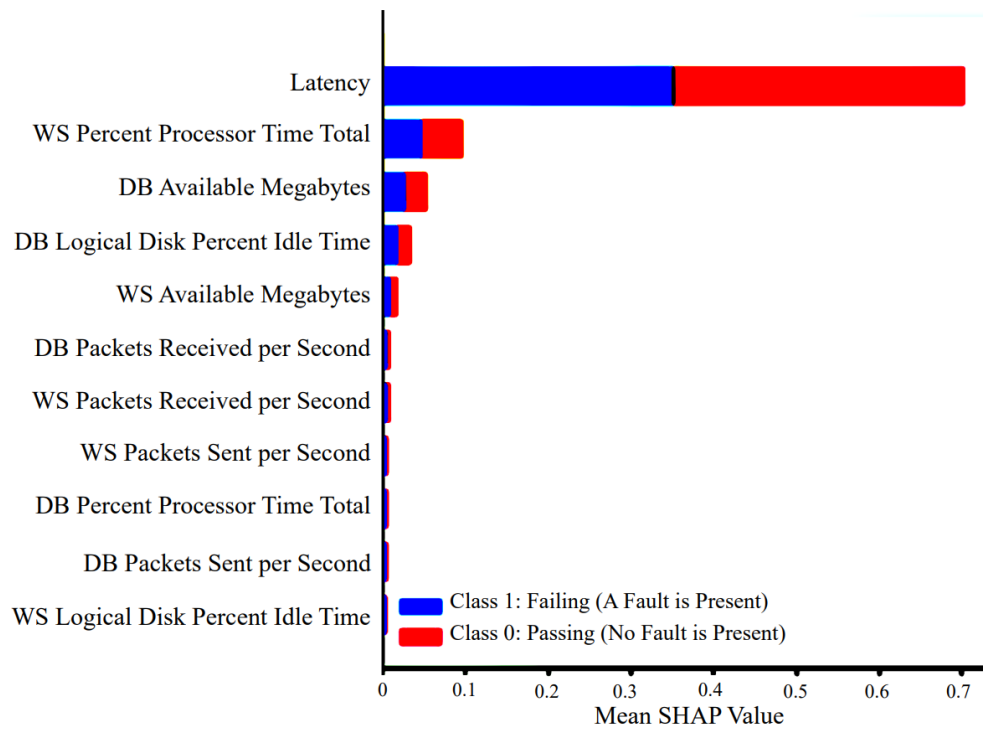


Figure 17: Mean SHAP Values for Random Forest Model

To understand the behavior of this black-box model, XAI can be applied to give insight into the decision-making process of the learners, i.e., ANN and Radom Forest in crafting the prediction mode. SHAP was used to identify the contribution of each feature discussed in Table 5 to the model predictions. Utilizing KernelSHAP—a weighted linear regression to approximate the SHAP values of each feature using the linear equation shown in Equation 1—the average contribution of each feature towards predictions of ‘Pass’ and ‘Fail’ on all instances in the testing set was computed. To visualize these values, the SHAP library for Python offers many plot types, the most common being a summary plot. The summary plot shows the mean SHAP values over an entire dataset towards each prediction class. A summary plot for the ANN model is shown in Figure 16, where ‘Class 1’ represents ‘Fail’ and ‘Class 0’ represents ‘Pass’. The same plot for the random forest model is shown in Figure 17.

From observing Figure 16 and 17, an issue is immediately observed. The contribution of the variables is very unbalanced. Most SHAP values are near 0, which means that the corresponding features have little or no contribution toward the predictions of the classifier model. In contrast, the SHAP value corresponding to the latency is very high. For both ‘Class 0’ (Pass) and ‘Class 1’ (Fail), the SHAP value is greater than 0.35. Since the SHAP value for latency is proportionally much greater than the other values, we can see that the ANN model is making decisions almost entirely based on the amount of latency present.

To examine this, we can plot the SHAP values towards a class. i.e., Pass or Fail, for each sample before they were averaged into the mean SHAP values seen in Figure 16. In Figure 18, this plot is shown using a bee swarm plot—a variation on a scatter plot that allows individual points to be more visible. Here, the SHAP values towards ‘Class 1’ (Fail) are shown. Here, the

coloring from blue to red indicates the magnitude of the associated feature value to the SHAP value being plotted.

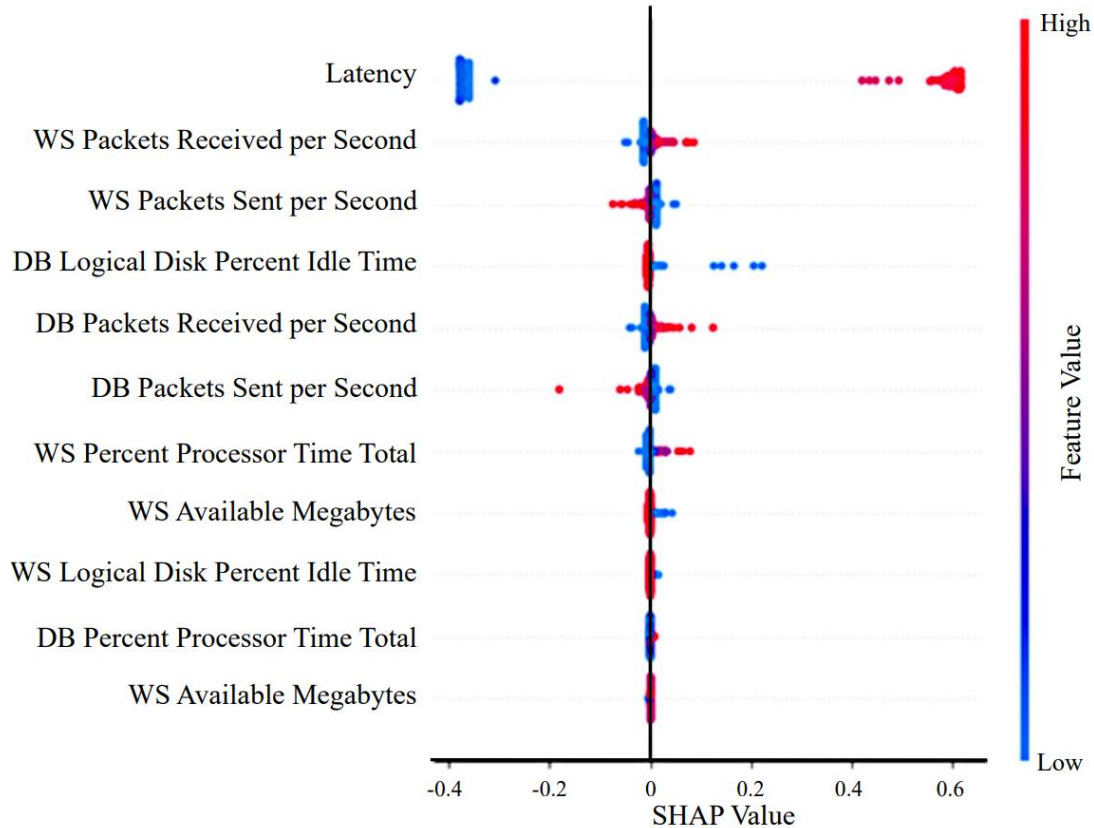


Figure 18: SHAP Values Towards Predictions of Fault Identification

Figure 18 shows again that the latency contributes highly to predictions where the ANN model identifies a fault, i.e., Fail. When the magnitude of the latency is high, the marginal contribution of latency is in the range 0.4 to 0.6. When the magnitude of the latency is low, then the marginal contribution is in the range -0.3 to -0.4.

This behavior can cause an issue with trust in the models. Making predictions based solely on the magnitude of the latency is likely a method that would fail in a production environment. Since this variable is affected by many factors, it could be unreliable in different scenarios. Across all the load tests, the latency ranged from 0 to 3 seconds, and this was using the same

configuration for the load generation script. In a production environment, the load and the resulting latency could be much more varied. If a low number of users are using the system, then the latency may remain low, even if a fault is present.

4.2.3 Updating the Classifier Models

Using the insight obtained from KernelSHAP explanations, it was evident that a method which gives less contribution to the latency feature needed to be developed. To examine possible reasons that the classifier models learned to associate the latency with the class label so strongly, correlation heatmap of the features of all instances in the testing set is shown in Figure 19.

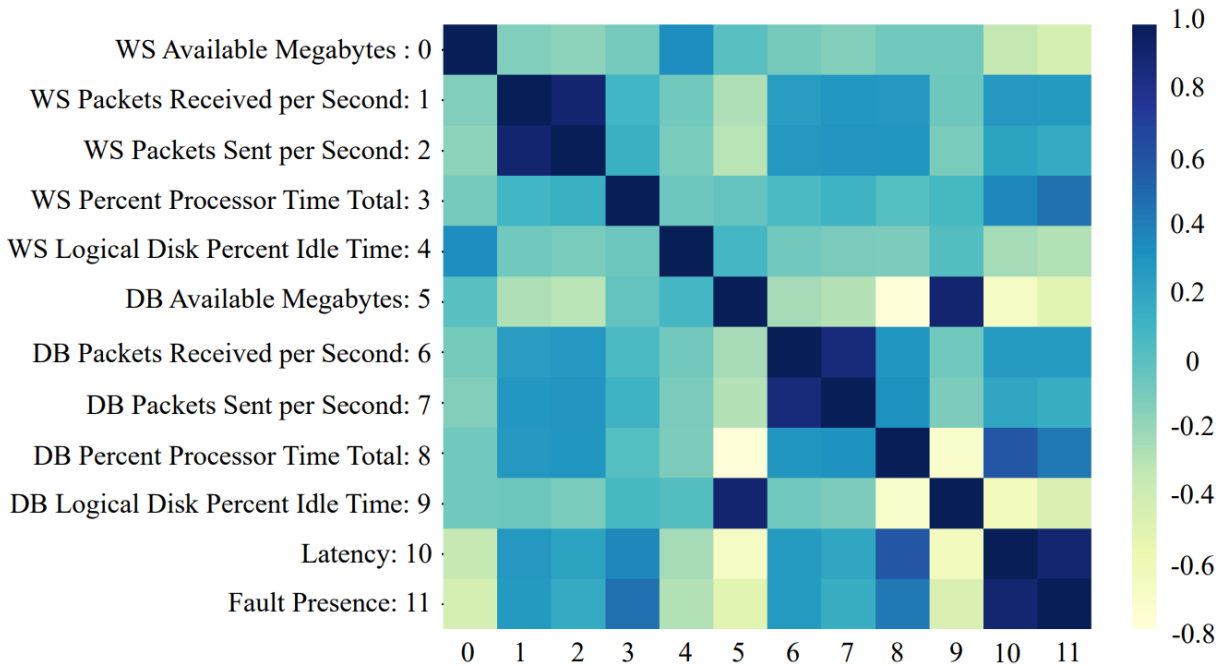


Figure 19: Correlation Heatmap of Testing Set

In Figure 19, we see a correlation of 0.91 between the class label and the latency. Since this correlation is very high, it was decided to retrain the models while the latency variable was

removed. This way, we can see if the models are able to identify patterns in the other features without the high correlation of latency providing a direct path to correct prediction.

The training and testing of the new models followed the same process as was completed for the previous models; the only difference was that the latency feature value was removed for every instance in the data set. After the new training was complete, the mean accuracy and loss over the new training set were recorded. For the ANN model, the training accuracy was 98%, and the MSE loss was $5.35e-7$. On the testing set, the mean testing accuracy was 97%, and the MSE loss was $3.73e-7$. For the random forest model, the mean training accuracy was 99%, and the mean testing accuracy was 96%.

Once again, we can use SHAP to examine the classifiers' behavior to determine if it better learned the importance of other features in the data set. Using KernelSHAP, a summary plot of the SHAP values for the updated ANN model over the testing set is shown in Figure 20.

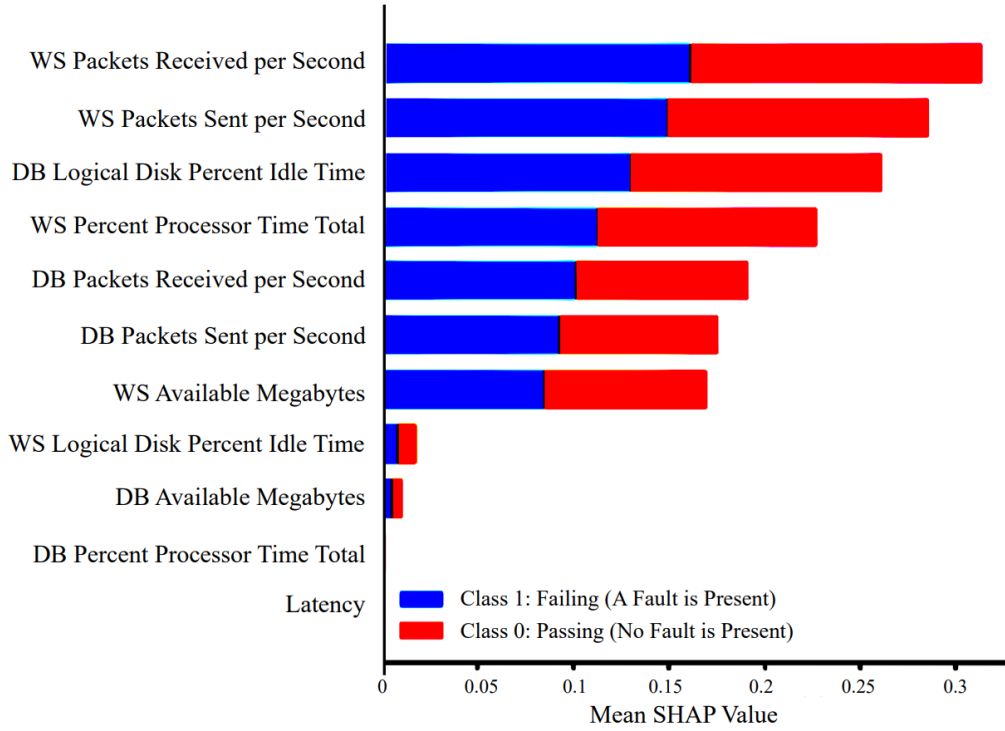


Figure 20: Mean SHAP Values Over Testing Set for Updated ANN Model

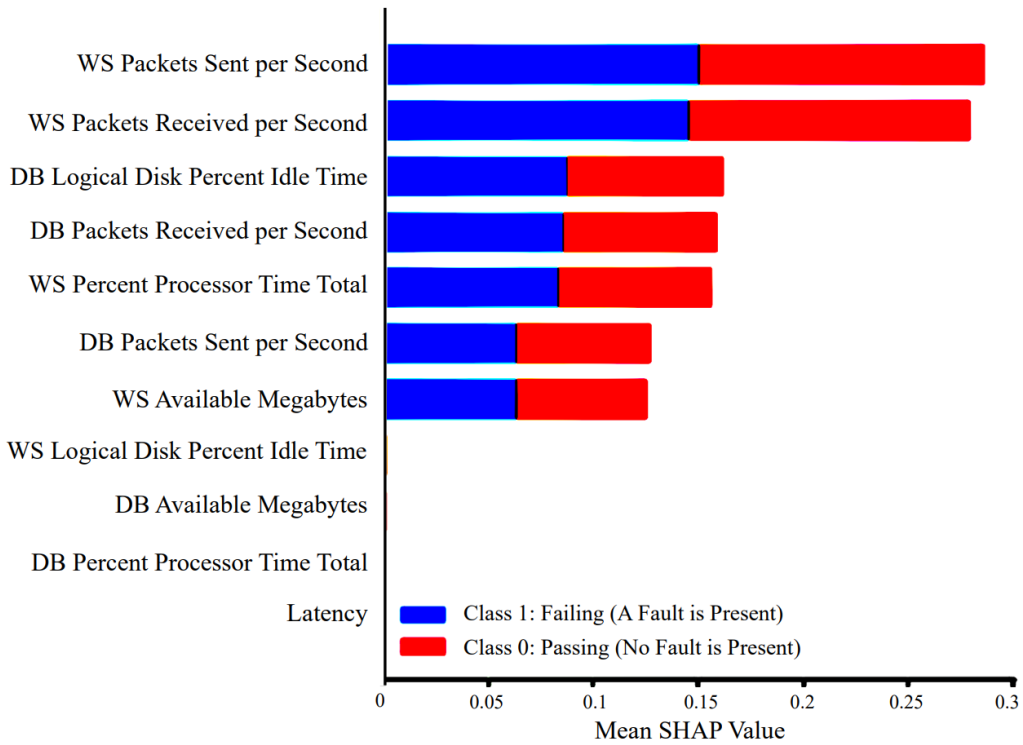


Figure 21: Mean SHAP Values Over Testing Set for Updated Random Forest Model

These SHAP values show improved results over the original models. The network traffic to and from the web server is now considered to be the largest contributor on average; however, this contribution does not significantly outweigh the contribution of other factors. The updated models take several performance metrics into account, such as CPU and memory utilization. These factors should show some impact considering that resource exhaustion of CPU and memory are two of the faults injected during the load test experiments.

To ensure that the importance of each feature is correctly understood by the models, we can show the specific SHAP values for instances recorded during specific experiments. If the model truly learned to generalize the data, then the SHAP values should point toward the root cause of the fault being identified.

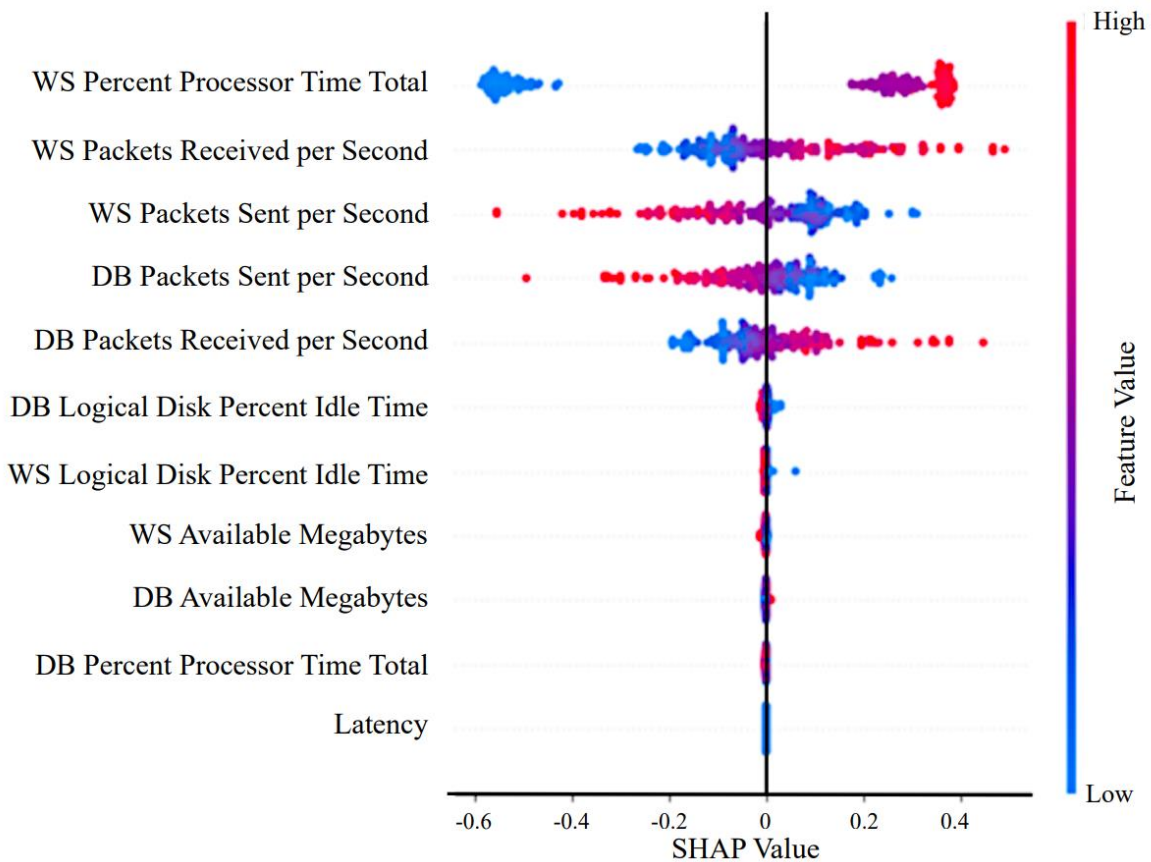


Figure 22: SHAP Values Towards Fault Identification During CPU Exhaustion

In Figure 22, we see the SHAP values towards ‘Class 1’ (Fail) for instances in experiment 3 (resource exhaustion of CPU). The percent processor time of the web server has the highest contribution. When the CPU is being utilized heavily, the marginal contribution ranges from 0.15 to 0.35; when the CPU is being utilized under normal load, the contribution ranges from -0.35 to -0.65. This shows that the model clearly recognized how CPU utilization should contribute to a prediction. Along with this, we can see a high variance in SHAP values related to network traffic (packets received/sent).

We can continue to examine summary plots for instances of each experiment to ensure that the model learned the importance of other features appropriately. Figure 23 shows the SHAP values towards the ‘Class 1’ for instances in experiment 4 (memory exhaustion).

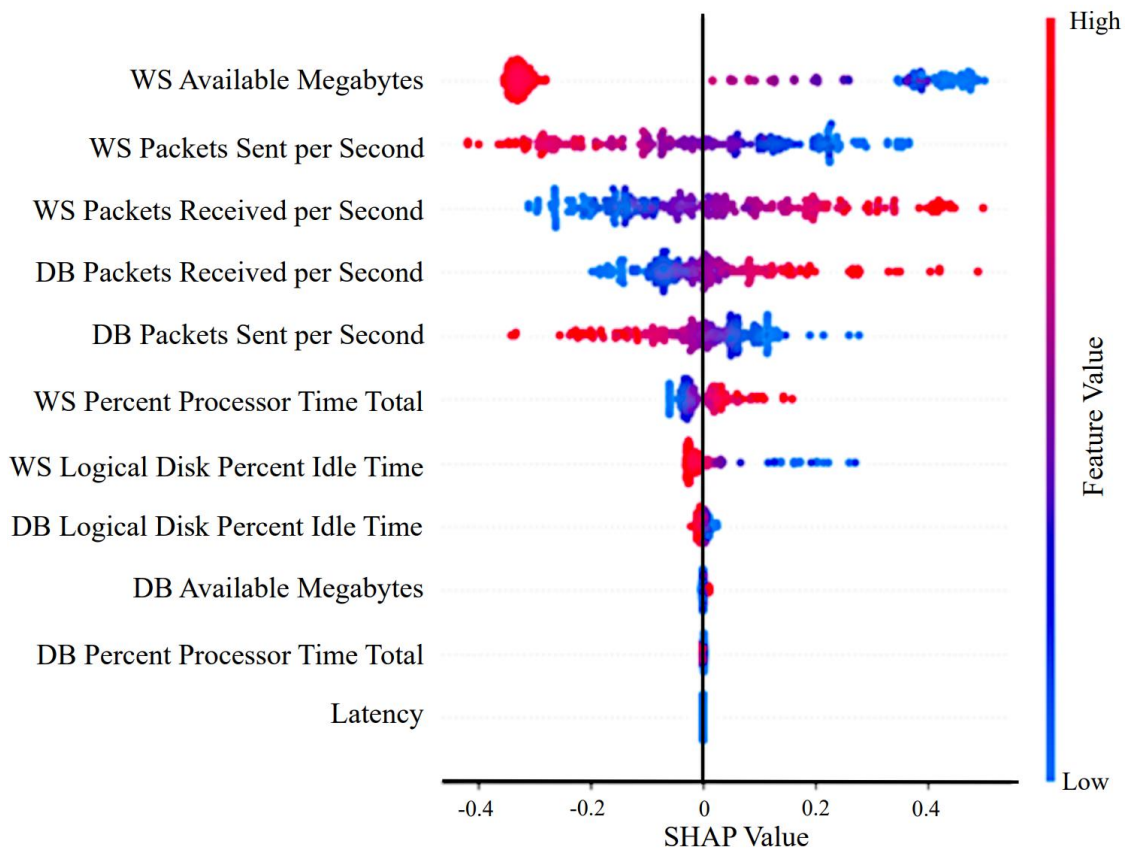


Figure 23: SHAP Values Towards Fault Identification During Memory Exhaustion

The updated ANN model demonstrates a better understanding of the importance of feature values. For instance, when detecting a fault during memory exhaustion, the model's prediction is highly influenced by the low amount of available memory in the webserver. The contribution in these cases ranges from 0.25 to 0.45. Conversely, in cases where a normal amount of memory is available (considered to be a high feature value in the scaling), the contribution ranges from -0.25 to -0.4. Additionally, the ANN considers the magnitude of the network traffic in making predictions. Experiment 4 shows that not all predictions were solely based on the feature value for available memory in the webserver.

Furthermore, SHAP values are instrumental in identifying faults in instances during experiment 5, which involves unscheduled replication.

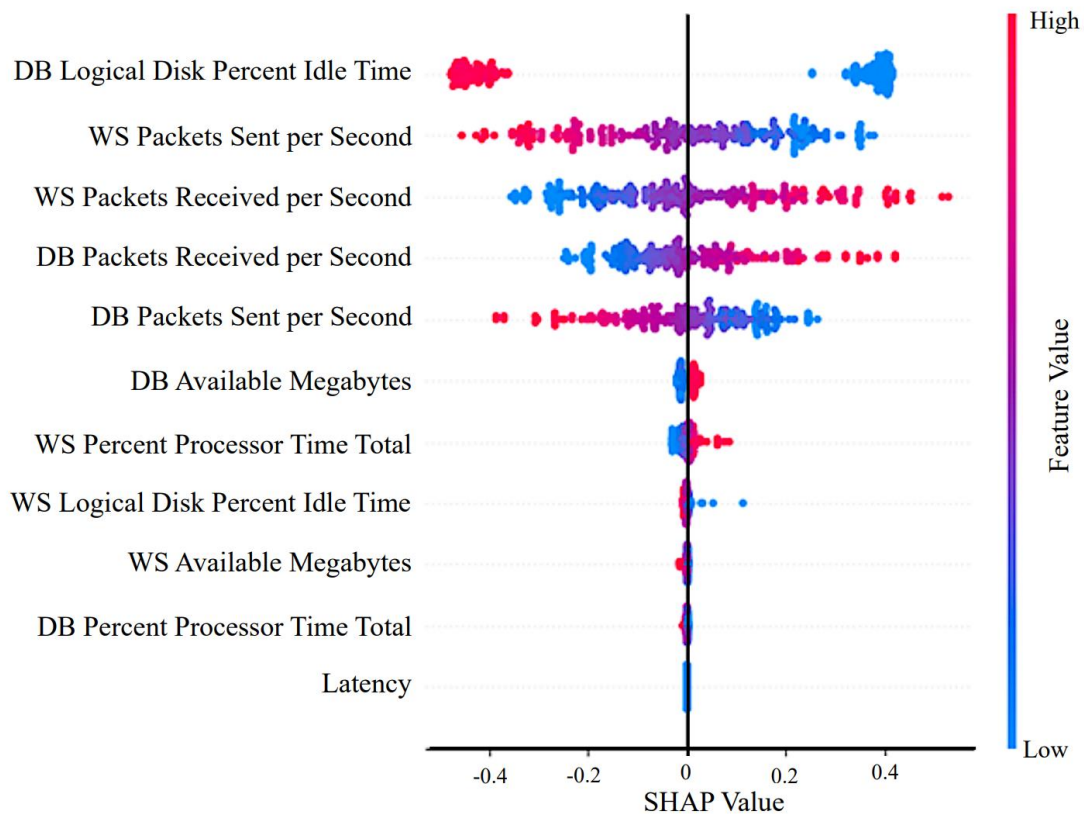


Figure 24: SHAP Values Towards Fault Identification During Unscheduled Replication

In the case of unscheduled replication, the classifier demonstrates a solid understanding of the different features that contribute to fault prediction. Specifically, the percentage of time that the database server's disk is idle is the most significant factor in predicting a fault. When the idle time is low, indicating higher than normal disk utilization, the contribution to the prediction is between 0.25 to 0.4. Conversely, when the disk has a high percentage of idle time, as it would under normal operating conditions, the contribution is between -0.35 to -0.45.

In the final experiment, which involves an interfering workload, SHAP values play a crucial role in identifying faults. Figure 25 illustrates the SHAP values towards fault identification in this experiment.

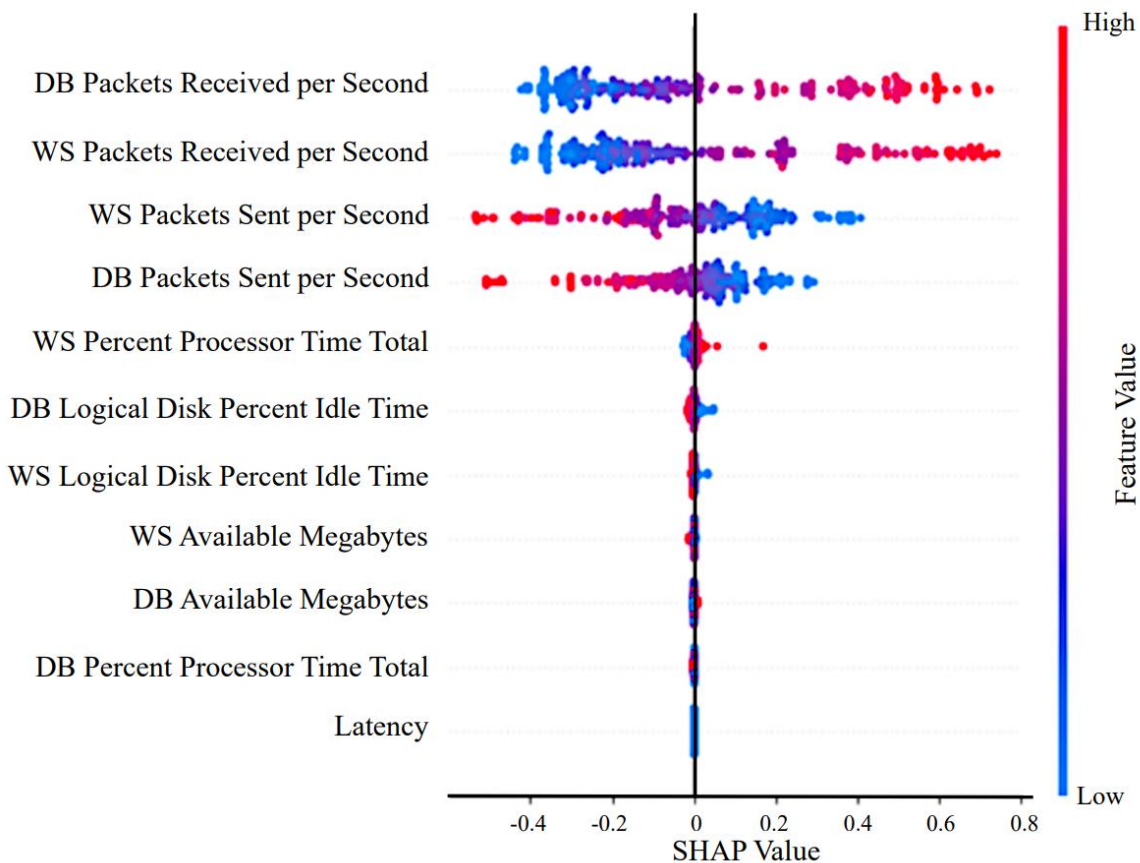


Figure 25: SHAP Values Towards Fault Identification During Interfering Workload

Detecting an interfering workload fault can be challenging, as shown in experiment 6, which greatly increased network traffic and connection latency without significantly affecting system resources. While the SHAP values indicate that the ANN model learned to identify this fault, its decision-making method may not always be reliable. For example, during an interfering workload, more network packets may be sent and received due to multiple job requests and execution. However, the SHAP values demonstrate that the model is more likely to identify an interfering workload fault when the number of packets sent from the system is lower. A high number of packets, in fact, negatively contributes in all cases shown.

Considering that the identification of a fault can include all types of faults induced during the experiments, it appears that the network may have over-generalized patterns related to the number of network packets being sent. For instance, memory exhaustion slows down system performance as the system must wait for memory to become available to complete tasks. Therefore, even if the system receives a high number of packets, the number of packets sent out from the system should be low. Consequently, a lower number of network packets sent out makes sense as a contributing factor to the identification of a fault.

To further examine the contribution of network packets sent and received to fault identification during an interfering workload, we can plot the SHAP values for a random instance from experiment 6 where the model predicted a fault. Figure 26 illustrates this plot.

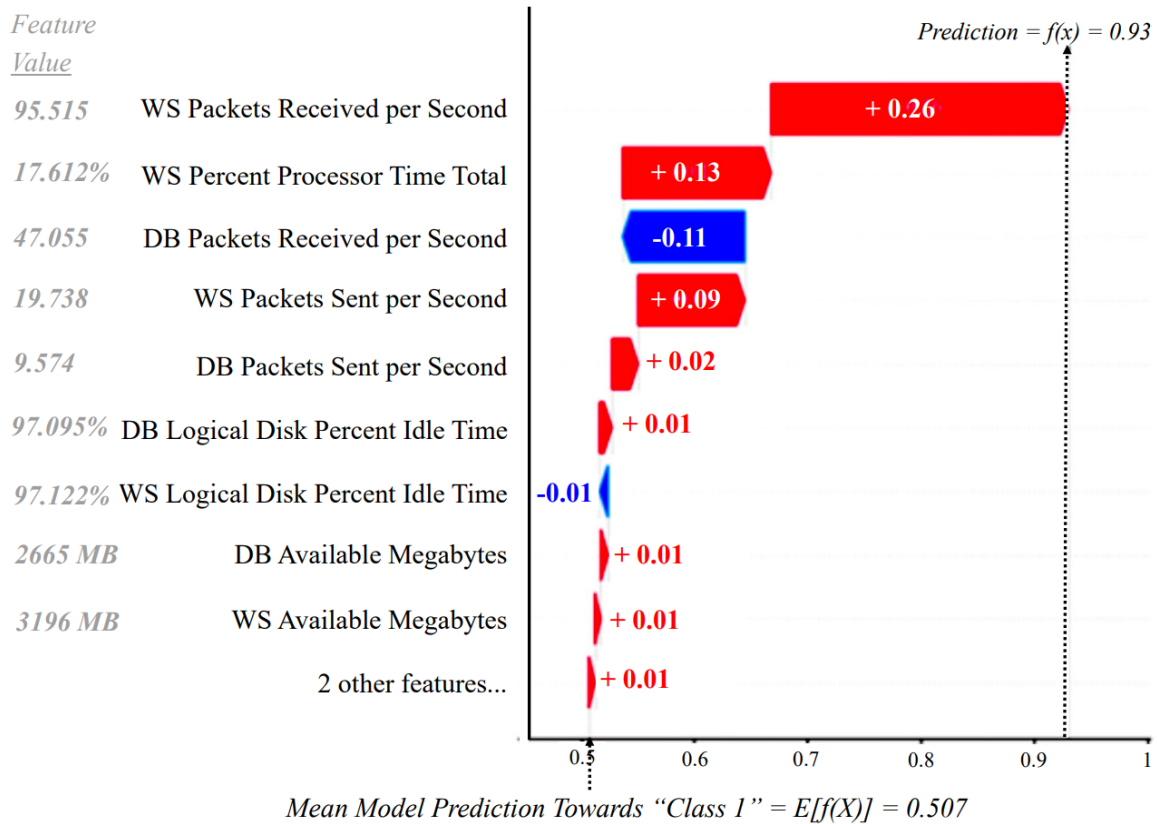


Figure 26: SHAP Values for a Random Instance During Interfering Workload

Here, we use a waterfall plot. This type of plot shows the SHAP values for one sample. For our sample chosen, we see the final prediction towards the class of identifying a fault is 0.93. The model was confident that a fault was present, and the SHAP values show us why. Starting at the average prediction towards this class, 0.507 (represented by $E[f(X)]$), the SHAP values show how the prediction ended up at 0.93. The largest contributing factor was the number of network packets received by the web server. This makes sense considering we are identifying a fault of interfering workload where other systems send requests to the software system being examined. The number of packets received per second by the webserver was 95.515, which is higher than the software system would normally receive, as shown in Figure 27.

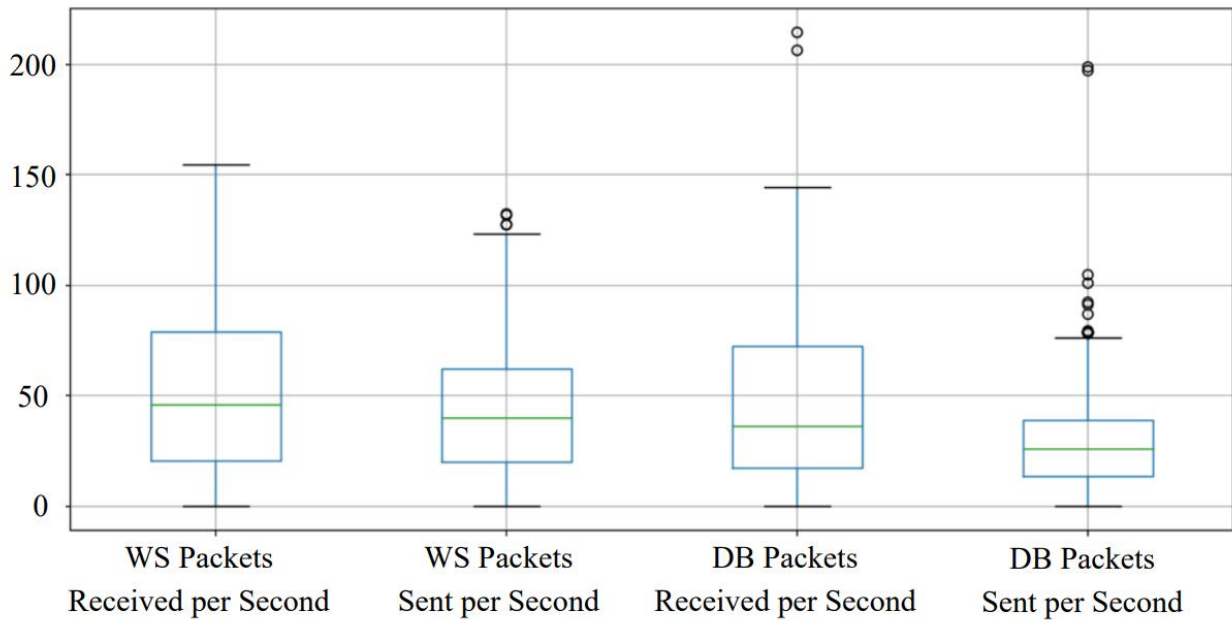


Figure 27: Boxplots of Network Traffic During Experiment 6

The next highest contributing factor in Figure 26 is the percent processor time of the web server. We see that the value was 17.612%, which is higher than the average percent processor time when no fault is injected—around 10 to 14%. We then see the contribution of packets received by the database server as the third-highest contributing factor. However, it negatively contributed in this case. Looking at the feature value, we see that the packets received per second was 47.055, so it makes sense that this value supports the idea that a fault is not present. Looking at Figure 27, we see that this number of packets received is entirely normal behavior. On average, the system receives around 35 packets per second. When an interfering workload was introduced, it introduced instances where the packets received per second by the database server

raised into the range of about 60 to 140 per second—in two rare instances we see the value climb above 200.

The fourth highest contributing factor is the number of packets sent from the webserver, which was 19.738. This provided a marginal contribution towards the prediction of a fault by 0.09. This value supports the idea that the model over generalized this feature. From Figure 27, we see that a value of 19.738 is very low. On average, the number of packets sent by the webserver was around 40. The introduction of interfering workload increased this value into the range of 50-125 packets per second. However, the ANN model identified a low number of packets sent as an indication of a fault, so this logic likely comes from how the network identifies other fault, such as memory exhaustion. This same logic can be applied to the fifth highest contributing factor, which was packets sent per second from the database server.

4.2.4 Resulting Explanation of the Classifier Models

After reviewing the SHAP values, we can draw final conclusions about the classifier models that can be useful for users. The classifiers perform well under normal operating conditions, with a high accuracy rate of about 97% in identifying normal behavior. For detecting faults, the models rely on several key factors, including the amount of network traffic, CPU and memory performance in the web server, and idle time of the database disk.

For resource exhaustion, high CPU or memory utilization in the webserver is the most important contributing factor, while for unscheduled replication, the utilization of the database disk is the key factor. In these cases, the model provides trustworthy results in response to high utilization of these resources.

However, when it comes to identifying interfering workload, predictions should be taken with caution. If the workload requires a high amount of network responses from the software

system, then the network may not always identify the fault correctly. This is because a low amount of response packets sent from the system is a key metric in cases of resource exhaustion. Therefore, the classifiers may fail to identify an interfering workload fault if the workload requires a high amount of network responses.

Chapter 5: Limitations of Work

The research presented in this thesis has several limitations. One limitation is the limited variety in the load test experiments. The classifier models were intended to predict whether or not a fault was present during any load test, so the output was binary. This output grouped all potential faults into one class. This implementation was chosen intentionally to allow a proper demonstration and application of a SHAP explanation, but this resulted in the usage of classifiers that were limited in scope.

Another limitation of this research is the computing time allowed for SHAP value calculation. Computing the SHAP values with KernelSHAP is very computationally expensive and time-consuming depending on the number of data instances used for background sampling. The more samples used, the more the SHAP values consider the marginal contribution in regard to the entire instance space; thus, making the SHAP values more precise. However, the amount of computational time is not always worth the slight boost in precision; estimating SHAP values with a subset of instances is often sufficient enough to provide a quality explanation of the model's decision-making process through feature contribution. Therefore, random sampling was utilized for the calculation of SHAP values in this research. Along with this, 250 background samples was the number chosen for visual representations of SHAP values. This allowed for many data instances used in computations to not be shown within the visual representations, such as the bee swarm plots. However, this was necessary to keep the figures simple to understand and explain.

Chapter 6: Conclusions and Future Work

6.1 Conclusion

AI has transformed the way that data is analyzed and processed, and its rapid growth now allows for the quick analysis of vast amounts of data in nearly every industry, leading to the automation of many processes. As AI continues to see performance improvements and growing usage in many fields, implementations of the underlying ML algorithms have become increasingly complex. Since many people cannot understand the reasoning behind AI predictions due to the opacity of modern-day ML algorithms, the field of Explainable AI (XAI) has gained significant attention over the past years. This thesis intended to build upon the existing body of work in the field of XAI by demonstrating its application to a real-world issue. A post-hoc XAI algorithm, SHAP, was utilized to explain black-box classifiers which were used to identify load test failures in large scale software systems.

This research concluded that SHAP was sufficient at providing an interpretable explanation of the classifier models using SHAP values (mean marginal contribution). Utilizing SHAP values allows for the models' decision-making processes to be understood, and this informed the process of updating and improving the classifier models to be trustworthy.

6.2 Future Work

For this research, there is future work which could expand upon the current research. An investigation into other XAI techniques could potentially help to improve the quality and ease-of-use of the resulting explanations being provided to performance analysts. For example, LIME considers interpretability when optimizing explanations, so it can utilize a variety of surrogate models and graphical explanations. Comparisons between the linear regression model used in this thesis and other surrogate models may help to find more interpretable explanations. Along

with this, LIME and SHAP both consider local explanations while the calculation of Shapley values must consider the entire feature space and instance space. As mentioned in Chapter 5, random sampling of instances across the instance space was used, and a threshold was set to reduce the overall computation time. Making comparisons between true Shapley values and the estimated SHAP values could also help to improve this research and justify the methodology of utilizing linear regression as a surrogate model.

Also, feature selection techniques could be implemented to examine new data. In this research, one feature—latency—was removed due to its exceedingly high correlation to the prediction class across the entire instance space. However, it should always be acknowledged that the addition of new data could cause potential changes in correlation. Therefore, a feature selection technique can be implemented to justify the removal of latency through future changes.

References

- [1] J. Hurwitz, M. Kaufman and B. Adrian, *Cognitive Computing and Big Data Analytics*, 1st ed. Wiley, 2015.
- [2] “What is Machine Learning?” IBM. <https://www.ibm.com/topics/machine-learning> (accessed Mar. 2023).
- [3] “What is Machine Learning?” University of California, Berkeley. <https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/> (accessed Feb. 2023).
- [4] B. Mahesh, “Machine Learning Algorithms - A Review,” *International Journal of Science and Research*, vol. 9, no. 1, 2020.
- [5] I. Mihajlovic, “How Artificial Intelligence is Impacting our Everyday Lives.” Towards Data Science. <https://towardsdatascience.com/how-artificial-intelligence-is-impacting-our-everyday-lives-eae3b63379e1> (accessed Nov. 2022).
- [6] “What is Explainable AI?” Carnegie Mellon University. <https://insights.sei.cmu.edu/blog/what-is-explainable-ai/> (accessed Jan. 2023).
- [7] A. Adadi and M. Berrada, “Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI),” *IEEE Access*, vol. 6, 2018.
- [8] C. Molnar. “Interpretable Machine Learning: A Guide for Making Black Box Models Explainable.” Github. <https://christophm.github.io/interpretable-ml-book/> (accessed Feb. 2023)

- [9] L. S. Shapely, "A Value for n-Person Games," *Contributions to the Theory of Games*, vol. 2, 1952.
- [10] A. Bhattacharya. "Understand the Workings of SHAP and Shapely Values Used in Explainable AI." Towards Data Science. <https://towardsdatascience.com/understand-the-working-of-shap-based-on-shapley-values-used-in-xai-in-the-most-simple-way-d61e4947aa4e> (accessed Dec. 2022).
- [11] S. Feingold. "The European Union's Artificial Intelligence Act, explained." <https://www.weforum.org/agenda/2023/03/the-european-union-s-ai-act-explained/> (accessed Mar. 2023).
- [12] C. Mills. "PayPal's Two-Hour Outage Could Have Cost Tens of Millions of Dollars." <https://gizmodo.com/paypals-two-hour-outage-could-have-cost-tens-of-million-1739893030> (accessed Dec. 2022).
- [13] "Restarts Cited in Skype Failure." *New York Times*. Accessed Dec. 2022, [Online]. Available: <http://www.nytimes.com/2007/08/21/business/worldbusiness/21skype.html>
- [14] "Facebook Accounts Unavailable." Facebook. <http://facebooklogin.net/help/facebookaccount-unavailable/>.
- [15] M. Haroon and A. Hassan, "Supporting software evolution using adaptive change propagation heuristics," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2008)*, pp. 177-186, 2008.

- [16] A. Hassan and R. Holt, "Predicting Change Propagation in Software Systems," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2004)*, pp. 284-293, 2004.
- [17] H. Malik, B. Adams and A. Hassan, "Pinpointing the subsystems responsible for the performance deviations in a load test," in *Proceedings of the IEEE 21st International Symposium on Software Reliability Engineering (ISSRE)*, pp. 201-210, 2010.
- [18] U. Fayyad, "A Data Miner's Story – Getting to Know the Grand Challenges," *13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2007.
- [19] M. Knop, J. Schopf and P. Dinda, "Windows Performance Monitoring and Data Reduction Using WatchTower," in *Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing (HPDC11)*, pp. 23-33, 2002.
- [20] T. Dharmesh *et al*, "A framework for measurement based performance modeling," in *Proceedings of 7th International Workshop on Software and Performance*, pp. 55-66.
- [21] S. Zaman, B. Adams and A. Hassan, "A qualitative study on performance bugs," in *Proceedings of 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pp. 199-208, 2012.
- [22] "Dell DVD Store Database Test Suite." <https://linux.dell.com/dvdstore/> (accessed Dec. 2022).

- [23] S. Pretet and P. Narasimhan, “Causes of Failures in Web Applications,” *Parallel Data Laboratory, Carnegie Data Laboratory*, 2005.
- [24] Z. Jiang, “Automated analysis of load testing results,” in *19th International Symposium on Software Testing and Analysis*, pp. 143-146.
- [25] “Perfmon.” Microsoft. <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/perfmon> (accessed Feb. 2023).
- [26] “Download HeavyLoad for Windows 7.” Jam Software. <https://www.jam-software.com/heavyload/heavyload-windows-7.shtml> (accessed Jan. 2023).
- [27] “Packet Sender - Download.” Packet Sender. <https://packetsender.com/download> (accessed Feb. 2023).
- [28] “Install TensorFlow 2.” TensorFlow. <https://www.tensorflow.org/> (accessed Nov. 2022).
- [29] “Installing scikit-learn.” scikit-learn. <https://scikit-learn.org/stable/install.html> (accessed Dec. 2023).

Appendix A: IRB Approval Letter



Office of Research Integrity

March 9, 2023

Eric Shoemaker
1540 4th Avenue
Huntington, WV 25701

Dear Eric,

This letter is in response to the submitted thesis abstract entitled "*Leveraging Explainable Artificial Intelligence (XAI) to Understand Performance Deviations in Load Tests of Large Software Systems.*" After assessing the abstract, it has been deemed not to be human subject research and therefore exempt from oversight of the Marshall University Institutional Review Board (IRB). The Code of Federal Regulations (45CFR46) has set forth the criteria utilized in making this determination. Since the information in this study does not involve human subjects as defined in the above referenced instruction, it is not considered human subject research. If there are any changes to the abstract, you provided then you would need to resubmit that information to the Office of Research Integrity for review and a determination.

I appreciate your willingness to submit the abstract for determination. Please feel free to contact the Office of Research Integrity if you have any questions regarding future protocols that may require IRB review.

Sincerely,

Bruce F. Day, ThD, CIP
Director

WE ARE... MARSHALL.

One John Marshall Drive • Huntington, West Virginia 25755 • Tel 304/696-4303
A State University of West Virginia • An Affirmative Action/Equal Opportunity Employer

Appendix B: Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
CNN	Convolution Neural Network
CPU	Central Processing Unit
DB	Database
DS2	Dell DVD Store Application (Version 2)
DVD	Digital Versatile Disk
EU	European Union
I/O	Input/Output
LAN	Local Area Network
LIME	Local Interpretable Model-agnostic Explanations
LSS	Large-Scale System
matplotlib	Matplot Library for Python
MB	Megabyte
ML	Machine Learning
MSE	Mean Squared Error
perfmon	Windows Performance Monitor
QoS	Quality of Service
RNN	Recurrent Neural Network
SHAP	Shapley Additive exPlanations
SLA	Service Level Agreement
WS	Web Server
XAI	eXplainable Artificial Intelligence