Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2023) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

The 18th International Conference on Future Networks and Communications (FNC)
August 14-16, 2023, Halifax, Nova Scotia, Canada

Explaining the Unseen: Leveraging XAI to Enhance the Trustworthiness of Black-Box Models in Performance Testing

Eric Shoemaker[a], Haroon Malik[a]*, Husnu Narman[a], Jamil Chaudri[b]

[a]*College of Engineering and Computer Sciences, Marshall University, USA*

Abstract

This paper presents a novel approach to enhance the trustworthiness of black-box and interpretable models in performance testing through the application of Explainable Artificial Intelligence (XAI). The proposed approach utilizes the Shapley Additive Explanation (SHAP) algorithm as a surrogate model, enabling performance analysts to gain insights into the decision-making process of black-box machine learning models. By incorporating SHAP around black-box models, analysts can obtain explanations regarding the pass or fail status of a test, as well as identify the relative significance of performance data to the machine learning models. To evaluate the effectiveness of the proposed approach, load testing experiments were conducted on a real testbed, employing industry-standard benchmarks and manually injected performance bugs. The results demonstrate that the proposed approach significantly enhances the trustworthiness of machine learning models by providing interpretable explanations for their decision-making. Furthermore, the approach showcases its versatility across domains and requires minimal effort to operate, making it highly applicable in various contexts.

Keywords: Explainable Artifical Intelligence; SHAP; Load test; Performance test; Neural netwroks; Random forest

1. Introduction

Companies are confronted with the challenge of managing and interpreting vast volumes of data, surpassing human capabilities and impeding informed decision-making. Efficient utilization of big data to derive insights and inform

* Corresponding author. Tel.: +1-304-690-4002;
E-mail address: malikh@marshall.edu

decisions has become crucial for gaining a competitive edge [1]. Consequently, the field of artificial intelligence (AI) has experienced significant growth, enabling pattern recognition, event forecasting, data classification, and more at a large scale. The monitoring of large-scale systems (LSS) has become increasingly critical and complex, with AI proving valuable in analyzing system performance. AI systems can provide 24/7 monitoring, enabling performance identification and facilitating load testing analysis, which evaluates system performance under specific loads, defined as the rate at which transactions are submitted.

Although AI offers promising solutions for load testing and other domains, concerns have emerged regarding the trustworthiness of AI. As AI advances, underlying machine learning (ML) algorithms, such as artificial neural networks (ANN) and random forests, have become more complex, making them less interpretable. While these ML algorithms enhance predictive power, ensuring the quality of AI models becomes challenging. Existing methods include hiring domain specialists to analyze the models, but this is costly, time-consuming, and may be limited by the availability of domain experts. Another approach is comparing model predictions to historical data, which requires substantial domain knowledge and the analysis of vast datasets. The third method involves evaluating AI using performance metrics like prediction accuracy and error. However, this approach lacks specificity and fails to provide explanations for unsatisfactory predictions.

1.1. Problem Statement

Performance testing, particularly load testing of large-scale systems, can generate a significant volume of performance data (e.g., 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 explain 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 the previous section, this paper aims to explore the concept of using XAI to improve the trustworthiness of both black-box and interpretable models, particularly in the context of performance testing and for most of the black-box machine learning models. Specifically, the paper 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.

1.3. Paper Contributions

This research work/paper aims 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. The paper makes several contributions

to the field of XAI and performance testing:

1. The paper contributes to the XAI literature by demonstrating the successful application of SHAP in the performance testing domain. To the best of our knowledge, the paper 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 performance tests.
2. Unlike previous studies that used synthetic data or log traces from the internet, the paper conducts experiments on a testbed that uses industry-standard performance benchmarks. 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.
3. To ensure the replication of our results, each experiment is presented as a Jupyter Notebook that is easily accessible to the research community for reproducibility and fair comparison.

Overall, the paper 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.

2. Background Knowledge

The section provided high-level background knowledge related to performance testing, especially load testing and an XAI technique Shapley Additive exPlanation (SHAP) algorithm.

2.1. Shapley Additive exPlanation (SHAP)

Shapley Additive exPlanation (SHAP) is a framework for explaining the output of any machine learning model by assigning importance values to each input feature. It uses game theory to distribute the contribution of each feature to the model's output among all possible combinations of features. By doing so, SHAP provides a unified and consistent way to explain predictions across different models and datasets. SHAP values can be used to understand model behavior, debug errors, identify bias, and improve model interpretability and transparency. SHAP proposes that the SHAP values for each feature of an instance x can be represented as a linear model g [1][2]. The linear model is shown in equation (i). Where, $\emptyset_j$ represents the SHAP value for feature j, and M represents the total number of different features in the feature space.

$$g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (i)$$

The efficiency property of Shapely values ensures that the feature contributions for an instance add up to the difference of predictions between that instance and the average prediction overall. Shapely values have become a trustworthy explanation method and are the focus of using XAI in legal situations. However, calculating Shapely values can be computationally expensive, so approximations are often used.

2.2. Load Testing

Large-scale systems (LSS) like data centers and server farms have become increasingly large and complex. Service providers must monitor them continuously to ensure high availability and peak performance, as performance issues can result in significant monetary losses. Availability or performance issues can also impact business-customer relationships and result in costly lawsuits or the breakdown of the business model. Load testing remains the most important part of performance testing for LSS, and performance analysts use it to detect early performance problems in the system before they become critical field problems [3]. However, load test analysis practices are labor-intensive, prone to errors, and little attention has been given to effectively analyzing the results of load tests. Load testing is a critical process to assess system performance under given load(s) and to identify functional and performance issues that may have been missed during functional testing. Performance problems can lead to system crashes or becoming unresponsive, which can result in monetary losses for companies. Load testing involves simulating thousands of concurrent transactions using one or more load generators and monitoring the application under test closely. The collected performance data is analyzed and compared to documented or expected behavior to identify issues.

Table 1. Hardware Specifications

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

Table 2. Fault Injection into the Load Tests Experiments

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

Load testing consists of four phases: environment setup, load test execution, load test analysis, and report generation. Load test analysis involves comparing the results of a load test to another load test result or pre-defined thresholds using quantitative metrics. Performance analysts in LSS often use "rules of thumb" or historical information stored in performance test repositories to assist them in load test analysis. When other options are not available, analysts rely on their personal judgment. The challenges of load test analysis in LSS include a large volume of data obtained from various subsystems and time constraints[4][5]. Load testing presents several challenges related to organizing and presenting vast amounts of performance data, dealing with time constraints, and extracting the necessary information from past test results. The use of inconsistent terminology and inadequate test report information can also make conducting root cause analysis difficult. These challenges can make it challenging for analysts to evaluate performance deviations or anomalies, particularly when dealing with a significant amount of performance counter data. Additionally, the limited amount of time allocated for analysis, especially in the context of a tight release schedule, further complicates the load testing process [6].

3. The 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.

3.1. Benchmark System

The system under examination is the Dell DVD Store (DS2) application, an open-source e-commerce simulation designed for benchmarking Dell hardware[10]. The application includes standard e-commerce features such as user registration, login, product search, and purchase. DS2 has different versions to support various programming languages and databases. For this study, the PHP distribution and a MySQL database were used, and a mix of transactions was used as the load, including user registration, product search, and purchases.

3.2. Testbed

To replicate the load testing process of large-scale software systems, a testbed of three machines was used for the DS2 system. The first machine was used as a web server running Apache to host the web application for browsing and purchasing DVDs. The second machine was the database server, hosting a MySQL database that held the store inventory and customer account information. The third machine acted as a load generator, mimicking real-world users by generating concurrent loads on the servers. The machines were physically connected to a LAN, and their hardware specifications are shown in Table 1. 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 servers. The web server was configured to allow many concurrent user connections, and the site was served using Apache Web Service. The database server was installed with MySQL Community Edition 5.7, configured to allow many concurrent connections, and 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.

3.3. Fault Injection

To test the methodologies in practical scenarios, fault injection was performed. A fault category, such as software failures, hardware failures, or human/operator errors, was selected, and corresponding failure triggers were determined. In this paper, software and human errors were used, as they were identified to be the most common causes of failures [3]. Two failure triggers were selected from the list of common triggers identified by Pretet et al. and are listed in Table 2. The selection was based on their suitability for load-testing experiments. Meticulous analysis of each test is crucial to ensure that the system is meeting its intended service levels agreement, such as response time or latency, as performance analysts typically need to conduct a multitude of tests with specific workloads under particular hardware and software conditions for each release or build of an application.

3.4. Load Test Experiments

To evaluate the proposed methodology several load test experiments were performed.

1) *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 [9].
2) *Experiment-2* (Base-line load): The second experiment's purpose was to measure the performance of the software system under a normal operating load. Many users may utilize 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.
3) *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.
4) *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's 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 for one 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 [8]. 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.
5) *Experiment-5:* Resource exhaustion is not the only factor that 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 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 [8]. Within each 15-minute interval, transactions from the DVD store website were delayed, causing slower responses to users even when resource exhaustion of the web server was not present.

6) *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 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 [7]. 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.

4. Machine Learning Models

Two classifier models were developed to monitor the system and identify the presence of faults, specifically the four types of faults typical to software systems listed in Table 2. Supervised learning algorithms were used to analyze the data, as the tests were conducted in a controlled environment with known fault times. The chosen machine learning algorithms were artificial neural networks (ANN) and random forest, which are both considered black-box models and have gained significant popularity in recent years due to their ability to handle high-dimensional datasets and perform nonlinear transformations. For the ANN model, TensorFlow was used for much of the implementation. The Sequential model from TensorFlow Keras API was used to compile a simple classification ANN with a 3-layer architecture consisting of an input layer, a hidden layer, and an output layer. The hidden layer utilized an S-shaped curve for the activation function to allow the ANN to learn non-linear relationships between the input features. For the output layer, a linear activation function was chosen, which outputs 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. This model utilized an ensemble of 100 decision trees and was implemented using Scikit Learn, a popular library implemented in Python to provide machine learning and data analysis tools. A visual representation of the random forest classifier was created using two random trees from the ensemble. To train the machine learning models, the data's feature values were scaled to [0, 1] using min-max scaling to ensure equal importance of all features. The data was divided into a training (70%) and testing (30%) group. The ANN model used Adam optimization and MSE loss, while the random forest model used bagging and mean accuracy. After training, the model's accuracy and MSE loss (ANN only) were calculated. The models were evaluated on the testing set, with both models achieving 99% mean accuracy.

5. Results and Discussion

The accuracy and loss metrics for the classifier models indicate that they have learned the data well, with mean prediction accuracies of 99% or greater even on unseen data. However, high accuracy and loss values alone are not enough to ensure model trustworthiness, as they could indicate overfitting. This occurs when a model memorizes specific patterns in the data rather than generalizing, resulting in incorrect predictions on unseen data. XAI techniques like SHAP can be used to gain insights into the decision-making process of black-box models such as ANN and Random Forest. SHAP was used to identify the contribution of each feature to the model predictions, and a summary plot was created to visualize these values as shown in Figure 1 and Figure 2. The SHAP values for most features for both Random Forest and ANN were near zero, indicating little to no contribution to the predictions. In contrast, the SHAP value for latency was very high, indicating that the ANN and Random Forest models were making decisions almost entirely based on the amount of latency present. To examine this further, individual SHAP values were plotted using a bee swarm plot shown in Figure 3, which showed that the ANN model's predictions of faults were heavily influenced by latency. However, relying solely on latency to make predictions could be unreliable in a production environment since it is affected by many factors and could vary significantly. Therefore, this behavior could affect the trust in the models and cause issues in practical applications. After using KernelSHAP explanations, it was discovered that the latency feature had a high correlation with the class label, leading to the development of a method that gave less contribution to the latency feature.

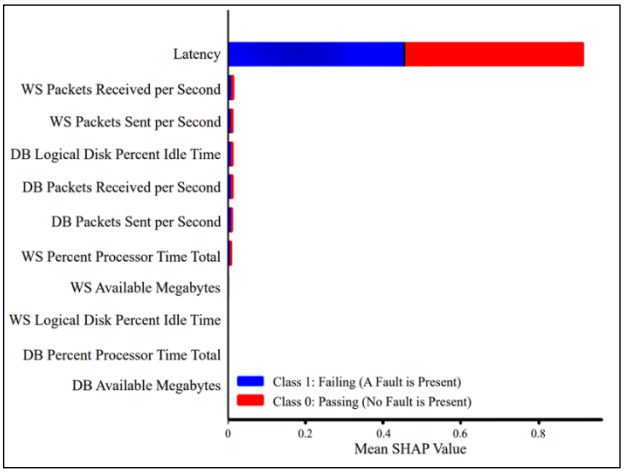

Figure 1. Mean SHAP Values for ANN Model

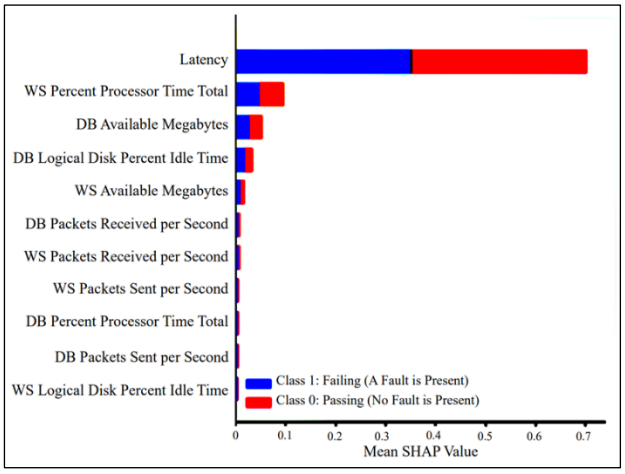

Figure 2. Mean SHAP Values for Random Forest Model

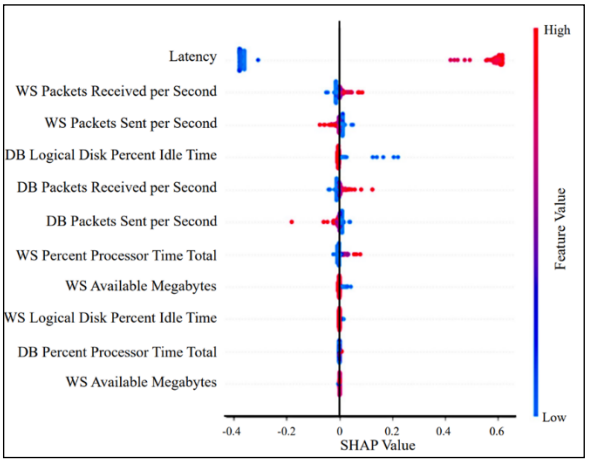

Figure 3. SHAP Values Towards Predictions of Fault Identification

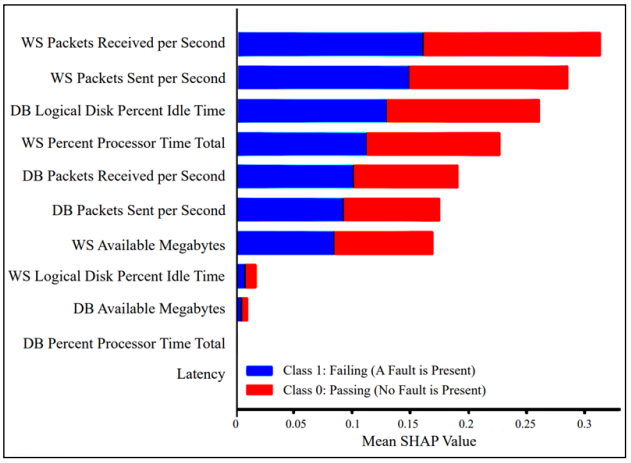

Figure 4. Mean SHAP Values Over Testing Set for Updated ANN Model

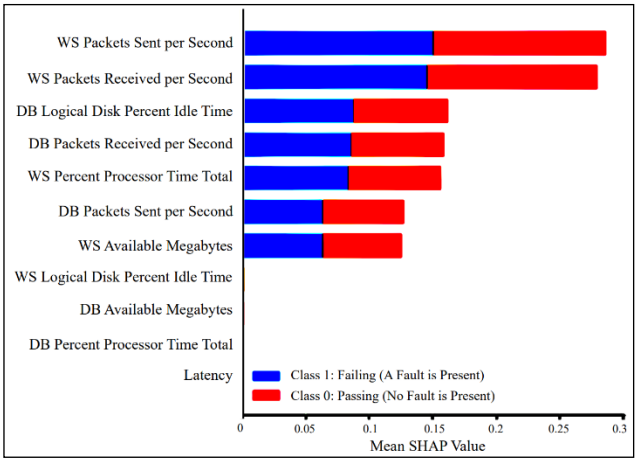

Figure 5. Mean SHAP Values Over Testing Set for Updated Random

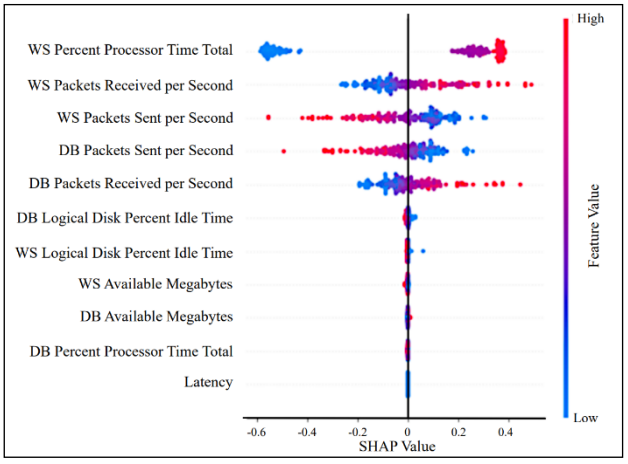

Figure 6. SHAP Values Towards Fault Identification During CPU

To determine if the models could identify patterns in the other features without the high correlation of latency, the models were retrained with the latency feature value removed for every instance in the data set. The mean accuracy

and loss were recorded for the new models. The ANN model achieved a training accuracy of 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. The random forest model had a mean training accuracy of 99%, and the mean testing accuracy was 96%. The SHAP values plots in Figure 4 and Figure 5 demonstrated that the updated models considered several performance metrics such as CPU and memory utilization. Additionally, specific SHAP values for instances recorded during specific experiments were examined, which showed that the model had a solid understanding of the different features that contribute to fault prediction as seen in Figure 6.

6. Conclusion and Future Work

AI has significantly changed the way data is processed, with its rapid growth leading to automation in numerous industries. However, the complexity of modern ML algorithms has made it difficult for people to comprehend the reasoning behind AI predictions, resulting in increased interest in Explainable AI (XAI). This paper demonstrates the application of XAI to a real-world issue by utilizing SHAP, a post-hoc XAI algorithm, to explain black-box classifiers used in identifying load test failures in large-scale software systems. The research concludes that SHAP is effective in providing interpretable explanations of the classifier models through SHAP values. By understanding the models' decision-making processes, it became possible to update and improve the classifier models to make them trustworthy.

Part of the future work involved exploring other XAI techniques to improve the quality and user-friendliness of the explanations provided to performance analysts. For example, LIME considers interpretability when optimizing explanations, utilizing surrogate models and graphical explanations. Comparing the linear regression model used in this study with other surrogate models may lead to more interpretable explanations. Additionally, while SHAP and LIME consider local explanations, Shapley values must consider the entire feature and instance space. Comparing true Shapley values with estimated SHAP values can improve this research and justify the use of linear regression as a surrogate model. Furthermore, implementing feature selection techniques can examine new data. Although one feature, latency, was removed due to a high correlation with the prediction class, the addition of new data could alter the correlation. Therefore, feature selection techniques can be used to justify the removal of latency with future changes in the data. These future research avenues can strengthen the results of this study and pave the way for further advancements in XAI techniques.

References

[1]. Interpretable Machine Learning: A Guide for Making Black Box Models Explainable. Available: https://christophm.github.io/interpretable-ml-book/.

[2]. A. Adadi and M. Berrada, "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)," IEEE Access, vol. 6, 2018.

[3]. S. Pretet and P. Narasimhan, "Causes of Failures in Web Applications," Parallel Data Laboratory, Carnegie Data Laboratory, 2005.

[4]. 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.

[5]. H. Malik, Z. M. Jiang, B. Adams, A. E. Hassan, P. Flora and G. Hamann, "Automatic Comparison of Load Tests to Support the Performance Analysis of Large Enterprise Systems," *2010 14th European Conference on Software Maintenance and Reengineering*, Madrid, Spain, 2010, pp. 222-231, doi: 10.1109/CSMR.2010.39.

[6]. Haroon Malik. 2010. A methodology to support load test analysis. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2 (ICSE '10). Association for Computing Machinery, New York, NY, USA, 421–424. https://doi.org/10.1145/1810295.1810408

[7]. Packet Sender. Available: https://packetsender.com/., Accessed November 2022.

[8]. HeavyLoad for Windows 7. Available: https://www.jam-software.com/heavyload/heavyload-windows-7.shtml., Accessed December 2022.

[9]. Perfmon. Available: https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/perfmon., Accessed December 2022.

[10]. Dell DVD Store Database Test Suite. Available: https://linux.dell.com/dvdstore/. Accessed October 2022.