Predict Performance of Processes in Operating Systems Using Machine Learning Techniques

Jiaqian Huang
Department of Computer Science and Engineering
the Ohio State University
Columbus, Ohio, United States
huang.2366@buckeyemail.osu.edu

***Abstract*—This work presents a comprehensive study on the use of machine learning techniques to predict process performance in operating systems. The proposed approach employs a range of process attributes, such as CPU usage, memory usage, and I/O operations, as input features to train and evaluate machine learning models. Various machine learning algorithms, including Linear Regression and Random Forest Regression, are tested to identify the most effective approach for predicting process performance. The performance of the approach is evaluated using a real-world dataset collected from both the Linux and macOS operating systems on a local machine. The results demonstrate that machine learning-based methods can accurately predict process performance. By implementing and evaluating the machine learning models, this study highlights the potential of machine learning in improving the performance and efficiency of operating systems.**

***Index Terms*—Machine learning, Process Performance, Operating System**

I. Introduction

The performance of processes in an operating system is a crucial factor that determines the overall efficiency and responsiveness of the system. Processes are tasks or programs executed by the operating system, and their performance can have a significant impact on the system's overall performance. Efficient allocation of resources, such as CPU time, memory, and I/O operations, is necessary to ensure that each process runs smoothly without delays or crashes.

Predicting process performance provides numerous benefits, including early detection of performance issues, improved resource utilization, better capacity planning, reduced downtime, enhanced user experience, and cost savings. To make accurate predictions, it is essential to monitor various process metrics, such as CPU utilization, memory usage, I/O throughput, and response time. System configurations can also be adjusted as needed to optimize performance, enabling processes to run efficiently and effectively.

Users on any operating system can check process information in the terminal using built-in command lines. This information typically includes the unique process ID for each task, the username of the task owner, task priority, total virtual memory used, actual physical memory consumption, shared memory size in kilobytes, task state, CPU and memory usage, CPU time, and the command being run. By leveraging large amounts of data with multiple attributes, machine learning algorithms can identify patterns and make accurate predictions about process performance, which motivates researchers to design and develop possible predictive models.

In this work, the application of predictive machine learning models to estimate the performance of processes is investigated. This work demonstrates that the data collected on a local machine can be generalized to represent the performance of processes on machines with different hardware specifications. Through experiments, Linear Regression and Random Forest Regression are efficient in estimating the performance by using the process information on both Linux and macOS operating systems. Furthermore, this study concludes that the performance of processes can be accurately predicted using data collected by any user on both Linux and macOS operating systems.

The remaining sections of this paper are structured as follows. Section II provides an overview of the related work in predicting performance. Section III outlines the study's design, including the methods used to collect data on both Linux and macOS operating systems and the predictive machine learning models employed. In Section IV, a detailed evaluation of experiments is presented, including the setup and results. Section V discusses the challenges encountered during the experiments. Finally, in Section VI, the findings are summarized, and a conclusion is provided for this work.

II. Related Work

Machine learning algorithms have gained popularity in accurately predicting performance and behavior in computer science. Previous works, such as Fu, Gupta, Mittal and Ratnasam [6], applied machine learning to predict application and use-case performance, but the prediction error remained high for realistic scenarios. Other studies have applied machine learning algorithms to predict computer operating system security, including whether a newly discovered vulnerability would enable attackers to cause denial of service [2] and developing a light-weight machine learning engine in the kernel space component [4]. Aslam, Sarwar and Batool [5] focused on improving CPU scheduling by applying Bayesian Decision Theory. However, there is a need for more general models that can accurately predict process performance across different operating systems and user scenarios. This study aims to address this gap by developing predictive machine learning

models for both Linux and macOS operating systems using data from diverse users.

III. Design

Here's an overview of each step in the design:

- Data collection: This step involves gathering process-related data from both Linux and macOS operating systems. The data are used as input features for training and evaluating machine learning models.
- Data preprocessing: After collecting the data, it is necessary to clean and preprocess it to ensure that it is consistent and ready for use in machine learning models. This involves removing any duplicates or incomplete data, normalizing the data to a common scale, and encoding categorical variables.
- Machine learning model building: In this step, several machine learning models are developed and trained by using the preprocessed data. The goal is to identify the most suitable approach for predicting process performance. Various machine learning algorithms are experimented, including Linear Regression and Random Forest Regression.
- Evaluation: To evaluate the accuracy of the machine learning models, couple evaluation metrics are applied on models which use real-world data. This will help verify the performance of our models and identify any potential issues.

A. Data Collecting

Data collecting is an essential part of in this project. Process-related data are required to be collected from various operating systems.

To collect the necessary data, a combination of built-in system utilities and custom scripts is applied in this work, which allow users to collect process attributes such as CPU usage, memory usage, I/O operations, and response time.

To ensure the accuracy and comprehension of data, collecting data from multiple operating systems at various intervals takes the next step. It is also recommended to cover a variety of processes data to represent different behaviors and collect them at different hours of a day such as peak hours and off-peak hours.

Specifically, Linux operating system and macOS operating system are chosen and focused on in this work, as they are widely used in real-world settings and will provide us with representative data.

1) Linux Operating System A kernel module, also referred to as a driver, is a software component that can be dynamically loaded or unloaded into the kernel of an operating system. It has the capability to interact with hardware components and offer services to user-space applications. The kernel module is coded in the C language, a low-level programming language that conforms to the kernel's programming interface (API) and driver model.

In this project, a kernel module is implemented to gather data generated by the operating system and store it in a virtual file on the Linux operating system. Additionally, a C file is executed in user-level to export the contents of the virtual file into an actual file.

The motivation behind creating a kernel module to collect process-related data on the Linux operating system is that it can be dynamically loaded and unloaded without requiring a reboot of the operating system. This flexibility in managing system resources does not impose a heavy burden on the kernel of the Linux operating system to handle.

2) macOS Operating System While it is technically possible to create a kernel module on macOS, it is generally not recommended due to the tightly controlled kernel architecture, the proprietary nature of the kernel code, and the availability of high-level programming tools and libraries for creating user-space applications. As a result, the kernel module is not loaded on macOS. Instead, one Python script is developed to gather data on the macOS operating system.

One advantage of using Python in a macOS environment is that it is relatively simple because the macOS operating system includes pre-installed development tools like a C compiler and the XCode development environment. These tools are critical for constructing Python from source code, and their availability on macOS makes it effortless to set up and use Python on a MacBook.

B. Data Preprocessing

After collecting process-related data from both the Linux and macOS operating systems, it is essential to conduct a comprehensive cleaning and preprocessing of the data to ensure consistency. This involves removing any irrelevant data that is not related to processes. Handling missing data is another important step, which can be achieved by imputing values based on the mean or median of the respective attribute. Finally, techniques like one-hot encoding or label encoding can be used to encode categorical variables such as process state into numerical values, allowing machine learning models to process them effectively.

By following these data cleaning and preprocessing steps, machine learning models can be trained on high-quality and consistent data, which improves their accuracy and performance in predicting process performance on operating systems.

C. Predictive Models

In this project, data collected from both the Linux and macOS operating systems are used to build predictive models to estimate the performance of processes in each respective operating system. 2 types of models are developed: Linear Regression and Random Forest Regression.

1) Linear Regression Linear Regression is one of the simplest and most commonly used linear models that aims to estimate the linear relationship between the predictor variables and the outcome variable. The primary objective of Linear Regression is to identify a linear equation that closely fits the data and best describes the relationship between the predictor variables and the response variable. It allows for

the identification of the most important predictor variables, estimation of the magnitude and direction of their effects, and avoidance of overfitting by regularizing the estimates. The multiple Linear Regression equation is as follows [3]:

$$y = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_n x_n + e \tag{1}$$

where y is dependent variable or predicted outcome variable, and $x_1, x_2, \ldots, x_n$ are independent variables or predictors, and b_0 is intercept or expected value of y when all independent variables are equal to 0, and $b_1, b_2, \ldots, b_n$ are the regression coefficients which contributes the effect of each independent variable on y, and e is the error term which represents the part of y that is not explained by independent variables.

In this study, the scikit-learn (sklearn) Python library is utilized to apply Linear Regression. The scikit-learn (sklearn) Python library provides a comprehensive set of tools for data preprocessing, model selection, and evaluation, which are utilized to build models that predict the target variable based on a set of input features.

2) Random Forest Regression Random Forest Regression is a popular machine learning algorithm used for regression analysis. It is an extension of the decision tree algorithm that builds multiple decision trees and merges them together to improve accuracy and reduce overfitting. The Random Forest Regression works by creating a random sample of the training data and building decision trees on each sample. The algorithm then combines the results of all decision trees to make predictions, reducing variance and improving prediction accuracy.

One of the key advantages of Random Forest Regression is its ability to handle high-dimensional data and capture complex nonlinear relationships between input and target variables. It is also robust to outliers and missing data, making it a versatile choice for a wide range of regression tasks [1].

In this study, Random Forest Regression was applied using the scikit-learn (sklearn) Python library. Overall, the algorithm is proved to be a powerful tool in predicting process performance, particularly when dealing with complex datasets.

3) Evaluation and Validation Some of machine learning models do not provide an estimated predictive error for themselves. Therefore, it is necessary to introduce an evaluation metric to ensure that the performance of a predictive model provides accurate results. Typically, an evaluation metric requires training the model on a dataset, using the model to make predictions on a holdout dataset that is not used during training, and then comparing the predicted values with the expected values in the holdout dataset.

In this work, R-squared [3] is applied as one of evaluation metrics. The form of it is Equation [3]:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{2}$$

where R^2 is the coefficient of determination, SS_{res} is the sum of squares of the residuals, and SS_{tot} is the total sum of squares.

R-squared does not consider any biases that may be present in the data. Thus, a well-built model may have a low R-squared value. When the model does not fit the data, it may get a high R-squared value.

Besides, Mean Square Error (MSE) [3] is applied as well. The following equation shows the formula for calculating Mean Square Error (MSE) [3]:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3}$$

where n is the number of observations in the dataset, and y_i is the actual value of the dependent variable for observation i, and $\hat{y}_i$ is the predicted value of the dependent variable for observation i.

The primary advantage of using Mean Square Error (MSE) is that it is easier to calculate the gradient compared to Mean Absolute Error (MAE), which requires complicated programming tools to calculate the gradient. It is a good metric to use if there are many outliers in the dataset. Moreover, by taking the square of errors, larger errors are emphasized more than smaller errors, allowing for greater focus on larger errors. The Mean Square Error (MSE) is expressed as the average of the squared differences between the predicted and expected values. The closer the model predictions are to the observations, the smaller the Mean Square Error (MSE) will be.

However, one of drawbacks of using only one testing set is that the test Mean Square Error (MSE) varies greatly depending on which observations are used in the training and testing sets.

To avoid this problem, cross-validation, a specific form known as k-fold cross-validation, is utilized for comparison which can fit a model several times using a different training and testing set each time and then calculate the test Mean Square Error (MSE) to be the average of all the test Mean Square Error (MSE)'s. The relative function is available in the scikit-learn (sklearn) Python library. Specifically, repeated K-fold Cross-Validation will be applied which k-fold cross-validation is simply repeated n times. Each time the training and testing sets are shuffled, so this further reduces the bias in the estimate of test MSE although this takes longer to perform than ordinary k-fold cross-validation. This configuration can evaluate the performance of machine learning models on a limited sample of data in a more robust manner. The equation derives from (3) and takes the form that:

$$TestMSE = \frac{1}{k} \sum_{i=1}^{k} MSE_i \tag{4}$$

where k is number of folds and MSE_i is $TestMSE$ on the ith iteration.

In addition, model validation for Linear Regression model can be taken into account. For instance, Linear Regression needs to match couple assumptions. The relationship between the dependent and independent variable needs to be linear [3].

The residual error should be normally distributed, and the mean of residual error should be 0 or close to 0 as much as possible [3]. Linear Regression requires all variables to be multivariate normal [3]. The data need to be checked to be homoscedastic meaning the residuals are equal across the regression line [3].

Besides, multicollinearity may occur when the independent variables are too highly correlated with each other. Variance Inflation Factor (VIF) identifies correlation between independent variables and strength of that correlation [3]. If Variance Inflation Factor (VIF) is below 5, no investigation is required to correct the multicollinearity. Variance Inflation Factor (VIF) is given by [3]:

$$VIF_i = \frac{1}{1 - R_i^2} \tag{5}$$

where R_i^2 is the coefficient of determination mentioned in (2) that regresses the ith predictor variable on all other predictor variables. The Variance Inflation Factor (VIF) measures the degree to which the variance of the estimated regression coefficient of the ith predictor variable is inflated due to collinearity with other predictor variables.

IV. Evaluation

A. Experiment Setup

The experimental setup involves collecting the data, preprocessing the dataset, implementing predictive models, evaluating and validating their performance using multiple metrics, and measuring and comparing their computational efficiency.

1) Data Collecting and Input Values on Both Operating Systems Before installing kernel module, proclog.c, a file built for the kernel module, was compiled. It contains several kernel functions to facilitate the logging of information related to processes into a virtual file. During the compilation process, several files with the same name but different file types were generated. The file with the .ko extension is the critical file used for installation.

To start logging data, the .ko file was loaded on Linux operating system. The load process was simple and quick by executing the make file command containing custom actions. It was added to the list of running kernel modules, and no additional steps were required to collect process-related information. When the .ko file was loaded successfully, the user was reminded via a message. If some issues occurred after installation, the user was able to execute the make file command to unload the kernel module.

The kernel module created a virtual file stored in the "/proc/" folder. This file can be written to by kernel-level functions but not by user-level applications. As the Linux environment was active after installation, the virtual file was consistently updated by the kernel module to log process-related data. The file grew in size dramatically in a very short time. However, due to its attribute as a virtual file, it did not take up significantly large space on the hard disk.

Since kernel modules are usually not designed to write actual files, as they operate at a low level in the operating system and are mainly used to interact with hardware components, system resources, and other kernel components, a user-level C file was designed to write the contents of the virtual file into an actual file. Furthermore, because it is not possible to export comprehensive data from virtual data due to its continuous update by the kernel module for data logging, an input step for the user to manually input a positive integer was created to set up the number of lines to export. The kernel module operated while the system was on. When the file was open for exporting, the read never reached the end, and the program for data export never terminated. In this experiment, the input number was 3000 which represented the user requested 3000 records of process-related information from virtual file.

On macOS Operating System, before the execution of Python program built for data collecting, it was required to install Python-related packages for program to import and use. In the program, a library called psutil was applied which can retrieve process-related data with a few lines of code. The program set up an input step for user to choose an interval and end time for data collection. Python program retrieved process data every interval that the user input and stored the data in a file in csv format. The entire process for data collection finished within the user-specified time. In this experiment, the input interval was set as 5 seconds and the time for data collection finished in 30 seconds.

Any user is capable of generating own dataset containing process-related information on either the Linux or macOS operating systems, provided that the user has access to the relevant operating system. Since the performance of processes may vary between different computers, the generated data may also differ. The dataset used in the experiment was collected using a personal laptop.

2) Data Preprocessing and Input Values In this experiment, the dataset was subjected to several preprocessing steps to make it suitable for machine learning models.

For the Linux operating system, preprocessing was performed during the export process to ensure compatibility with common data analysis tools. Each row was thoroughly checked, and all white spaces were replaced with commas to match the CSV file format. This eliminated the need for an additional TXT to CSV conversion process. Conversely, data collection on macOS operating systems was streamlined using the csv Python library. This library enabled writing the header of a CSV file and directly appending records into rows, requiring no additional modifications to prepare the dataset in CSV format.

To prepare the dataset for machine learning models, several preprocessing steps were taken. Firstly, the dataset was checked for missing data, and unnecessary data was dropped, such as columns containing the same data for all rows. Next, label encoding was used to convert categorical data to numerical data, since one-hot encoding led to high memory consumption due to the large number of categories. Besides, all rows that had outliers in at least one column were removed.

To enhance the user experience, a feature was implemented where the program prompted the user to input the data to

process and the type of model to build upon execution. This approach reduced the wait time for the user, as the model building process took several minutes or longer. By allowing the user to specify desired model before the processing began, the program was able to let user wait for one model to be generated.

In both model building processes, the dataset was then split into training and testing sets using a 3:1 ratio and a random seed of 42 was set, which is a commonly used integer for random seeds. Setting the random seed ensures that other developers who use the same dataset and source code can reproduce the exact output. The training set was used to train the predictive models, while the testing set was used to evaluate their performance.

Overall, these preprocessing steps ensured that the dataset was of high quality and suitable for use in machine learning models.

3) Predictive Models Implementing If the user chose Linear Regression for process prediction, in preparation for building Linear Regression model, several specific steps were taken to ensure that the data was appropriate for this type of analysis. To scale and center the features of a dataset, standard scaler was applied and made data prepared in a standardized distribution of features, ensuring that each feature contributed equally to the learning process and the impact of outliers was reduced. Next, the linear relationship was checked by scatter plotting "Actual value Vs Predicted value" and multivariate normality was checked with quantile-quantile (q-q) plot. "Residual value Vs Fitted value" scatter plot was checked too. Another step was to check for multicollinearity, which can reduce the precision of estimated coefficients and weaken the statistical power of regression models. To address this issue, highly correlated independent variables were identified and removed using the Variance Inflation Factors (VIF) technique. A check function was used to calculate the Variance Inflation Factors (VIF), ensuring that the data had low to moderate levels of multicollinearity. Linear Regression assumes that there is minimal or no multicollinearity in the data, so these steps were crucial to ensure the validity of the models.

If choosing Random Forest Regression, the user needed to expect a longer processing time because there were several parameters to set up, and the GridSearchCV function from the scikit-learn (sklearn) Python library required time to calculate and select the best parameters from various options. The GridSearchCV function was utilized to select the optimal hyperparameters for the Random Forest Regression model on the Linux and MacOS data. The selected parameters for the Linux data were that $maxDepth$ was 7, and $maxFeatures$ was $sqrt$, and $minSamplesLeaf$ was 1, and $nEstimators$ was 500, and $randomstate$ was 42. Similarly, for the MacOS data, the GridSearchCV function selected parameters that $maxDepth$ was 7, and $maxFeatures$ was $sqrt$, and $minSamplesLeaf$ was 1, and $nEstimators$ was 300, and $randomstate$ was 2017. By utilizing these parameters, the Random Forest Regression model was able to achieve optimal performance for the given datasets.

B. Experiment Result

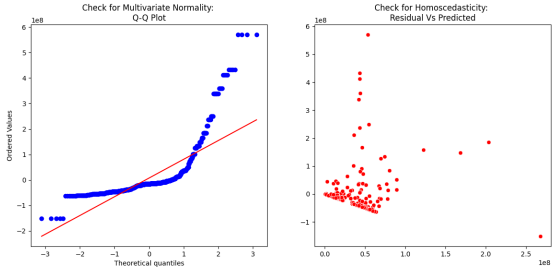

Fig. 1. Linear Regression check applying Linux data.

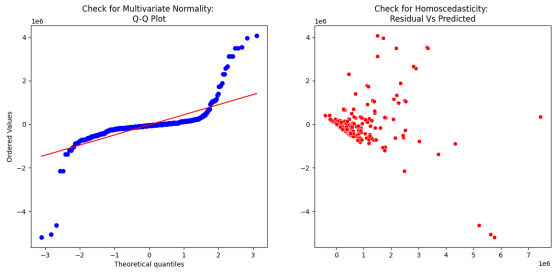

Fig. 2. Linear Regression check applying MacOS data.

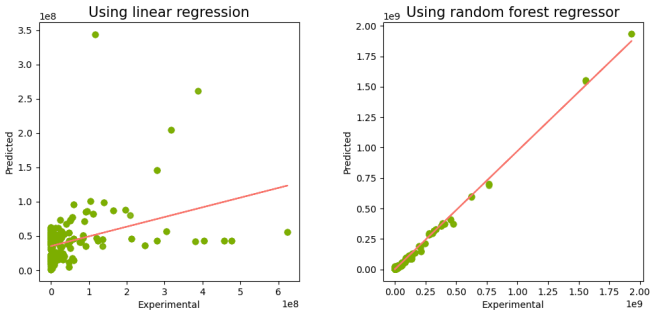

Fig. 3. Machine Learning Algorithms applying Linux data.

TABLE I
EVALUATION METRICS CALCULATED ON LINUX DATA

Methods	Linear Regression	Random Forest Regression
Test MSE	8.805821e+15	3.226405e+14
Test RMSE	9.383933e+07	1.796220e+07
Test R2	0.173071	0.99298
MSE Mean	7.684483e+15	3.127584e+14
RMSE Mean	8.733552e+07	1.760835e+07

After analyzing Fig. 3 and Fig. 4, it is evident that Linear Regression was not an effective machine learning algorithm for the Linux and macOS datasets. The data exhibited insufficient linearity, which resulted in inaccurate predictions. Conversely, Random Forest Regression algorithm proved to be successful in fitting the data and produced better results.

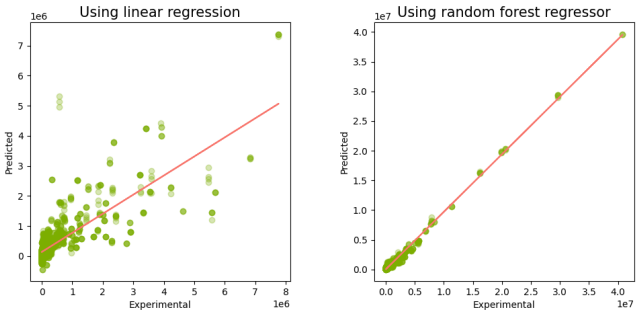

Fig. 4. Machine Learning Algorithms applying MacOS data.

TABLE II
EVALUATION METRICS CALCULATED ON MACOS DATA

Methods	Linear Regression	Random Forest Regression
Test MSE	3.895207e+11	6.636878e+10
Test RMSE	624115.918959	257621.382557
Test R2	0.514342	0.992468
MSE Mean	3.346303e+11	9.660352e+10
RMSE Mean	576119.614839	302881.154619

Based on Table I and Table II, Random Forest Regression outperformed the other models in terms of both training and testing performance while both data collected on MacBook and Linux operating system were processed. It achieved the lowest MSE, RMSE and R-squared scores on the testing data, indicating a higher level of accuracy in its predictions and a better fit to the data. In contrast, Linear Regression exhibited a lower level of accuracy in its predictions, with higher MSE and RMSE and R-squared scores on the testing data.

Overall, Random Forest Regression may be a better choice for predicting the outcome variable based on the given input variables. However, further investigation and evaluation may be necessary to determine the optimal model for a particular application.

V. CHALLENGES AND FUTURE WORK

The development of kernel modules poses various challenges that require careful consideration. Kernel module development presents a significant debugging challenge. Unlike regular user space applications, kernel modules lack access to detailed debugging information provided by the operating system, making it arduous for developers to identify and resolve issues that may arise during the module's development process. Developers need to rely on various strategies to identify errors in their code, such as kernel logs, print statements, and kernel debuggers. However, these methods can be time-consuming and inefficient, especially when dealing with complex kernel modules. Despite this challenge, the use of virtual machines, which is the only method applied in this work and make the work relatively easy, can help isolate and reproduce kernel module-related problems in a controlled environment, facilitating the debugging process. However, it is not efficient enough for developers to deal with the tedious process of debugging in this work and the potential issues may occur without being caught by operating system or developers.

Furthermore, not all process-related information is accessible on both operating system. Some inherent limitations exist within kernel modules to access information, resulting in incomplete data collection by the kernel module. In addition, built-in Python functions are not capable to collect data related to io counter and net counter for each process, which are critical factors for process performance.

In future work for topic of process performance, other models with potentially superior performance can also be explored. Additionally, greater attention can be paid to model validation. To validate a model, checking certain assumptions during data preparation is necessary, as is the case with Random Forest Regression, and it is recommended to perform model validation for this type of model as well. Furthermore, measures of variable importance can be implemented to select the useful variable to interpret the most fitted forest [1]. Besides, the time required to load the machine learning Python script is considerable and takes several minutes. Therefore, it is important to explore ways to shorten the execution time and improve the efficiency of the process.

VI. CONCLUSION

In conclusion, this research explored the use of Linear Regression and Random Forest Regression models to estimate process performance and behavior in Linux and macOS operating systems. The study showed that only Random Forest Regression can accurately predict process performance based on collected data from both operating systems and it outperformed the Linear Regression in terms of training and testing accuracy. These results demonstrate that not all machine learning algorithms are suitable for prediction and there is a potential of machine learning techniques in analyzing and understanding process performance in different operating systems. These predictions made by Random Forest Regression can provide valuable insights into how processes are performing, which predicting process performance can help achieve a more efficient system performance and enable programs to run efficiently, which is essential for optimal system functionality. The developed models provide a foundation for future research in this field and may lead to improved system optimization and performance.

REFERENCES

[1] A. Cutler, D. R. Cutler and J. R. Stevens, "Random Forests," Ensemble Machine Learning: Methods and Applications, pp. 157-176, January 2011

[2] F. Alenezi, and C. P. Tsokos, "Machine Learning Approach to Predict Computer Operating Systems Vulnerabilities," 2020 3rd International Conference on Computer Applications and Information Security (ICCAIS), pp. 1 – 6, March 2020

[3] G. A. F. Seber and Alan J. Lee, "Linear Regression Analysis", Wiley, Febr 2003

[4] I. U. Akgun, A. S. Aydin and E. Zadok, "KMLib: Towards Machine Learning For Operating Systems," On-device Intelligence Workshop MLSys 2020, Febr 2020

[5] N. Aslam, N. Sarwar and A. Batool, “Designing a Model for improving CPU Scheduling by using Machine Learning,” (IJCSIS) International Journal of Computer Science and Information Security, Vol. 14, No. 10, Oct 2016

[6] S. Fu, S. Gupta, R. Mittal and S. Ratnasamy, “On the Use of ML for Blackbox System Performance Prediction,” Symposium on Networked Systems Design and Implementation, Apr 2021