# Index terms: Machine learning, Process Performance, Operating System

# 0. Abstract

This work presents a study on using machine learning techniques to predict the performance of processes in operating systems. The machine learning-based approach uses various 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 models and random forests, are experimented with to identify the most suitable 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 local machine. The results shows that machine learning-based methods can accurately predict process performance. By implementing and evaluating the machine learning models, the findings demonstrate the potential of machine learning in improving the performance and efficiency of operating systems.

# 1. Introduction

The performance of processes in an operating system is a critical aspect that determines the overall efficiency and responsiveness of the system. Processes refer to the tasks or programs executed by the operating system, and their performance can significantly impact 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 can help achieve a more efficient system performance and enable programs to run efficiently.

Predicting process performance offers several benefits, including early detection of performance issues, improved resource utilization, better capacity planning, reduced downtime, improved user experience, and cost savings. To make such predictions, various process metrics, such as CPU utilization, memory usage, I/O throughput, and response time, need to be monitored. The system configuration can be adjusted as needed to optimize performance too.

On any operating system, users can check process information in the terminal using built-in command lines. This information usually includes the unique process ID for each task, the username of the task owner, task priority, total virtual memory used, the actual physical memory consumption, shared memory size in kilobytes, the state of the task, CPU and memory usage, CPU time, and the command being run. By leveraging vast amounts of data with multiple attributes, machine learning algorithms can identify patterns and make predictions about process performance.

In this work, application of predictive machine learning models to estimate the performance of processes is explored. This work demonstrate 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 tree 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 organized as follows. Section II provides an overview of the related work in predicting performance. Section III outlines the design of study, 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. Finally, in Section VI, the findings are summarized, and a conclusion is provided for this work.

# 2. Related Work

Machine learning algorithms have gained popularity in recent years as an approach to accurately predict the performance or behavior in different fields. Previous works have attempted to develop models for this purpose. For example, Fu et al. [345] applied machine learning to predict the performance of applications and use-cases, but the prediction error remained high for a couple of realistic scenarios. Asri et al. [234] used machine learning algorithms such as decision tree and support vector machines to predict the risk of cancers for diagnosis. Liu et al. [456] aimed at predicting manufacturing process performance and demonstrated that their proposed method outperformed the Elman Recurrent Neural Network (ERNN) in terms of mean prediction errors, which had previously been shown to perform well for highly non-linear and non-stationary time series.

The topic of process performance has been widely studied in the field of computer science. Aslam et al. [123] focused on improving CPU scheduling by applying Bayesian Decision Theory. Ardalani et al. [56] proposed a machine-learning-based tool, Cross-Architecture Performance Prediction (XAPP), that used single-threaded CPU implementation to predict GPU performance. XAPP leverages established machine learning techniques to learn the correlation between program properties, hardware characteristics, and GPU execution time, achieving 26.9% average error on a set of 24 real-world kernels and providing a highly desirable tool for estimating GPU performance before writing a GPU implementation.

While many studies have focused on predicting process performance in specific contexts, there is a need for more general models that can accurately predict process performance across a wide range of scenarios. The present study aims to address this gap by developing predictive machine learning models that can accurately estimate the performance of processes on both Linux and macOS operating systems, using data collected from a diverse range of users.

# 3. Design

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

\item 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.

\item 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.

\item 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 models and random forests.

\item 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.

## 3.1 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.

## 3.1.1 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.

## 3.1.1 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, a Python function is developed to gather data on the macOS operating system.

One advantage of building 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 build Python on a MacBook.

## 3.1.3 Data Preparation

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.

## 3.2 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 models and random forest regression. 4 different models were used, including linear regression, lasso regression, ridge regression, and random forest regression.

## 3.2.1 Linear Models

Linear models are a family of statistical models that assume a linear relationship between the predictor variables and the response variable. They are commonly used in regression analysis to predict the value of a continuous outcome variable based on one or more predictor variables.

Linear regression is a basic form of linear modeling that seeks to estimate the linear relationship between the predictor variables and the outcome variable. The goal of linear regression is to fit a linear equation to the data that best describes the relationship between the predictor variables and the response variable. The equation takes the form that [546794586745]:

*\begin{equation}*

*y = b\_0 + b\_1x\_1 + b\_2x\_2 + \cdots + b\_nx\_n + e*

*\end{equation}*

*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.*

Ridge regression is a variant of linear regression that uses L2 regularization to prevent overfitting. It adds a penalty term to the objective function of linear regression that penalizes large coefficient values. This penalty term shrinks the coefficients towards zero, thereby reducing the variance of the estimates. Ridge regression can be particularly useful when there are many predictor variables, and some of them are highly correlated.

Lasso regression is another variant of linear regression that uses L1 regularization to prevent overfitting. It adds a penalty term to the objective function of linear regression that shrinks some of the coefficients to exactly zero. This leads to a sparse model, where only a subset of the predictor variables is used in the model. Lasso regression can be particularly useful for feature selection and model simplification.

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

In summary, linear models, including linear regression, ridge regression, and lasso regression, are widely used due to their simplicity and interpretability, and can help identify the most important predictor variables, estimate the magnitude and direction of their effects, and avoid overfitting by regularizing the estimates.

## 3.2.2 Random Forest Regression (RFR) Models

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 the accuracy and reduce overfitting. The random forest algorithm works by creating a random sample of the training data and building decision trees on each sample. The random forest then combines the results of all decision trees to make predictions.

The basic idea behind random forest is to create multiple decision trees with different subsets of the training data and different features. Each decision tree is trained on a different subset of data and makes a prediction based on a different set of features. The results of all the decision trees are then combined to make a final prediction. The combination of multiple decision trees helps to reduce the variance and improve the accuracy of the predictions.

In a random forest regression model, the target variable is continuous, and the input variables can be continuous or categorical. The algorithm works by splitting the data based on the values of the input variables and fitting regression models to each subset of the data. The final prediction is the average of the predictions made by all models.

One of the advantages of random forest regression is that it can handle high-dimensional data and capture complex nonlinear relationships between the input variables and the target variable. It is also robust to outliers and missing data.

In this study, Random Forest Regression is applied by using the scikit-learn (sklearn) Python library as well.

Overall, random forest regression is a powerful machine learning algorithm that can be used for a wide range of regression tasks. It is especially useful when dealing with complex datasets.

## 3.2.3 Evaluation and Validation

*\begin{equation}*

*\boldsymbol{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}*

*\end{equation}*

*where $\boldsymbol{\beta}$ is the vector of regression coefficients, $\mathbf{X}$ is the design matrix, and $\mathbf{y}$ is the vector of response variables.*

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, Mean Square Error (MSE) is applied as an evaluation metric to train a model and minimize the cost function. The form of it is Equation [32548205]:

*\begin{equation}*

*MSE = \frac{1}{n} \sum\limits\_{i=1}^n (y\_i - \hat{y}\_i)^2*

*\end{equation}*

*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 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 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 MSE will be.

However, one of drawbacks of using only one testing set is that the test 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 which can fit a model several times using a different training and testing set each time and then calculate the test MSE to be the average of all the test MSE’s. 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.

*\begin{equation}*

*Test MSE = \frac{1}{k} \sum\_{i=1}^{k} \text{MSE}\_i*

*\end{equation}*

*where $k$ is Number of folds and $\text{MSE}\_i$ is Test MSE on the $i$th iteration*

Besides, R-squared is applied as an evaluation metric as well. The form of it is Equation [9234780234]:

*\begin{equation}*

*R^2 = 1 - \frac{SS\_{res}}{SS\_{tot}}*

*\end{equation}*

*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.

Model validation for linear regression model can be taken into account. Multicollinearity occurs 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. The variance inflation factor (VIF) is given by:

*\begin{equation}*

*VIF\_i = \frac{1}{1-R\_i^2}*

*\end{equation}*

*where $R\_i^2$ is the coefficient of determination of the regression model that regresses the $i$th predictor variable on all other predictor variables. The VIF measures the degree to which the variance of the estimated regression coefficient of the $i$th predictor variable is inflated due to collinearity with other predictor variables.*

If VIF >1 & VIF <5 moderate correlation, VIF < 5 critical level of multicollinearity.

# 4. Evaluation

## 4.1 Experiment Setup

The experimental setup is shown in this section. It involves collecting the data, preprocessing the dataset, implementing predictive models, evaluating their performance using multiple metrics and cross-validation, and measuring and comparing their computational efficiency.

## 4.1.1 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. 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.

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 long as the Linux environment is active, the virtual file is consistently updated by the kernel module to log process-related data. The file can grow in size dramatically in a very short time. However, due to its attribute as a virtual file, it does 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 operates while the system is on. If the file is open for exporting, the read never reaches the end, and the program for data export never terminates.

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. The Python code 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.

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.

## 4.1.2 Data Preparation and Input Values

In this experiment, the dataset was prepared for machine learning models by performing several preprocessing steps.

For the Linux operating system, data preprocessing was done during the export process to ensure the compatibility of the exported file with common data analysis tools. Each row was carefully checked and all white spaces were replaced by commas, which matches the format of a CSV file. This step eliminates the need for an additional conversion process from TXT to CSV format.

On the other hand, the data collection process on macOS operating systems was streamlined using the Python library csv. This library allows the header of a CSV file to be written and the records of a Python dictionary to be appended directly into rows. Therefore, no additional modification was done to prepare the dataset in CSV format.

To further prepare the dataset for machine learning models, the dataset was then checked for missing data and unnecessary data were dropped such as the same data 0 in the entire single column. Next, categorical data was converted to numerical data using label encoding to avoid working with categorical data directly because the number of categories was quite large and one-hot encoding led to high memory consumption during the process of trying different approaches. Additionally, the dataset was ready to be split into training and testing sets using a 75:25 ratio with 42 random seeds which is one of the popular integer numbers for random seeds. Setting up random seed could enable other developers who got the source code and same dataset to generate the exact same output when re-running the output. The training set was used to train the predictive models, while the testing set was used to evaluate their performance.

Overall, by performing these preprocessing steps, the dataset was of high quality and appropriate for use in machine learning models.

## 4.1.3 Predictive Models

In this study, the Python library sklearn was utilized to construct various machine learning models, including linear regression, ridge regression, lasso regression, and random tree regression. To evaluate the performance of each model, various metrics were applied including Mean Squared Error, R2 score, and cross-validation scores. For cross validation, the function cross\_val\_score was used to estimate model's accuracy in a more robust way than with just the typical train-test split. All the data were fed inside it, and the function made the necessary train-test splits, compared with using train\_test\_split.

Part of model validation for linear regression model was applied. Before building regression models, multicollinearity, which may reduce the precision of estimated coefficients and weaken the statistical power of regression models, was checked and some highly correlated independent variables were removed using Variance Inflation Factors (VIF). Variance Inflation Factors (VIF) were calculated by a check function to ensure the data was at less or moderate multicollinearity. Linear regression assumes that there is little or no Multicollinearity in the data.

The random tree regression model was specifically set with 1000 decision trees running in the model and 42 random states, and several parameters were adjusted to determine the criterion used to determine model outcomes, the maximum possible depth of each tree, and the maximum number of features the model considered when determining a split.

The model's consistency was further evaluated using Mean Squared Error, R2 score, and cross-validation scores. During cross-validation, RepeatedKFolder function from sklearn was utilized to determine the number of splits to use. This cross-validator repeats the K-Fold n times with different randomization in each repetition. The robustness of results was ensured by splitting the data into 5 folds and repeating the process 10 times, each with different random splits. This configuration could evaluate the performance of machine learning models on a limited sample of data in a more robust manner.

## 4.2. Experiment Result

Based on the results presented in the table, random forest model outperformed the other models in terms of both training and testing performance. It achieved the lowest MSE and RMSE scores on the testing data, indicating a higher level of accuracy in its predictions. Additionally, the random forest model had the highest R2 scores on both the training and testing data, indicating a better fit to the data. It gets 3.510039 MSE and 0.999908 R2 score.

On the other hand, the linear regression, Lasso regression, and Ridge regression models all showed a lower level of accuracy in their predictions, with higher MSE and RMSE scores on the testing data. The Lasso regression model, in particular, performed poorly with an accuracy of only 96.37% and a high standard deviation.

Overall, random forest model 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. Besides, other machine learning models can be applied in future work to seek a model with better performance.

# 5. Challenges

The development of kernel modules presents several challenges that should be taken into account. Although kernel modules offer a practical way to extend the functionalities of the kernel, it is important to recognize that there are inherent limitations to this approach. One of the main challenges of kernel module development is the difficulty of debugging. Unlike regular user space applications, kernel modules do not have access to detailed debugging information provided by the operating system, which makes it challenging for developers to identify and fix issues that arise during module development.

Debugging on kernel modules is a complex task that requires extensive knowledge of the kernel internals and advanced debugging techniques. Developers must 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. In addition, since kernel modules operate in a privileged mode, any mistake in their implementation can potentially cause severe system crashes or security vulnerabilities, making debugging even more critical.

Despite these challenges, several techniques and tools have been developed to ease the debugging process for kernel modules. For instance, the use of virtual machines, which is the method applied in this work, can help isolate and reproduce kernel module-related problems in a controlled environment, facilitating the debugging process.

In the future work, more work on model validation can be paid on attention. In order to validate model, checking few assumptions is necessary for linear regression model. The common assumptions for linear regression model are:

1. Linear Relationship: In linear regression the relationship between the dependent and independent variable to be linear. This can be checked by scatter plotting Actual value Vs Predicted value.

2. The residual error plot should be normally distributed.

3. The mean of residual error should be 0 or close to 0 as much as possible.

4. The linear regression require all variables to be multivariate normal. This assumption can best check with Q-Q plot.

6. Homoscedasticity: The data are homoscedastic meaning the residuals are equal across the regression line. We can look at residual Vs fitted value scatter plot. If heteroscedastic plot would exhibit a funnel shape pattern.

# 6. Conclusion

In this work, 2 different machine learning techniques, linear regression and random forest regression, are applied to learn and estimate process performance and behavior, based on data from Linux operating system and from macOS operating system. The data used during the experiment are collected from both Linux operating system and macOS operating system.

The results of the study show that the performance of processes in both operating systems can be accurately predicted using the collected data. Random forest model performs the best among all 4 models based on the training data and testing data. The development of these models provides a framework for further exploration and understanding of the performance of processes in different operating systems.

# 7. Reference

[7] Montgomery D. C., Peck E. A. and Vining G. C. Introduction to Linear Regression Analysis. Wiley, New York, NY, 2001.

[123][N. Aslam, N. Sarwar, A. Batool. Designing a Model for improving CPU Scheduling by using Machine Learning. In Proc. of (IJCSIS) International Journal of Computer Science and Information Security, Vol. 14, No. 10, Oct 2016](https://d1wqtxts1xzle7.cloudfront.net/51178809/25_Paper_30091654_IJCSIS_Camera_Ready_pp._201-204-libre.pdf?1483532507=&response-content-disposition=inline%3B+filename%3DDesigning_a_Model_for_Improving_CPU_Sche.pdf&Expires=1681967949&Signature=K1B8MJz86KVRkUnkCZ3Ex2Bu~IiK~wUZKw7K4umy6OfykU~pHPlmOKnWe5zRDxRp6V8BcOo3jBRpBzm5svb-uTE4qpO2kyvv91Qn2zfRAFvGQBW~gpxPyCMTrVywl40GjQOrSTNB3kEKm5ixK-PE9x6~Ur5Q8UwM159D9oCLJrYCnkaVDxg3Ee7pVny958vtUZpIuqCyc~AGi6~8WOusZRnLhwBmwNve8fs7GMtpwzqZTIe~0JYbYhM4bOkcdMGbwlMIUmHlmkV7ho6UUhNKFZn0XDRkIpabKPa-ZGyE-unyKJxxPWscII8Mo-lcg7VvoBqPQ9GHG4g92n3FtV5Agg__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA)

[345]S. Fu, S. Gupta, R. Mittal, S. Ratnasamy. On the Use of ML for Blackbox System Performance Prediction. In Proc. of the 18th USENIX Symposium on Networked Systems Design and Implementation, Apr 2021

[234] [Hiba Asri a, Hajar Mousannif b, Hassan Al Moatassime c, Thomas Noel d. Using Machine Learning Algorithms for Breast Cancer Risk Prediction and Diagnosis. The 6th International Symposium on Frontiers in Ambient and Mobile Systems. 12 May 2016.](https://www.sciencedirect.com/science/article/pii/S1877050916302575)

[456] [Jianbo Liu a, Dragan Djurdjanovic a, Jun Ni a, Nicolas Casoetto a, Jay Lee b. Similarity based method for manufacturing process performance prediction and diagnosis. Computers in Industry. Volume 58, Issue 6, August 2007, Pages 558-566. December 2006](https://www.sciencedirect.com/science/article/abs/pii/S0166361506001850)

[56] [Newsha Ardalani, Clint Lestourgeon, Karthikeyan Sankaralingam, Xiaojin Zhu. Cross-architecture performance prediction (XAPP) using CPU code to predict GPU performance. MICRO-48: Proceedings of the 48th International Symposium on MicroarchitectureDecember](https://dl.acm.org/doi/abs/10.1145/2830772.2830780)

[?]A. Maros, F. Murai, A. P. Couto da Silva, J. M. Almeida, M. Lattuada, E. Gianniti, M. Hosseini, and D. Ardagna. “Machine learning for performance prediction of spark cloud applications,” In 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), pages 99–106. IEEE, July 2019

by M. Abdollahzadeh, M. K. Akbari, and M. Sadeghizadeh, published in the International Journal of Computer Science and Information Security, Volume 14, Issue 1, January 2016, Pages 201-204.

[?]https://www.usenix.org/system/files/osdi18-lee.pdf

[?]https://ieeexplore.ieee.org/document/4085157

[?]https://kodu.ut.ee/~dumas/pubs/WhiteBoxJSEP.pdf

[?]https://www.fsl.cs.stonybrook.edu/~umit/files/kmllib-sysml-paper.pdf

[?]http://alumni.cs.ucr.edu/~kishore/papers/tencon.pdf

[?]https://www.researchgate.net/publication/220938947\_Efficiently\_exploring\_architectural\_design\_spaces\_via\_predictive\_modeling

[?]https://dl.acm.org/doi/abs/10.1145/325164.325119

[?]https://www.seas.upenn.edu/~leebcc/documents/lee2006-asplos.pdf

https://dev.to/arepp23/how-to-write-to-a-csv-file-in-c-1l5b

https://towardsdatascience.com/how-to-build-your-first-machine-learning-model-in-python-e70fd1907cdd

https://towardsdatascience.com/how-to-use-random-seeds-effectively-54a4cd855a79

https://www.analyticsvidhya.com/blog/2021/05/know-the-best-evaluation-metrics-for-your-regression-model/

https://www.linuxjournal.com/article/8110