PREPRINT

SCIENCE & TECHNOLOGY

Journal homepage: http://www.pertanika.upm.edu.my/

Random Forest Model for Software Build Time Prediction on CI/CD Pipeline

Wen Han Seow[1], Chia Yean Lim[1]* and Sau Loong Ang[2]

[1]School of Computer Sciences, Universiti Sains Malaysia, 11800, Minden, Pulau Pinang, Malaysia
[2]Department of Computing and Information Technology, Tunku Abdul Rahman University of Management and Technology, Penang Branch, 11200, Tanjung Bungah, Pulau Pinang, Malaysia

ABSTRACT

In the fast-paced world of software engineering, Continuous Integration/Continuous Delivery (CI/CD) pipelines are essential to deliver software builds continuously. However, the varying time taken for software builds to complete on these pipelines can challenge scheduling software delivery and impact productivity. To the best of researchers' knowledge, machine learning techniques have never been used to predict software build time in the CI/CD pipeline. This research attempted to apply data science and machine learning techniques, including linear regression (LR), support vector regressor (SVR), random forest regressor (RFR), and XGBoost regressor, to predict software build completion time to address this research gap. Past build events were used as a dataset to train and identify the best-performing model by evaluating the time a software build takes to complete. Different factors contributing to software build time on the CI/CD pipeline were also analyzed to identify opportunities for improvement. This research found that the random forest (RF) model achieved the best and outstanding performance of 14.306 in mean squared error (MSE). This model could be deployed to provide completion time estimates for software builds, enabling better code delivery scheduling. This research also suggested opportunities for improvement in the CI/CD pipeline by discovering major factors causing high build time in the CI/CD pipeline that engineers could rectify to reduce software build time in the CI/CD pipeline.

Keywords: CI/CD, machine learning, random forest, regression, software engineering

ARTICLE INFO

Article history:
Received: 30 October 2023
Accepted: 3 October 2024
Published: 21 February 2025

DOI: https://doi.org/10.47836/pjst.33.2.22

E-mail addresses:
seowwenhan@student.usm.my (Wen Han Seow)
cylim@usm.my (Chia Yean Lim)
angsl@tarc.edu.my (Sau Loong Ang)
*Corresponding author

INTRODUCTION

Continuous Integration/Continuous Delivery (CI/CD) is a method of frequently delivering applications or software by introducing automation into the stages of software development (Red Hat, 2023). In

ISSN: 0128-7680
e-ISSN: 2231-8526

short, new software codes submitted by the software engineers will go through a pipeline consisting of several stages of processes defined with steps to be tested and built, then proceed to deployment into the production environment. This method was widely used in software development companies of all sizes.

At this point in time, CI/CD is no longer a stranger to the software engineering field and has been widely adapted into the software development process across companies and industries. Benefits such as ensuring code quality, delivering codes faster with accelerated release rate, simplified rollback and cost reduction are among the top reasons companies adopt CI/CD (Silverthorne, 2022). In a study aimed at investigating how CI/CD adoption can impact open-source repositories hosted on GitLab and GitHub, it was found that adoption of CI/CD enhanced commit velocity by 141.19% in more than 12,000 repositories (Fairbanks et al., 2023).

A survey conducted by JetBrains showed the importance of CI/CD. 44% of the participants confirmed regular usage of CI/CD tools, and a significant 22% of them even adopted a new CI/CD tool within the past year (Bedrina, 2023). Since the usage of the CI word back in 1991, CI/CD has evolved from a relatively niche practice into an industry standard (Snyk, 2020).

In software development companies, the developer productivity (DevProd) team often aims to enhance the overall software development process and environment for all software engineers. Factors such as software tools and their configurations, development environments, cloud services like Amazon Web Services (AWS) and most importantly, the CI/CD framework could affect the software's build time in their own way (Amazon Web Services, 2023). With the size of an established technology company, more and more software engineers would join the company, and the number of software products would grow daily. There is no doubt a pressing need to improve the productivity of software engineers by streamlining their daily jobs.

There have always been issues in the availability and reliability of CI/CD service, and the unsatisfactory performance of software build time on the CI/CD pipeline has led to concern among the DevProd team and software engineers in the software development company. Software engineers in the company approach the DevProd team regularly for support and troubleshooting problems during their daily software engineering work. The DevProd team has also continuously made an effort to improve the quality of service of the CI/CD pipeline to benefit the company's software production efficiency. Hence, the continuous effort to find ways to further optimize the CI/CD quality of service is never-ending.

It was discovered that data science techniques can be used to eradicate the existing software build time issues in the CI/CD pipeline. In a multi-national software development company, the DevProd team has collected the data and logs of software build events for a long period. Information such as *time start*, *time end*, *region*, and *operating system* were collected and archived on data platforms for data-keeping purposes. With the dataset being stored and kept unused, the company now sees a chance to use it well. Data science techniques can be applied to the dataset for further analysis, achieving the goal of providing a better software development experience for the company. To the best knowledge of literature searching, no prior research was conducted on applying machine learning (ML) methods on CI/CD pipelines to predict software build time and analyze the major factors causing high software build time.

This research was conducted by applying several data science techniques to find out the causes behind the CI/CD pipeline's flaky service quality while utilizing the dataset to train a machine learning model that predicts the estimated time taken by a software build. Data analysis efforts were carried out to look for factors behind the unusually long software build time and availability issues of the CI/CD service. Supervised machine learning methods like regression can be applied to predict the estimated time the software build takes to complete. It not only saves software engineers time by giving them an expectation of time for their software build to complete in the CI/CD pipeline but also enables software engineers to plan their tasks better by hand. When it comes to urgent bug fixes and patches where deployments are expected to happen as soon as possible, the estimated build time will be helpful for the software team to anticipate the successful build to go live on production.

With the predicted output of the machine learning model on the estimated time taken of software build, anomaly build events that end prematurely or take extended time will be detected and then reported to the DevProd team. The team could immediately look after the CI/CD pipeline and respond when anomalies happen. A variety of reasons could lead to CI/CD pipeline issues and out-of-ordinary software build times. However, these issues could be investigated and rectified swiftly if identified instantly. The research hopes not only to benefit the DevProd team and the software company but also to discover and study the possibility of applying data science and machine learning techniques to the hard-to-reach areas of software build data on the CI/CD pipeline. It is crucial to software development productivity because it would greatly reduce engineering effort, money, and time.

Literature Review

Continuous Integration (CI) and Continuous Delivery (CD)

Continuous Integration (CI) is a software development practice where the software engineers merge new codes into the remote code repository (such as GitHub) regularly; the automated builds and tests are running in a usually cloud-hosted CI pipeline instead of the software engineer's local development environment (Amazon, 2023). CI saves the software engineer's effort and time by running the software builds and tests over the centralized pipeline, where the CI pipeline is usually the single source of truth on whether the build has passed the tests and is good to go for deployment into production.

Continuous Delivery (CD) is the process where new software codes are automatically built, tested and released into production continuously (Amazon, 2023). It is important to ensure the code changes are fit for going into the production environment with the help of a series of automated software tests. The stages of the CI/CD process are shown in Figure 1.

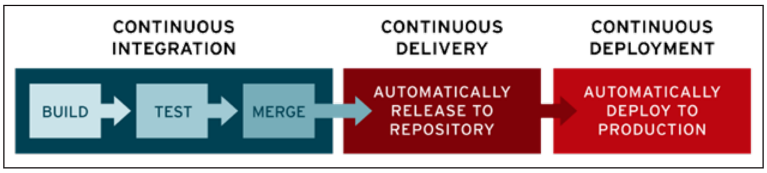


Figure 1. Stages of CI/CD Process (Red Hat, 2023)

When the DevProd team has already integrated the CI/CD pipeline by involving a wide range of AWS services and other Software as a Service (SaaS) products such as Buildkite, project development will be easier after understanding how the products interact with the CI/CD pipeline that has been serving the company's software engineers.

Contrary to what most believe, the build time of software impacts a software developer's productivity more than most people would imagine. By accurately providing the estimated build time, developers could have better control over their time planning and organizing their tasks based on the predicted build time of software on the CI/CD pipeline. For example, the developer may go for a lunch break when the build is predicted to take around an hour. When the build is expected to take 5 minutes, the developer may proceed to check the emails or go for a quick code review. The essence is to come up with a build time prediction that is accurate enough for developers to gain control of time and optimize the tasks in their hands (Jaspan & Green, 2023). Ultimately, it brings more efficiency into daily tasks by reducing time wasted waiting for the software build to be completed or time spent on miscellaneous tasks when the software build has already been completed.

Google conducted an in-house experiment to measure how the improvement of software build time affects developer satisfaction (Jaspan & Green, 2023). In the experiment group, developers were equipped with performant machines that outperformed the control group machines by a modest 15%. The results of the experiment have shown that developers managed to achieve higher self-reported productivity and higher self-reported velocity in their work. Developers in the experiment group also measured a greater satisfaction level than the control group. It is a concrete example of how faster build time could bring benefits to developers and the company.

CI/CD pipelines are not easy to troubleshoot. The time taken for software to build on a CI/CD pipeline could fluctuate due to many factors, including machine specification, agent availability, and CI/CD pipeline configurations. Engineers spend much time daily discovering the reasons that have caused inconsistency or even a spike in software build time on the CI/CD pipeline. The action taken to remedy the issue may not always be the best action to resolve the issue. It is challenging to quickly come up with the right solution to fix the slow-performing CI/CD pipeline. While there are no existing machine learning methods to help in the prediction of the CI/CD pipeline build time and discovering the factors affecting the build time, the effort to optimize the build time is always based on conventional and traditional methods such as expert knowledge, software engineer's best guessing, or finding similar solution from online sources. The current methods are illustrated in Table 1.

Table 1
Comparison between current methods of debugging CI/CD pipeline

Current Method	Methodology	Weakness
Expert knowledge	Experts will diagnose the issue based on experience and domain knowledge	Highly dependent on having an experienced expert with deep domain knowledge
Engineer's guessing	Engineers debug based on their guessing and experience	Guessing may be inconsistent and unreliable
Finding solutions from online Sources	Engineers look for possible solutions from online sources, such as forums, to fix the issue	It is time-consuming to look for solutions from different sources Unreliable and unverified answers can waste time

Memory Bottleneck in Compile-time

In a study by Cahoon (2002) on compile-time analysis based on the programming language Java, he found that memory hierarchy in modern architecture is among the major performance bottlenecks in compile-time. However, when the software build and software compilation happens on the CI/CD pipeline rather than the software engineer's local development machine, the local development machine's performance does not matter anymore. It is acceptable for the software engineers to have a relatively weaker machine since the software is expected to be built and pass the automated tests on the remote CI/CD pipeline. The CI/CD pipeline is the centralized location and the single source of truth to ensure the software build is fit for going into production. This means that improving the performance of the CI/CD pipeline will reduce the software build time for the software engineers, triggering a build in the pipeline. It shows that investing effort in optimizing the CI/CD pipeline is worthwhile.

Machine Learning Techniques for CI/CD Prediction

This research explored various machine learning (ML) techniques such as random forest (RF), XGBoost (XGB), gradient boosting (GB), k-nearest neighbor (KNN), support vector vegressor (SVR), which the characteristics and performance of the techniques are believed to be suitable in the CI/CD prediction's context. Kaliappan et al. (2021) have compared the performance of the above ML models in their study by evaluating them on the prediction of the COVID-19 reproduction rate. The research used common evaluation metrics such as mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), determination coefficient (R^2), relative absolute error (RAE), and root relative squared error (RRSE) to benchmark the performance of the ML model in regression task. The comparison result is shown in Table 2.

Table 2
Performance comparison between ML models in a Regression task

Performance Metrics	RF	XGB	GB	KNN	SVR
MAE	0.020	**0.019**	0.021	0.019	0.077
MSE	**0.001**	0.002	0.002	0.002	0.007
RMSE	**0.038**	0.041	0.041	0.041	0.085
R-Squared	**0.976**	0.973	0.973	0.973	0.884
RAE	0.106	**0.102**	0.112	0.102	0.404
RRSE	**0.154**	0.166	0.164	0.165	0.341

*Source: Kaliappan et al. (2021)

As shown in Table 2, RF and XGBoost performed very well against the other ML models used in the study. RF achieved the best MSE, RMSE, R-Squared and RRSE, while XGBoost achieved the best MAE and RAE. RF and XGBoost are very popular ML models used in machine learning tasks. They are also ensemble ML methods that consider multiple ML models to produce a final prediction output that is generally more accurate than a single ML model.

Based on the analysis of Kaliappan et al.'s (2021) findings, RF and XGBoost are believed to be the ideal models to be used in the study because of their strong performance. Meanwhile, this study would also adopt other simple regression models, such as linear regression (LR) and support vector regression (SVR), to benchmark the performance with the two suggested ML models.

Related Work on Applying Machine Learning to DevOps Processes

The research conducted by Battina (2021) was identical to this research because it applied machine learning techniques to optimize the DevOps processes. Several machine learning techniques were studied to apply them to DevOps processes for better software quality. The author stated that dataset inputs, which are outputs from Git and Jenkins, can provide insight into the software delivery process. The anomalies of large code volumes and long build times in the data can be identified using machine learning models. The said study has a certain level of similarity to this research in the sense of performing prediction on software build time with a dataset of build event logs. This research brings confidence and proves the feasibility of delivering a solution that detects anomaly software build events in a CI/CD pipeline by comparing the instance with the prediction.

In another similar research, the authors proposed applying machine learning techniques to find defects in the CI/CD pipeline (Lazzarinetti et al., 2021). Attributes from software code version control include *commits* (author, commit message, date), *changes in commit* (added lines, deleted lines), and *sonar measures* (line of codes in the commit), which were used to identify the defects in the CI/CD pipeline. Albeit the slight difference in research methodology where the said research measures the software code version control attributes while this research uses attributes of the software build event on CI/CD pipeline, it shows there were prior efforts on experimenting with machine learning techniques on CI/CD pipelines with the same goal to improve the productivity of software engineers.

Related Work on Predicting Continuous Integration Build Failures with Evolutionary Search

Saidani et al. (2020) have conducted research to predict build failures on continuous integration by using evolutionary search. The research used a novel search-based approach based on multi-objective genetic programming (MOGP) instead of machine learning or deep learning. A model was built to predict whether the CI build will succeed or fail. The research was conducted with a benchmark of 56,019 builds from 10 large-scale software projects running on the Travis CI build system (Saidani et al., 2020). As a result, the author found that the method mentioned achieved a statistically better result than the models developed with machine learning techniques.

The main differences between the research conducted by Saidani et al. (2020) and this research are the features used and the prediction outcome. In the former, the researcher was trained on attributes specific to the files and codes changed in the commit, such as *change size, file changes,* and *committer experience*. In this research, the features used were the CI/CD pipeline attributes as well as the time when the commit is pushed to the pipeline. The target of the prediction is different as well. The former tries to predict whether the CI build will pass or fail, while this research predicts the time taken for the software build to complete in the CI/CD pipeline.

Literature Review Discussion

At the time of writing, no research papers and reports were found on the Internet or in research databases with the exact same objective: to predict the software build time on the CI/CD pipeline with machine learning. This research is considered relatively unique because it lacks other research to be benchmarked and compared against.

It is firmly believed that this research will be impactful and stir inspiration in the software development industry. Software engineers and technical leaders could benefit from their daily work of software development by applying the ML model to predict software build time on CI/CD pipeline and perform data analysis to root cause CI/CD pipeline issues, ultimately bringing better productivity in software development.

Without other research to compare with, this research should be conducted with a proper methodology and evaluated with correct judgment based on the research context and business requirements.

MATERIALS AND METHODS

Research Framework

Figure 2 shows the components of the research framework conducted in this research. The software development process could be illustrated in four areas: (1) development environment, (2) CI/CD Pipeline, (3) CI/CD prediction, and (4) production environment. The four areas are described in Table 3.

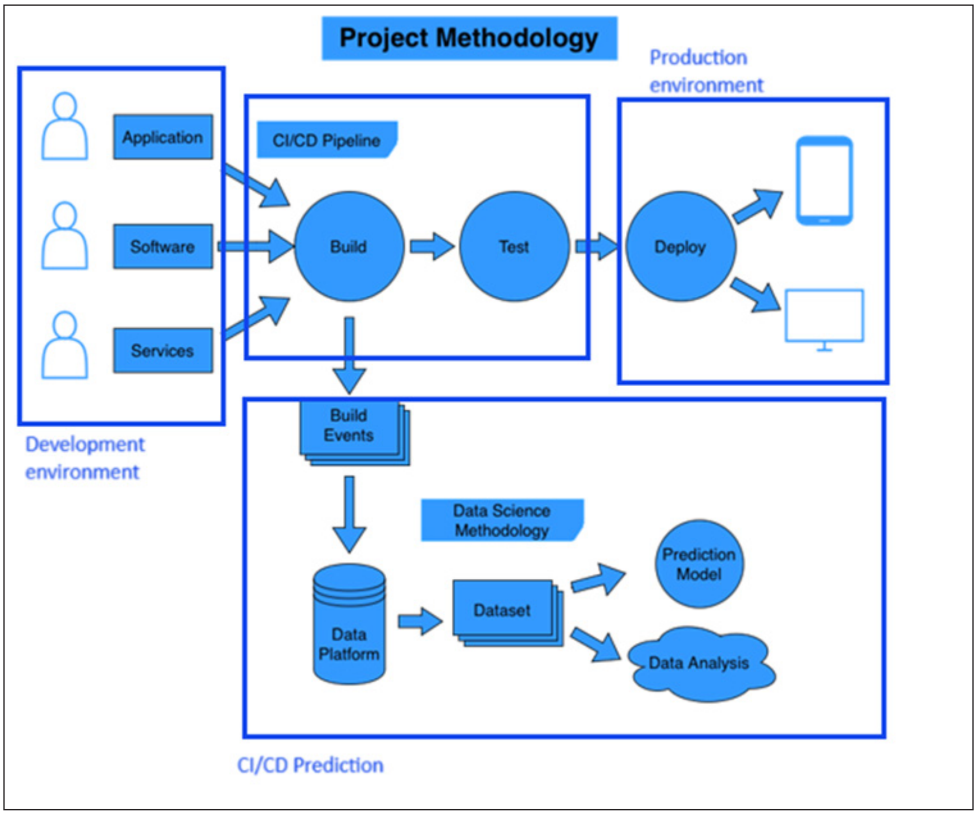


Figure 2. Research framework

Table 3
Description of research framework's components

Component	Description
Development environment	It is the environment where software engineers build applications and software codes locally. The finished codes will be pushed to the remote repository and version control system for further perusal.
CI/CD pipeline	It tracks the remote repositories and checks the status of new codes constantly "pushed." Auto build process will take place whenever new "commits" are available on the remote repository. Unit tests or integration tests come next after a successful build.
CI/CD prediction	The collected build event dataset is used to conduct software build time prediction using data science techniques.
Production environment	It is the environment where the successfully built and tested codes are deployed. New changes for the webpage can be rolled out immediately to the live environment. New web API services can be published to the production environment easily and efficiently with minimal to no server downtime.

Experiment Setting and Machine

Model selection and training were also solely conducted on the local machine, which performed very promisingly. The machine used is an Apple MacBook Pro 16 equipped with an Apple M1 Pro as the central processing unit (CPU) and 32GB of random access memory (RAM).

The biggest training dataset used in this project is around 800MB in CSV file form. The historical data was collected from a multi-national software development company. With the dataset loaded and preprocessed into Jupyter Notebook for model training, the machine handled the training without any hint of stress and completed the training within a few minutes. Considering the scale of the project's dataset, it is an acceptable performance for this project.

Research Methodology

This research adopted the cross-industry standard process for data mining (CRISP-DM), an industry-proven data science methodology, to organize data science projects. The sequence of the phases (Figure 3) may not be strict, and most projects may move back and forth between them when deemed necessary (IBM, 2021).

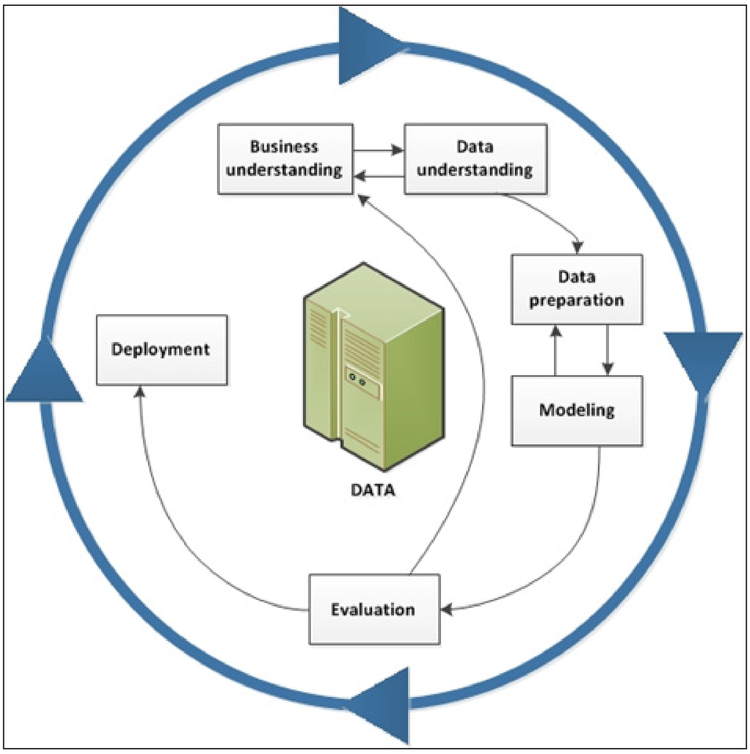


Figure 3. CRISP-DM methodology (IBM, 2021)

The first step in this research was to understand the business requirements and the available dataset. The Research acknowledges that the company's DevProd team was encountering inconsistent software build time on the CI/CD pipeline and would like to resolve this issue. The research has proposed adopting ML models, training the ML models with the build event dataset to predict CI/CD pipeline software build time, and further data analysis of the root causes of high software build time.

Next, the dataset needs to be prepared before the modeling step. Several steps, such as feature selection, feature encoding, and feature engineering, were performed to train the ML model.

With several ML models in mind, this research evaluated the ML models' performance on this preprocessed dataset. Mean squared error (MSE) was chosen as the evaluation metric because it is a common metric used to benchmark ML models in regression tasks. It averaged out the positive and negative differences by squaring the values. According to Hodson et al. (2021), mean squared error (MSE) is an ideal performance benchmark because of its link to the concept of cross-entropy from information theory. An ML model with a low MSE score is considered a good ML model, whereas a perfect ML model would have a perfect score of 0 in MSE.

After picking the best ML model, it will be tested with deployment. Depending on the scenario, the ML model might be tested on a test project to experiment with its performance and results. Further adjustments to the ML model and project setup can be made at this point.

Steps to Conduct Build Time Prediction with Data Science Techniques

Step 1: Feature Selection

Feature selection is the process of choosing the best features for the model under investigation (Rosidi, 2023). According to Rosidi (2023), correlated features that would cause multicollinearity and high computational power caused by too many features in a dataset were among the reasons why feature selection is crucial to a machine learning model.

The original dataset is a CSV file generated by a Python script specifically written to preprocess the dataset file format exported by Splunk. The dataset has 79 columns. The required features are selected based on the domain expert's knowledge of the dataset. For example, it is logical to remove the feature *git_commit_id* since a Git commit ID that looks like "070dcfee" will not make any sense to the machine learning model since it is generated without bringing any meaning or significance other than uniquely identifying the specific Git commit. Many other features in the original dataset carrying unmeaningful values, such as randomized character strings, null values, and irrelevant values, were also removed because they would not provide useful information for model training. Finally, 13 columns (features) were selected as the prediction variables.

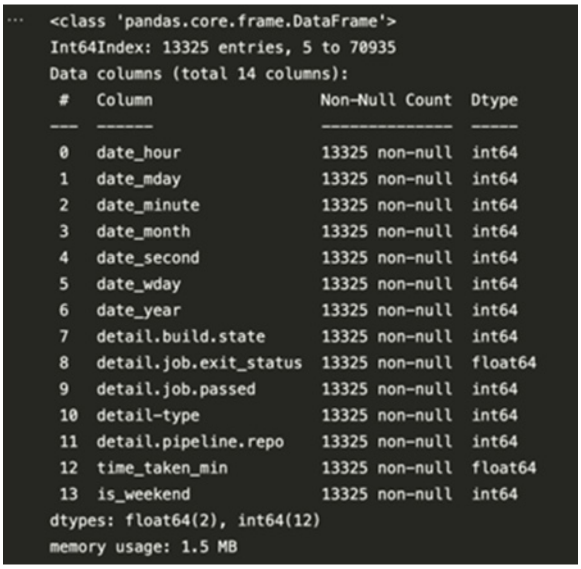
Step 2: Feature Encoding.

Feature encoding is the step to transform features with string values, usually into numerical values comprehensible by machine learning models. It can be done manually by providing a mapping of values in the form of a data structure or by using data science libraries to automate the process when it comes to features with a high number of distinct values.

Several feature encoding methods were used to transform the variables into the right format for the prediction process. A simple encoding process is adapted to store the mapping value for month and day, such as by declaring a Python Dictionary variable. In the case of features with a large number of unique values, label encoding was used to transform the relevant feature from categorical string labels into number labels by using the scikit-learn library method called *LabelEncoder*.

Step 3: Feature Engineering.

The feature engineering technique creates useful features not in the original dataset and potentially improves the machine learning model's performance. In this case, the variable *time_taken_min* is created by calculating the time difference between existing features *end_time* and *start_time* to be this machine learning project's target variable (output). The new variable generated with feature engineering is added to the existing 13 variables chosen from the feature selection step to form 14 columns (features), as shown in Figure 4, to be used as the prediction model's variables.



```
...  <class 'pandas.core.frame.DataFrame'>
     Int64Index: 13325 entries, 5 to 70935
     Data columns (total 14 columns):
      #   Column                 Non-Null Count  Dtype
     ---  ------                 --------------  -----
      0   date_hour              13325 non-null  int64
      1   date_mday              13325 non-null  int64
      2   date_minute            13325 non-null  int64
      3   date_month             13325 non-null  int64
      4   date_second            13325 non-null  int64
      5   date_wday              13325 non-null  int64
      6   date_year              13325 non-null  int64
      7   detail.build.state     13325 non-null  int64
      8   detail.job.exit_status 13325 non-null  float64
      9   detail.job.passed      13325 non-null  int64
      10  detail-type            13325 non-null  int64
      11  detail.pipeline.repo   13325 non-null  int64
      12  time_taken_min         13325 non-null  float64
      13  is_weekend             13325 non-null  int64
     dtypes: float64(2), int64(12)
     memory usage: 1.5 MB
```

Figure 4. A snippet of dataset columns after feature selection and engineering steps

Step 4: Identifying Relevant Machine Learning Models

According to the famous no-free-lunch theorem (NFL), the researchers cannot formally ground their conviction that some machine learning models are more sensible than others (Sterkenburg & Grünwald, 2021). Machine learning models have varying behaviors in different datasets, and the performance of the models is not guaranteed to be the same in all situations. Hence, there is a need to test the waters by trying out several machine-learning models to find the most suitable model for this dataset.

The project first conducted experiments with simpler machine learning models such as linear regression (LR and support vector regression (SVR to get a glimpse into how these models would perform against the baseline benchmark. The experiments' results were unsatisfactory compared to the baseline benchmark.

Therefore, more complex machine learning models, such as RF, multilayer perceptron regressor and XGBoost, were added to the experiment list to see if the complex model would achieve better performance than the simpler model. The results of each machine-learning experiment are shown in Table 4.

Table 4
Result of experiments with different machine learning models

Model Name	MSE
Baseline-dummy	299.36
Linear regression	293.50
Support vector regressor	308.49
Random forest	260.73
Multilayer perceptron regressor	278.91
XGBoost regressor	275.76

Table 4 shows that the more complex machine learning models, such as XGBoost regressor and multilayer perceptron regressor, have performed better than the baseline benchmark, linear regression, and support vector regressor by achieving lower MSE scores. However, they are less outstanding in comparison to the RF model because the model has achieved the best result with the lowest MSE score out of all models. As such, the RF model would be used to perform the software build time prediction.

RESULTS AND DISCUSSIONS

Experiment Set Up

The experiment was conducted with two datasets with different instances (Table 5) to explore whether the number of instances affects the prediction model's performance. Both datasets underwent the same data preprocessing methods and were trained using the same machine learning, the random forest (RF) model. For a small dataset, 70% of the data was used for training, while the other 30% was used for testing. The dataset is retrieved from Splunk with 1 week's worth of data for the build event dataset. As for the big dataset, 5-fold cross-validation was used to validate the model. The dataset was retrieved from AWS CloudWatch, which had 1 month's worth of data for the build event dataset. The number of features used in the big dataset was reduced to 8 from 14 in the small dataset by removing irrelevant features that scored zero in feature importance, deduced from the RF model trained on the small dataset.

Table 5
Comparison of small and large datasets

Dataset	Small Dataset	Big Dataset
Size of data frame	2394	44344
Number of features	14	8
Number of instances	171	5543

The prediction of software build time on the CI/CD pipeline was made using the trained RF model to perform prediction on a new data instance.

For example, the trained RF model will give a new data instance with the same set of features. Since the RF model was trained on the dataset with the same set of features, it will be able to provide a prediction for the new data instance after "learning" from the training dataset.

Table 6 shows the result of the software build prediction with an RF model with small and big datasets. The RF model's performance in the smaller dataset is highly assuring and outperformed by the dummy regressor, which is used as the baseline for comparison in this experiment. On the other hand, the RF model with a big dataset did not outperform the baseline dummy regressor too much (less than 1.0) as previously expected.

Table 6
Prediction result comparison for small and big datasets

Dataset	**Model**	**MSE**	**RMSE**
Small dataset	Dummy regressor	53.232	7.296
	RF model	14.306	3.782
Big dataset	Dummy regressor	19.008	4.360
	RF model	16.642	4.079

Discussion on Result

The experiment showed that the RF model performed better when using a small dataset than a big dataset. There could be several possible reasons for the not-so-good performance of a big dataset. Firstly, the dataset might have a high variance. Secondly, the dataset might be imbalanced. Thirdly, more features could be required for the big dataset to produce more accurate prediction output. Fourthly, the repository that was used might have glitches in the build time in the dataset.

Considering the outstanding performance of the RF model over the dummy regressor in the small dataset, it is believed that the result of the RF model that was trained in this research was significant. With the RMSE score of 3.782 achieved in the small dataset, where the feature unit is in minutes, the model's prediction is only 3.782 minutes away from the actual result on average. In a dataset with software build, events take 20 minutes to 1 hour. The RMSE score of 3.782 achieved by the RF model is considered outstanding in this business context.

While it is fine to conduct software build time prediction with a small dataset at a time by using a dataset with build data for a shorter period, with the latest trend analysis, the RF model is the ideal machine learning model to be used on this software build dataset for software build time prediction.

Direct comparison of this research with other existing research is unavailable because this research is unique as far as the researchers' best effort of literature search. However, it is important to note that the result achieved by the RF model is reassuring and gives confidence in the technique of applying ML models to predict CI/CD pipeline software build time and provide further analysis using the model trained to give better suggestions on root cause leading to slow CI/CD pipeline performance, in comparison to using the traditional and unstructured methods as discussed in Table 1.

Impact of Research in the Real World

The RF model has been deployed in a small-scale project's CI/CD pipeline for testing and observing its performance. After deploying the model on the testing project's CI/CD pipeline, the team has observed a similar performance of MSE score with the RF model trained. The RF model is able to predict software build time on the CI/CD pipeline, which is relatively close to the actual software build time on the pipeline.

Due to the limited scope of the test environment and dataset involved, the major factor that affects the software build time on the CI/CD pipeline the most is the start time. It seems that in this environment and scope, the start time, especially the hour it started, will impact the most on how much time it will take to finish the software build on the CI/CD pipeline.

This is explainable because although software building might happen frequently within a certain hour, the busy CI/CD pipeline might not have enough agents and hardware resources to perform the software build requests in a timely manner. Ultimately, this leads to a longer waiting time and, thus, a longer software building time.

It is hoped that this research will inspire software engineers, CI/CD engineers and technical leaders to adopt ML techniques into their infrastructure and technology stack. This will improve software development productivity by providing accurate software build completion time. It will also decrease CI/CD pipeline downtime coupled with performance improvement with the help of data analysis and machine learning techniques to debug issues.

Adopting the technique proposed in this research will save time and provide a higher accuracy of troubleshooting on the CI/CD pipeline. These will all lead to a lesser downtime of CI/CD pipelines with higher performance. Engineers will also better understand the factors leading to depriving performance of the CI/CD pipeline. The monetary costs of the computing infrastructures will be saved for the above-mentioned reasons. Faster software shipping speed thanks to the efficient CI/CD pipeline enables speedy business software delivery. A healthy and efficient CI/CD pipeline would improve the recovery crisis of website downtime.

CONCLUSION

This research explored the machine learning model to predict software build time with various sizes of datasets. Five machine learning models, linear regression, support vector regressor, random forest, multilayer perceptron regressor, and XGBoost regressor, were experimented with to find the best prediction model to predict software build time with the selected 14 features. The experiment showed that the RF model with a small dataset in this research experiment is the best model for software build time prediction.

Without prior research on this specific subject matter, this research has uniquely provided an approach to adopting data science and ML techniques in the CI/CD pipeline to predict the software build time-based on a dataset of past software build events. Data analysis could be done with the help of the trained model to further analyze and debug the performance issues in the CI/CD pipeline.

In an ideal world, developers would not only wish to reduce their software build time optimally but also get an accurate prediction of their software build time. It enables the developers to perform at a higher level of productivity and brings more satisfaction to their daily jobs. This brings us to the goal of this research: discovering the feasibility of applying machine learning and data analysis to help predict the software build time, as well as discovering major factors that could lead to better-optimized software build time.

FUTURE WORK

The first possible improvement that came to mind was introducing more features to the dataset. The project dataset did not have several good candidates for features such as agent (a virtual agent that runs jobs), waiting time, and instance startup time (time for virtual machines to boot into usable state). They would certainly bring more information and insight into the study and help better analyze how other factors could also contribute to a higher build time. By taking more meaningful dataset features into account, bottlenecks can be identified to help bring improvement for a faster build time. The machine learning model also gains robustness with more relevant features in the training dataset.

Another future work that would take the project idea to a further step is to train a generalized machine learning model. The RF model was trained in very specific variables in the current project. Only one single code repository and specific CI/CD pipeline configuration were included in this project dataset. The model is expected to not perform well universally outside the specific features or variables it was trained on. The ideal machine learning model should have the ability to cater to changes, such as predicting across different code repositories and adding extra steps into the CI/CD pipeline. These are indeed ambitious targets for the project and require an immense amount of expert knowledge with a deep understanding of the domain. An enormous amount of data collection, analysis and research into the CI/CD pipeline is required to deliver this seemingly bold yet highly rewarding improvement.

ACKNOWLEDGEMENT

The authors thank Universiti Sains Malaysia for funding this paper's publication.

REFERENCES

Amazon Web Services. (2023). *Practicing continuous integration and continuous delivery on AWS*. AWS. https://docs.aws.amazon.com/pdfs/whitepapers/latest/practicing-continuous-integration-continuous-delivery/practicing-continuous-integration-continuous-delivery.pdf

Battina, D. S. (2021). The challenges and mitigation strategies of using devops during software development. *International Journal of Creative Research Thoughts (IJCRT), 9*(1), 4760–4765.

Bedrina, O. (2023, August 7). *Best continuous integration tools for 2023 – Survey results*. JetBrains Blog. https://blog.jetbrains.com/teamcity/2023/07/best-ci-tools/

Cahoon, B. D. (2002). *Effective compile-time analysis for data prefetching in Java* [Doctoral dissertation, University of Massachusetts Amherst]. University of Massachusetts Amherst. https://www.cs.utexas.edu/users/mckinley/papers/cahoon-thesis.pdf

Fairbanks, J., Tharigonda, A., & Eisty, N. U. (2023, May 23-25). *Analyzing the effects of CI/CD on open source repositories in github and gitlab.* [Paper presentation]. IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA), Orlando, Florida. https://doi.org/10.1109/sera57763.2023.10197778

Hodson, T. O., Over, T. M., & Foks, S. S. (2021). Mean squared error, deconstructed. *Journal of Advances in Modeling Earth Systems*, *13*(12), Article e2021MS002681. https://doi.org/10.1029/2021ms002681

IBM. (2021, August 17). *CRISP-DM help overview*. IBM. https://www.ibm.com/docs/en/spss-modeler/saas?topic=dm-crisp-help-overview

Jaspan, C., & Green, C. (2023). developer productivity for humans, part 4: Build latency, predictability, and developer productivity. *IEEE Software*, *40*(4), 25–29. https://doi.org/10.1109/ms.2023.3275268

Kaliappan, J., Srinivasan, K., Mian Qaisar, S., Sundararajan, K., Chang, C. Y., & C, S. (2021). Performance evaluation of regression models for the prediction of the COVID-19 reproduction rate. *Frontiers in Public Health*, *9,* Article 729795. https://doi.org/10.3389/fpubh.2021.729795

Lazzarinetti, G., Massarenti, N., Sgrò, F., & Salafia, A. (2021, November 30). *A machine learning based framework for continuous defect prediction in CI/CD pipelines*. [Paper presentation]. Proceedings of the Italian Workshop on Artificial Intelligence and Applications for Business and Industries (AIABI), Milan, Italy.

Red Hat. (2023, December 12). *What is CI/CD?*. Red Hat. https://www.redhat.com/en/topics/devops/what-is-ci-cd

Rosidi, N. (2023, June 6). *Advanced feature selection techniques for machine learning models*. KDnuggets. https://www.kdnuggets.com/2023/06/advanced-feature-selection-techniques-machine-learning-models.html

Saidani, I., Ouni, A., Chouchen, M., & Mkaouer, M. W. (2020). Predicting continuous integration build failures using evolutionary search. *Information and Software Technology*, *128*, Article 106392. https://doi.org/10.1016/j.infsof.2020.106392

Silverthorne, V. (2022, February 15). *10 Reasons why your business needs CI/CD*. GitLab. https://about.gitlab.com/blog/2022/02/15/ten-reasons-why-your-business-needs-ci-cd/

Snyk. (2020, October 1). *What is CI/CD? CI/CD Pipeline and Tools Explained*. Snyk. https://snyk.io/learn/what-is-ci-cd-pipeline-and-tools-explained/

Sterkenburg, T. F., & Grünwald, P. D. (2021). The no-free-lunch theorems of supervised learning. *Synthese*, *199*(3), 9979–10015. https://doi.org/10.1007/s11229-021-03233-1