# ML Based Approach to Estimate the CPU-Burst Time of Processes – A survey and Implementation

Kishan Trivedi
Computer Science and Engineering
Santa Clara University
Santa Clara, United States of America
khtrivedi@scu.edu

Maitreyee Gadwe
Computer Science and Engineering
Santa Clara University
Santa Clara, United States of America
mgadwe@scu.edu

Gururaj Suresh Patil
Computer Science and Engineering
Santa Clara University
Santa Clara, United States of America
gpatil@scu.edu

***Abstract*— The Prediction of length of a CPU burst is crucial to scheduling algorithms like SJF and SRTF. Conventional methods for predicting CPU burst time may not provide accurate or dependable results. We worked on an ML-based approach to estimate the CPU-Burst Time of processes. To complete the task, we used feature selection approaches to determine the most important process features, and then forecast the CPU-burst for the process. To test and assess the approach, LightGBM is used in conjunction with Gradient Boosted Decision Trees, LightGBM Random Forests, and LightGBM Goss. In comparison to earlier research, the experimental results revealed greater accuracy with Random Forest as weak learner showing maximum results. In addition, the use of attribute selection approaches enhanced performance in terms of time and estimation. We used "GWA-T-1 DAS2" dataset for model training.**

***Keywords—Machine Learning, CPU, process scheduling, LightGBM***

## I. Introduction

Hundreds of processes run in an Operating System (OS). However, in a single processor system only one process at a time can use CPU for its execution. In order for multiprogramming to work, it is necessary for it to decide which process has to be scheduled first for its execution on the CPU. This task is performed by a scheduler and is called as "Process Scheduling". The task of allocation is done using various scheduling algorithms that makes decision based on different factors that defines the priority of a process. For example, algorithm like Shortest Job First (SJF) and Shortest Remaining Time First (SRTF) uses the amount of time that a process would require to complete its execution to decide which process will use the CPU first. The linear approaches used in the past were neither reliable nor accurate for the required prediction. In order to overcome the limitations of this approaches, Machine learning came into picture to make accurate predictions. Multiple approaches have been made using different algorithms to predict the CPU burst time. Our goal in this paper is to demonstrate the working of LightGBM algorithm to predict CPU burst time for a set of processes.

### A. *LightGBM*

In contrast to other boosting algorithms that develop trees leve l-by-level, LightGBM splits the tree leaf wise. To grow, it chooses the leaf with the greatest delta loss. The leaf-wise algorithm has a lower loss than the level-wise algorithm since the leaf is fixed.

### B. *Weak Learner (Random Forest and Decision Tree)*

Colloquially, a model that performs slightly better than a naive model. More formally, the notion has been generalized to multi-class classification and has a different meaning beyond better than 50 percent accuracy.

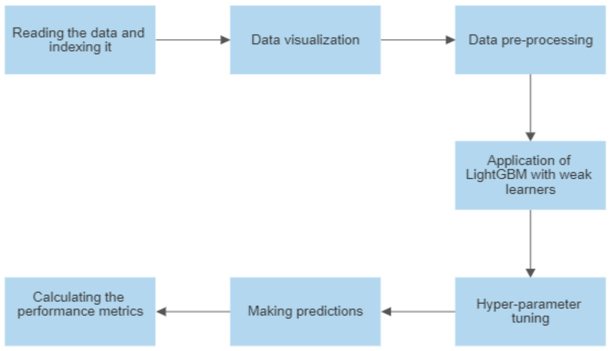
Random Forest is a Supervised Machine Learning Algorithm that is commonly used to solve classification and regression problems. It constructs decision trees from various samples and uses the majority vote for classification and the average for regression. One of the most essential characteristics of the Random Forest Algorithm is that it can handle data sets with both continuous and categorical variables, as in regression and classification, respectively. It performs better results for classification problems.

A decision tree is a supervised machine learning algorithm used for classification. In a
decision tree, each internal node represents an attribute or feature and leaf nodes represent
classes. The past from node to leaf denotes classification rules. The algorithm uses
Information Gain value as metrics to construct the decision tree. Attributes having higher
values of Information Gain are placed closer to the root.

$$\text{Entropy} = \sum_{i} -p \log_2 p$$

Information gain = Entropy(parent) - [weighted average] Entropy(children)

## II. Design Methodology and Space Exploration


Reading the data and indexing it
Data visualization
Data pre-processing
Application of LightGBM with weak learners
Hyper-parameter tuning
Making predictions
Calculating the performance metrics
.

### A. *Steps involved in model training*

a) **Reading and indexing the data**: Before starting any machine learning project, we need to read the data before using it in the project. We read the data using a Python library called pandas which is one of the most popular libraries available for data analysis. The dataset we have used is a CSV file - **GWA-T-1 DAS2.** A CSV file is a table of values separated by commas. The CSV file is the most basic format for data storing among the wide variety of formats available for data storage. The pandas library has an important object - ''Data Frame'. A Data Frame is a table containing an array of *entries*, with each entry having a certain *value*. We converted the dataset mentioned above to a data frame object and indexed it using the column 'JobID' since it contained the corresponding row numbers.

b) **Data Visualization**: The graphical display of information and data is known as data visualization. Data visualization tools make it easy to observe and comprehend trends, outliers, and patterns in data by using visual plots like charts, graphs, and maps. Data visualization tools and technologies are critical in analysing large datasets and making sense out of them. Data visualization tools and packages are helpful when it comes to quickly detecting trends in data, which would otherwise be a time-consuming process.

c) **Data pre-processing**: The process of changing raw data into a usable and understandable format is known as data pre-processing. More often than not, real-world data is prone to inconsistencies in formatting, redundant or unexpected values, human errors and incompleteness. Data preparation solves these problems by making datasets more complete and efficient to analyse. It's an important step that can make or break machine learning projects. It speeds up knowledge extraction from datasets and may have an impact on the performance of ML models

d) **Hyperparameter Optimization**: We ran a few experiments and arrived at the optimal hyperparameters of LightGBM. All the remaining hyperparameters were left at their default values. After tuning, we reached at the below optimal set of hyperparameters:

- feature_fraction = 0.9 (default = 1)
- min_data_in_leaf = 1(default = 20)
- max_depth = 8(default = -1)
- num_leaves = 256(default = 31)
- max_bin = 1024(default = 255)

We chose the learning_rate (learning rate of the model) to be 0.01 for slow convergence and ran the model for 450 iterations. We also set the level of verbosity of LightGBM to 0

e) **Making predictions and calculating performance metrics**: After the tuned hyperparameters were obtained, we made predictions on the training, validation and test dataset using the optimal set of hyperparameters. This is the step where the output is obtained in the form of arrays.

f) **Calculating the performance metrics:** We used two performance metrics - Mean Absolute Error(MAE) and Root Mean Squared Error(RMSE) to measure the performance of our model. We later compared our design to the ones prepared by the authors of our reference paper. This comparison was based on the performance metrics mentioned above.

### B. *Data pre-processing: Steps*

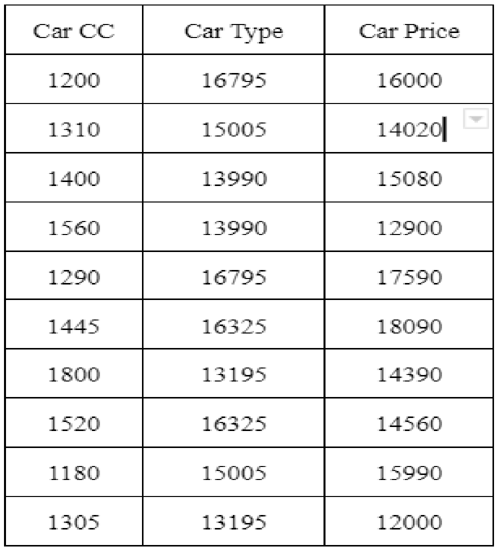
1) Data cleaning - The practice of cleaning datasets by accounting for missing values, removing outliers, correcting inconsistent data points, or smoothing noisy data is known as data cleaning or cleansing. The main goal of data cleaning is to provide complete and accurate samples for machine learning models. In the dataset we used, the column 'ReqTime' (which stands for requested time) had a few -1 values. This is an erroneous value as the requested time can never be -1. We replaced -1 with 0 to make it consistent with the remaining values in the column.

2) Data integration - Data integration is an important aspect of data preparation since data is collected from many sources. Integration may result in multiple inconsistencies and redundant data points, resulting in models that are less accurate. Since we got all our data from the same source and dataset, there was no need for any data integration.

3) Data reduction - Data reduction is used to condense the dataset by reducing the amount of data so that it contains only the important columns. This step maintains the integrity of the original data even though the volume is reduced. The process of selecting a subset of columns or features that are the most important is called feature subset selection. We did feature subset selection first by dropping the columns 'ReqMemory', 'PartitionID', 'UsedNetwork','UsedLocalDiskSpace', UsedResources', 'ReqPlatform', 'ReqNetwork', 'ReqLocalDiskSpace', 'ReqResources', 'VOID' and 'ProjectID' since they contained -1 in all the 11,24,772 rows and then by selecting the 10 best features out of the remaining ones

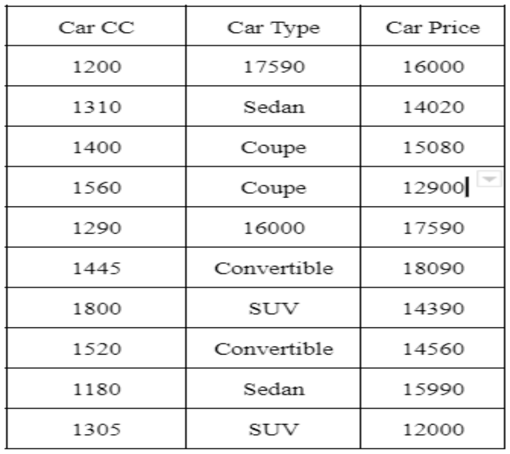
by using SelectKBest which is a class in the Scikit-learn library.

4) Data transformation - Data transformation involves data conversion from one format to another. For instance, if there are string values in our dataset, they need to be converted to numerical values using an appropriate encoding scheme. This is an essential step because machine learning models accept only numerical values as input. The encoder we have used in our project is called the LeaveOneOutEncoder. This is available in the Scikit-learn library. It is a variation of another encoding scheme called Target Encoding. The two encoding schemes are illustrated below :



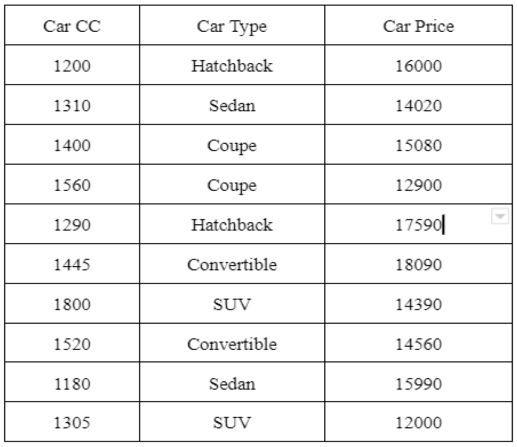
| Car CC | Car Type | Car Price |
|---|---|---|
| 1200 | 16795 | 16000 |
| 1310 | 15005 | 14020 |
| 1400 | 13990 | 15080 |
| 1560 | 13990 | 12900 |
| 1290 | 16795 | 17590 |
| 1445 | 16325 | 18090 |
| 1800 | 13195 | 14390 |
| 1520 | 16325 | 14560 |
| 1180 | 15005 | 15990 |
| 1305 | 13195 | 12000 |

a. Target Encoding: Consider the example dataset mentioned below where 'Car Price' is the output column.



| Car CC | Car Type | Car Price |
|---|---|---|
| 1200 | 17590 | 16000 |
| 1310 | Sedan | 14020 |
| 1400 | Coupe | 15080 |
| 1560 | Coupe | 12900 |
| 1290 | 16000 | 17590 |
| 1445 | Convertible | 18090 |
| 1800 | SUV | 14390 |
| 1520 | Convertible | 14560 |
| 1180 | Sedan | 15990 |
| 1305 | SUV | 12000 |

*In the dataset above, 'Car Type' is a string. In the above though, there are only five car types viz. Sedan, Hatchback, Coupe, SUV and Convertible, in real datasets, there could be way too many of such categories(say,*



| Car CC | Car Type | Car Price |
|---|---|---|
| 1200 | Hatchback | 16000 |
| 1310 | Sedan | 14020 |
| 1400 | Coupe | 15080 |
| 1560 | Coupe | 12900 |
| 1290 | Hatchback | 17590 |
| 1445 | Convertible | 18090 |
| 1800 | SUV | 14390 |
| 1520 | Convertible | 14560 |
| 1180 | Sedan | 15990 |
| 1305 | SUV | 12000 |

*400). One-hot encoding is not a viable option of encoding in such cases. Target encoding converts these string values to numerical values by averaging the target values for a particular string in the 'Car Type' column and replacing these string values by the target average. After target encoding, this is how the above dataset would look like :*

b. Leave One Out Encoding - This is the same as target encoding except that the target value of the string to be replaced is omitted from the average calculation. The original dataset would look like this after applying LeaveOneOutEncoder:

### *C. Performan Metrics*

1. Mean Absolute Error is a model performance metric which is calculated with respect to a test dataset. It is the average of the absolute values of every prediction error of the test dataset.

$$\frac{1}{n}\sum_{j=1}^{n}\left|y_j - \hat{y}_j\right|$$

2. Root Mean Squared Error is a model performance which is calculated with respect to a test dataset. It is the square root of the average of the squares of the prediction errors over the test dataset.

$$\sqrt{\sum_{i=1}^{n}\frac{(\hat{y}_i - y_i)^\wedge 2}{n}}$$

## III. HARDWARE AND SOFTWARE TOOLS AND CONFIGURATIONS

a) Hardware Configuration:

i) Laptop model and configuration: Apple MacBook Air (13", M1, 2020)

ii) Chip details: Apple M1 chip, 8-core CPU with 4 performance cores and 4 efficiency cores, 7-core GPU, 16-core Neural Engine

iii) Memory: 16 GB

b) Software Tools:

i) Sci-kit learn is a free machine learning library for the Python programming language which provides a range of supervised and unsupervised learning algorithms via a consistent interface. It is designed to integrate well with the Python numerical and scientific libraries NumPy and SciPy. The library is focused on modeling data and not on loading, manipulating and summarizing data. For these features, the libraries of NumPy and Pandas were referenced. Some popular models provided by scikit-learn that were integrated into this project include feature extraction, dimensionality reduction, feature selection, cross-validation and supervised models.

ii) Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

iii) Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy.

## *IV. Results of the exploration*

The three weak learners used in LightGBM appeared to be performing better than all algorithms tested in the reference paper that was used as a base work in our experiment. Out of the three, LightGBM Random Forest performed best in terms of Mean Absolute Error (MAE) followed by Gradient-based one-sided sampling (Goss) and Gradient Boosted Decision Tree (GBDT).

Table 1 shows the comparison of MAE and Root Mean Squared Error (RMSE) for all three weak learners and the equation for MAE and RMSE can be given as:

Root Mean Squared Error = $\sqrt{\sum_{i=1}^{n} \frac{(\widehat{y_i} - y_i)}{n}}$

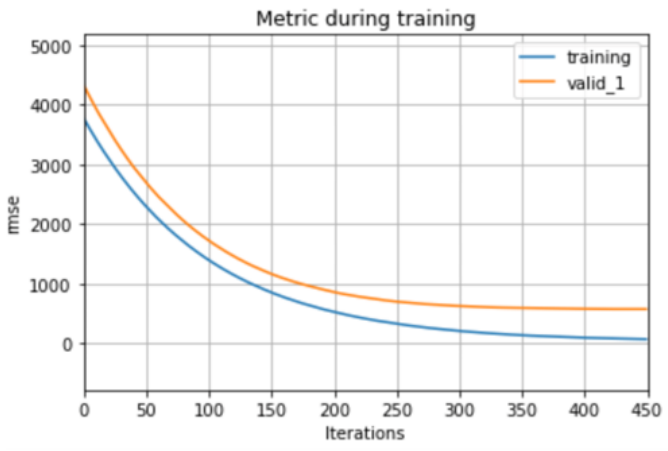
Mean Absolute Error = $\frac{1}{n} \sum_{j=1}^{n} |y_j - \widehat{y_j}|$

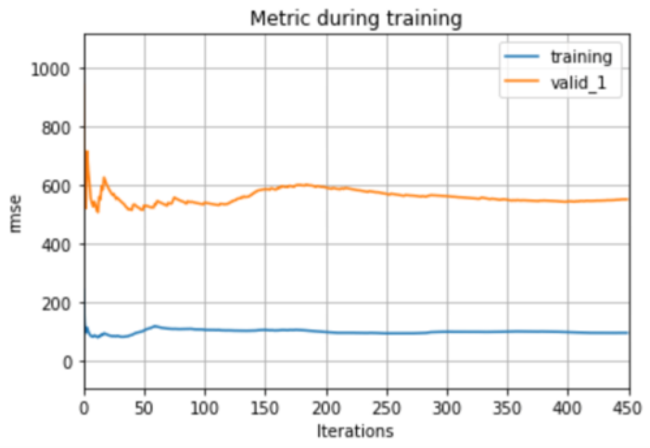
where $n = number\ of\ tuples\ (rows)$ , $\widehat{y_i} = Predicted\ value, y_i = Actual\ value$

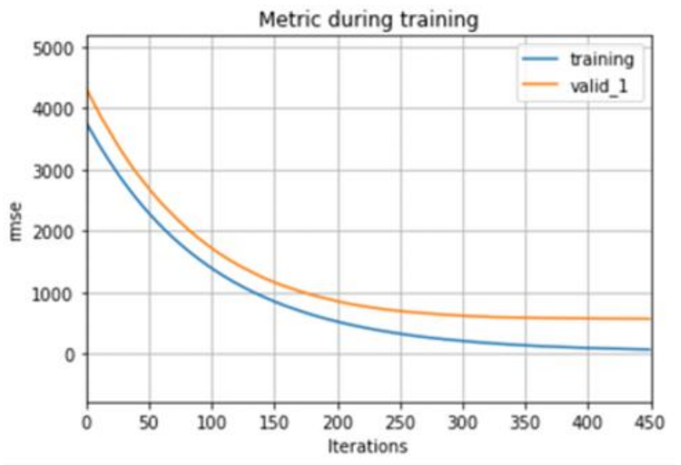
| | Gradient Boosted Decision Tree | LightGBM Random Forest | LightGBM Goss |
|---|---|---|---|
| MAE | 12.406 | 11.870 | 12.213 |
| RMSE | 388.695 | 409.753 | 386.540 |

Table 1

The results were plotted on a graph to observe the convergence. The change in the graphs was seem to be constant after approximately 450 iterations. Combinations of different parameters and their respectively values were experimented in order to obtain an accurate and improved result. All three weak learners seemed to be giving a RMSE score between the range of 385 and 410. The graphs plotted as RMSE vs. Number of iterations are shown below:







The results were compared with previous work by Parthamesh Samal, Sagar Jha, and Raman Kumar Goyal on "CPU Burst-Time Estimation using Machine Learning" as the performance metrics used by them (MAE and RMSE) were similar to what is provided into LightGBM. The algorithms used by them were XGBOOST, KNN, Decision Tree, and Random Forest. The results of their performance metrics are shown in table 2 below.

*Table 2*

| | XGBoost | KNN | Decision Tree | Random Forest |
|---|---|---|---|---|
| MAE | 36.141 | 242.703 | 21.652 | 14.150 |
| RMSE | 342.430 | 2215.790 | 815.458 | 291.774 |

To summarize the outcomes, it is seen that Random Forest gave best performance with an MAE of 14.150 in the reference paper while LightGBM Random Forest gave an improved value of 11.870 for the same performance metric. RMSE at the same time did not improve, but, was still acceptable and close to previous results.

### A. *Summary of Results*

We accomplished the Machine Learning model to improve CPU burst time prediction. Our model predicts an MAE of 11.870 for LightGBM Random Forest which is an improvement of approximately 14.72% when compared with the previous results on Random Forest. Similarly, a significant improvement of 42.7% was observed in Gradient-Boosted Decision Tree when compared with results of Decision Tree in previous experiments.

We chose LightGBM as it has many advantages like faster training, high efficiency, low memory usage, and better accuracy and the same were reflected in our experiment while training the model. The biggest advantage of LightGBM for our experiment was its capability to handle large-scale data as the GWA-T-1 DAS2 dataset have more than 11,24,772 rows and 29 features and using just 30% of the same to train the model was also a challenge.

Moreover, LightGBM uses a leaf-wise tree growth algorithm, which is unlike many other algorithms that uses depth-wise growth. Leaf-wise growth algorithms converge faster than depth-wise. This is the reason they involve a risk of overfitting. Further, to deal with overfitting, parameters like "min_data_in_leaf" and "num_leaves" can be optimized. These parameters greatly helped us dealing with the problem of overfitting in our experiment. "Max_depth" that defines the depth of the tree was also used to optimize the performance of the model.

It is evident that using a boosting algorithm like LightGBM can actually improve the performance of a weak learner like Random Forest. It is worth noting that the improvement was seen by only tuning few of the parameters of the algorithm while leaving rest at their default values.

## V. RETROSPECTIVE

Adding more data, adds diversity. It decreases the generalization error because the model becomes more general by virtue of being trained on more examples. With more computation power and training with a larger fraction of data from the available dataset we can get even more accurate results. We can also try to vary the test and train data ratio to get better results.

Hyper-parameters are used to tune the model, the accuracy and efficiency of the algorithm can be improved by tuning these parameters.

New untested parameters can be added to tune the performance of the model.

We can carry out experiments with other boosting algorithms like ADABoost to carry out prediction of CPU burst time.

## REFERENCES

[1] P. Samal, S. Jha and R. K. Goyal, "CPU Burst-Time Estimation using Machine Learning," 2022 IEEE Delhi Section Conference (DELCON), 2022, pp. 1-6, doi: 10.1109/DELCON54057.2022.9753639.

[2] T. Helmy, S. Al-Azani and O. Bin-Obaidellah, "A Machine Learning-Based Approach to Estimate the CPU-Burst Time for Processes in the Computational Grids," 2015 3rd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS), 2015, pp. 3-8, doi: 10.1109/AIMS.2015.11.

[3] M. R. Mahesh Kumar, B. R. Rajendra, C. K. Niranjan and M. Sreenatha, "Prediction of length of the next CPU burst in SJF scheduling algorithm using dual simplex method," Second International Conference on Current Trends In Engineering and Technology - ICCTET 2014, 2014, pp. 248-252, doi: 10.1109/ICCTET.2014.6966296.K. Elissa, "Title of paper if known," unpublished.

[4] A. Pourali and A. M. Rahmani, "A Fuzzy-Based Scheduling Algorithm for Prediction of Next CPU-Burst Time to Implement Shortest Process Next," 2009 International Association of Computer Science and Information Technology - Spring Conference, 2009, pp. 217-220, doi: 10.1109/IACSIT-SC.2009.83.