Efficient CPU Burst Forecasting with Low-Latency Machine Learning Models

***Abstract*—Efficient CPU scheduling in operating systems depends on the accurate prediction of process CPU burst times. Traditional methods such as exponential averaging (EA) provide simple estimates but often perform poorly under dynamic and heterogeneous workloads. This study investigates the use of machine learning (ML) models, *k*-nearest neighbors (KNN), decision tree (DT), random forest (RF), and multilayer perceptron (MLP), to predict CPU burst times using only submission-time attributes from the GWA-T-4 AuverGrid dataset. These ML models are evaluated against the conventional EA method. In addition to predictive performance, we assess the models' inference latency to examine their suitability for real-time scheduling. While KNN achieves high predictive accuracy, its relatively high inference time may constrain its use in latency-sensitive environments. Our experimental results show that all ML models outperform EA across key metrics, with the decision tree offering the most favorable balance between accuracy (mean absolute error (MAE) of 3514.09 and correlation coefficient (CC) of 0.84) and efficiency (0.0033 ms per sample). Confidence interval analysis and hyperparameter tuning are also conducted to assess model robustness. These findings demonstrate the feasibility of accurately predicting submission-time bursts using machine learning techniques, supporting more effective scheduling decisions in real-world systems.**

***Index Terms*—CPU Scheduling, Machine Learning, Burst Time Prediction, Exponential Averaging.**

I. Introduction

Efficient process scheduling remains a critical component of modern operating system (OS) design, significantly influencing key performance indicators such as CPU utilization, average waiting time, turnaround time, and system responsiveness. Among various CPU scheduling algorithms, Shortest Job First (SJF) and Shortest Remaining Time First (SRTF) are theoretically optimal in minimizing average waiting time in a uniprocessor computing system. However, both algorithms require prior knowledge of the CPU burst time, the length of time a process needs the CPU before completing or initiating I/O operations, which is generally unknown at the time of scheduling.

In general, OS designs rely on techniques such as EA to estimate future CPU burst lengths based on past execution history. EA predicts the next CPU burst by assigning a weighted average between the most recent actual burst and the previous prediction, controlled by a smoothing factor. While EA is computationally lightweight and simple to implement, its predictive ability declines significantly in environments characterized by high workload variability, process heterogeneity, or sudden resource demand shifts. Moreover, EA's dependence on historical observations makes it ineffective for newly arriving processes without any prior execution data [1], [2]. To overcome these limitations, recent research has explored the application of ML techniques to predict CPU burst times. ML models are capable of capturing complex, nonlinear relationships between process attributes and burst durations, offering a data-driven alternative to traditional heuristics. Prior work by [2] demonstrated that models such as support vector machines (SVM), DT, and KNN can substantially outperform traditional methods when trained on real-world workload datasets. Similarly, studies like [3] showed the efficacy of ML approaches in execution time prediction for grid and cloud applications, even without historical execution logs.

Numerous works have explored burst time prediction using statistical and intelligent methods to enable SJF-like scheduling. Fuzzy-based approaches estimate the next CPU burst using past patterns [14], and later refinements integrate fuzzy inference into the exponential averaging formula for better flexibility [15]. More recent studies apply machine learning models such as decision trees and KNN to predict burst times [16], but typically omit runtime feasibility analysis. Some rely on idealized simulators with unrealistic assumptions like known burst durations [17], while others address fairness or probabilistic selection [18] without tackling real-time constraints.

Extending on these efforts, this study [12] applied ensemble models such as RF, XGBoost, and DT to the GWA-T-1 DAS2 dataset, achieving high predictive accuracy and found that RF is the best-performing model. The work [13] further developed a real-time prediction framework using linear and tree-based regressors on the GWA-T-4 AuverGrid dataset. Another work [10] explored a different approach by using ML classifiers to select the most efficient scheduling algorithm based on process attributes. The paper [11] provided a detailed analysis of traditional CPU scheduling algorithms and highlighted the difficulty in accurately estimating burst time, an issue that modern ML techniques aim to overcome. The predictive power of submission-time attributes such as memory requests, user identifiers, and submit timestamps are collectively demonstrated in these studies.

Recent studies on CPU burst time prediction using ML have predominantly examined tree-based or linear models, without providing a thorough assessment of their suitability for real-time scheduling environments. Existing research often omits the evaluation of inference time and operational latency, which are key factors for integration into practical, time-sensitive systems. Moreover, although MLPs offer strong capabilities for modeling complex nonlinear patterns, their use in this context has not been adequately investigated. This work ad-

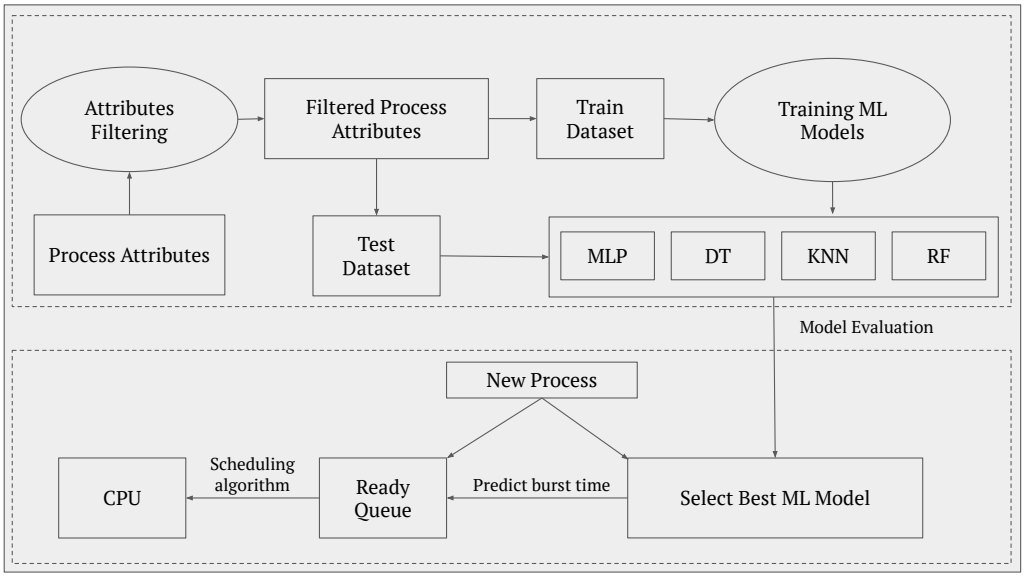

Fig. 1. Overall workflow for CPU burst time prediction and integration with scheduling system.

dresses these gaps by including MLPs in the comparative analysis and by conducting a detailed feasibility study that measures per-sample inference time and quantifies other ML model robustness through confidence interval analysis. We also compare the performance of several ML models, i.e., KNN, DT, and RF, on the GWA-T-4 AuverGrid dataset [9], incorporating hyperparameter tuning, inference latency analysis, and confidence interval estimation to determine the applicability of these models for real-time scheduling implementation.

This study presents two major contributions by providing a feasible framework for incorporating ML models into dynamic OS scheduling by considering both predictive accuracy and practical deployment factors such as inference latency and model reliability by analyzing confidence intervals. The findings support the development of intelligent CPU scheduling, particularly for newborn processes in dynamic and heterogeneous systems.

II. Dataset Description

A. GWA-T-4 AuverGrid Dataset Overview

The dataset used in this study is the GWA-T-4 AuverGrid workload archive [9], collected from a real-world production grid environment comprising multiple geographically distributed clusters. AuverGrid consists of five clusters totaling 475 CPUs, with each cluster equipped with Dual 3GHz Pentium IV Xeons running Scientific Linux. These clusters are geographically located in the Auvergne region of France, offering a realistic and heterogeneous environment for workload generation.

The full workload archive contains 404,176 job records, with each record comprising 29 attributes that describe submission-time parameters, resource requests, execution behavior, and other metadata such as user and system identifiers. Such a rich and detailed dataset provides an excellent foundation for building predictive models aimed at improving CPU scheduling efficiency.

B. Data Preprocessing

For the scope of this study, we selected the first 5,000 job records based on chronological order. Jobs with missing or invalid entries, such as those with RunTime set to -1, were excluded during preprocessing. This step resulted in a clean dataset containing 4,340 valid entries. The dataset was partitioned into a training set comprising 80% of the records (3,472 jobs) and a testing set comprising the remaining 20% (868 jobs).

C. Attribute Selection

Feature selection was guided by the objective of simulating realistic early-stage prediction scenarios where only submission-time information is available. Accordingly, the features were classified into two broad categories: non-historical attributes and historical attributes.

Non-historical attributes refer to those features that are immediately available at the time of job submission. Attributes such as SubmitTime, UserID, UsedMemory, ReqTime, and ReqMemory are critical in capturing the initial profile of a job without relying on any execution history.

Historical attributes, on the other hand, are features that are only available after the job has begun execution or completed execution phases. These features were also included in secondary experiments for comparison purposes but were excluded from the models intended for early prediction. Historical attributes combine both historical and submission-time information, including features like WaitTime, NProcs, AverageCPUTimeUsed, and Status.

III. Methodology

The overall workflow of the proposed CPU burst time prediction system is illustrated in Figure 1. The methodology comprises data preparation, model training, model evaluation, and final deployment for real-time burst prediction at process submission. Initially, raw process attributes are extracted from the workload dataset. A feature filtering stage is employed to select only the submission-time attributes, excluding features that require historical or execution-based information. The filtered attributes are then partitioned into a training set and a testing set. The training data is used to develop predictive models, while the testing data is reserved for evaluation purposes.

MLP, DT, KNN, and RF are trained multiple times independently to select the best parameters for each of the models. Final models were trained using the best parameters after the fine-tuning. Each model's performance is evaluated on the test set using several metrics, including MAE, Coefficient of Determination (R^2), CC, and Relative Absolute Error (RAE). The best-performing model is selected based on a balance between prediction accuracy and computational efficiency. In the final deployment phase, when a new process arrives, its submission-time attributes are used to predict the expected CPU burst time using the selected model. This predicted value assists the scheduling algorithm in ordering the ready queue more intelligently, aiming to improve overall system performance.

A. Classical Technique: Exponential Averaging

One classical approach for predicting CPU burst times is the EA method. EA predicts the next CPU burst length by computing a weighted average between the last observed burst time and the previous prediction [1]. The prediction at time step $n+1$ is given by:

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n \tag{1}$$

where τ_{n+1} is the predicted next burst time, t_n is the actual measured burst time of the last execution, τ_n is the previous prediction, and α is a smoothing constant between 0 and 1. A smaller α assigns more weight to historical data, resulting in smoother but slower-to-adapt predictions, whereas a larger α makes the model highly responsive to recent changes but more volatile. Although computationally efficient with constant time complexity, EA often fails in dynamic environments, particularly when no prior history is available for new processes [2].

B. Model Training and Validation

All ML models were trained on 80% of the available dataset and validated on the remaining 20%. Five-fold cross-validation was employed during the hyperparameter tuning phase to ensure that selected parameters generalized well to unseen data.

For KNN, the optimal number of neighbors k was selected as 7 for non-historical attributes and 3 for historical attributes. DT was tuned by adjusting the maximum tree depth between 5 and 20, with the best performance achieved at a maximum depth of 20 in both cases. RF models were constructed using 50 estimators with a maximum depth of 20 for non-historical attributes, and 100 estimators with a maximum depth of 20 for historical attributes, balancing model complexity and variance reduction.

The MLP model architecture featured four hidden layers with 128, 64, 32, and 16 neurons, respectively, utilizing ReLU activation functions and optimized using the Adam optimizer. Early stopping was applied during MLP training to halt learning once validation loss plateaued, preventing overfitting due to excessive training epochs.

All models were evaluated not only based on predictive accuracy but also computational efficiency during inference, recognizing that real-time scheduling scenarios demand models capable of delivering rapid and stable predictions.

IV. Results and Analysis

A. Performance of Exponential Averaging

The performance of the EA method for different smoothing factors α is illustrated in Figure 2. In this figure, the CPU burst time predictions for α values ranging from 0.1 to 0.9 are compared against the actual burst times. The red solid line represents the actual observed burst times, while the dashed lines correspond to different α values.

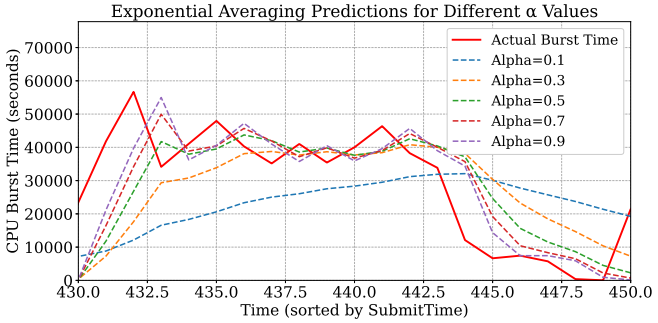

Fig. 2. Exponential Averaging predictions for different α values compared to actual CPU burst times.

From Figure 2, it is observed that smaller values of α, such as $\alpha = 0.1$, result in slower responsiveness, causing the predicted burst times to lag behind the actual values significantly. On the other hand, higher α values, such as $\alpha = 0.9$, respond more quickly to changes but introduce noticeable prediction instability. The intermediate values of α, particularly around $\alpha = 0.5$, show a reasonable compromise between responsiveness and prediction stability.

To quantitatively evaluate the EA method, four evaluation metrics, i.e., MAE, R^2, CC, and RAE were computed for each α setting. Table I summarizes the performance.

As presented in Table I, increasing α generally reduces MAE and RAE, indicating better responsiveness to recent process behavior. However, it is notable that while $\alpha = 0.9$ yields the lowest MAE and RAE values, it also results in the poorest R^2 score, suggesting less reliable overall model fitting. The R^2 value, which measures the proportion of variance explained by the model, declines significantly beyond $\alpha = 0.5$.

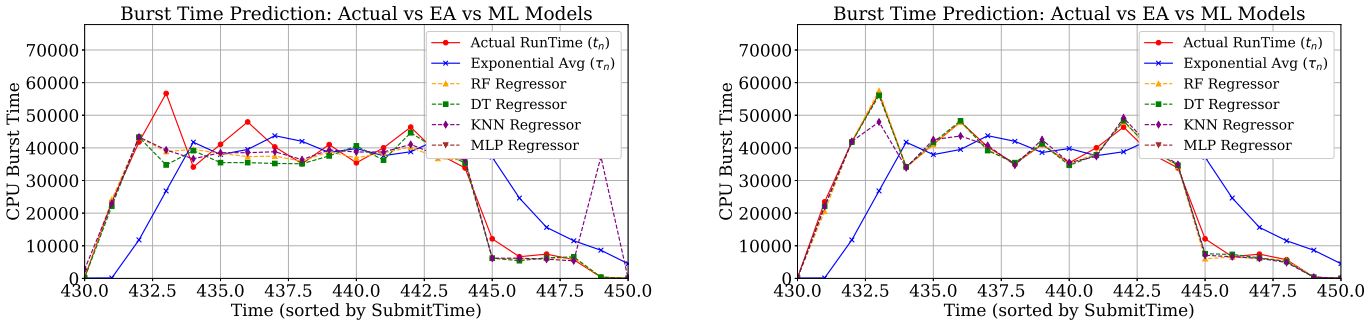

Fig. 3. Burst time prediction comparison: (left) using non-historical attributes; (right) using historical attributes.

TABLE I
PERFORMANCE OF EXPONENTIAL AVERAGING FOR DIFFERENT α VALUES

Alpha	MAE	R^2	CC	RAE (%)
$\alpha = 0.1$	20235	0.246	0.500	72.76
$\alpha = 0.3$	16372	**0.340**	0.599	58.87
$\alpha = 0.5$	14833	0.322	**0.605**	53.34
$\alpha = 0.7$	14048	0.260	0.588	50.51
$\alpha = 0.9$	**13605**	0.158	0.561	**48.92**

Considering all evaluation metrics together, $\alpha = 0.5$ provides a good trade-off between prediction accuracy and model stability. Although slightly higher MAE and RAE are observed compared to $\alpha = 0.9$, the stronger R^2 and CC values at $\alpha = 0.5$ justify its selection for further comparative analysis against machine learning models.

B. Comparison of Machine Learning Models

To assess the effectiveness of different models in predicting CPU burst time, we first visualize the predicted burst times compared to actual burst times using both non-historical and historical attributes. The burst time prediction comparison across different models using non-historical and historical attributes is shown in Figure 3.

As shown in Figure 3, when models are trained using only non-historical attributes, EA significantly lags in accurately predicting burst times, especially for processes with abrupt changes in execution length. ML models, particularly RF and DT, align much more closely with the actual burst time curve, demonstrating better adaptation to varying workloads. MLP also follows the trend but exhibits slightly higher oscillations compared to tree-based models, while KNN predictions show moderate deviations, especially at higher burst times.

When historical attributes are incorporated, as illustrated in Figure 3, all ML models achieve visibly tighter alignment with the actual burst times. RF and DT almost perfectly track the real curve, indicating that additional historical information significantly improves prediction quality. MLP's prediction becomes smoother and closer to the ideal trend, while KNN also benefits but still exhibits small deviations for longer bursts. Meanwhile, EA remains the least accurate, highlighting its inherent limitations regardless of feature set expansion.

These visualizations clearly emphasize that even when restricted to non-historical attributes, ML models offer superior predictive performance compared to EA. However, including historical attributes further enhances prediction accuracy, especially for more complex models like RF and DT.

C. Performance Evaluation

The evaluation of model performance is illustrated in Figure 4, where each model's performance on non-historical and historical attributes is compared across four evaluation metrics: MAE, R^2, CC, and RAE.

As seen in Figure 4, all ML models substantially outperform EA across all evaluation metrics. The inclusion of historical attributes consistently improves model performance across all models, further widening the performance gap between EA and ML approaches. Focusing first on MAE, DT achieves the lowest MAE value (1150.83) when historical attributes are incorporated, closely followed by RF (1505.44). RF shows the best performance in MAE in the case of non-historical features. In contrast, EA maintains a very high MAE (14833.00) regardless of the feature set.

Analyzing the R^2 metric, MLP achieves the highest R^2 score (0.94) with historical attributes, followed by RF (0.93). KNN performs best with non-historical information. EA again performs poorly, with an R^2 value of 0.32.

When examining the CC, MLP, RF, and DT models reach CC values above 0.95 when historical attributes are used, indicating a very strong linear alignment between predicted and actual burst times. KNN also shows a notable improvement. In contrast, EA remains at a weak CC of approximately 0.60 under both settings.

RAE further highlights the benefits of historical attributes. DT achieves the lowest RAE (4.14%), followed by RF (5.41%) and KNN (6.89%) when using historical features. All models significantly reduce RAE compared to their non-historical versions, whereas EA remains consistently high at around 53.34%.

Overall, these comparisons confirm that ML models significantly surpass traditional EA in all predictive metrics. Furthermore, integrating historical attributes leads to substantial gains, particularly for KNN and MLP. Even when limited to

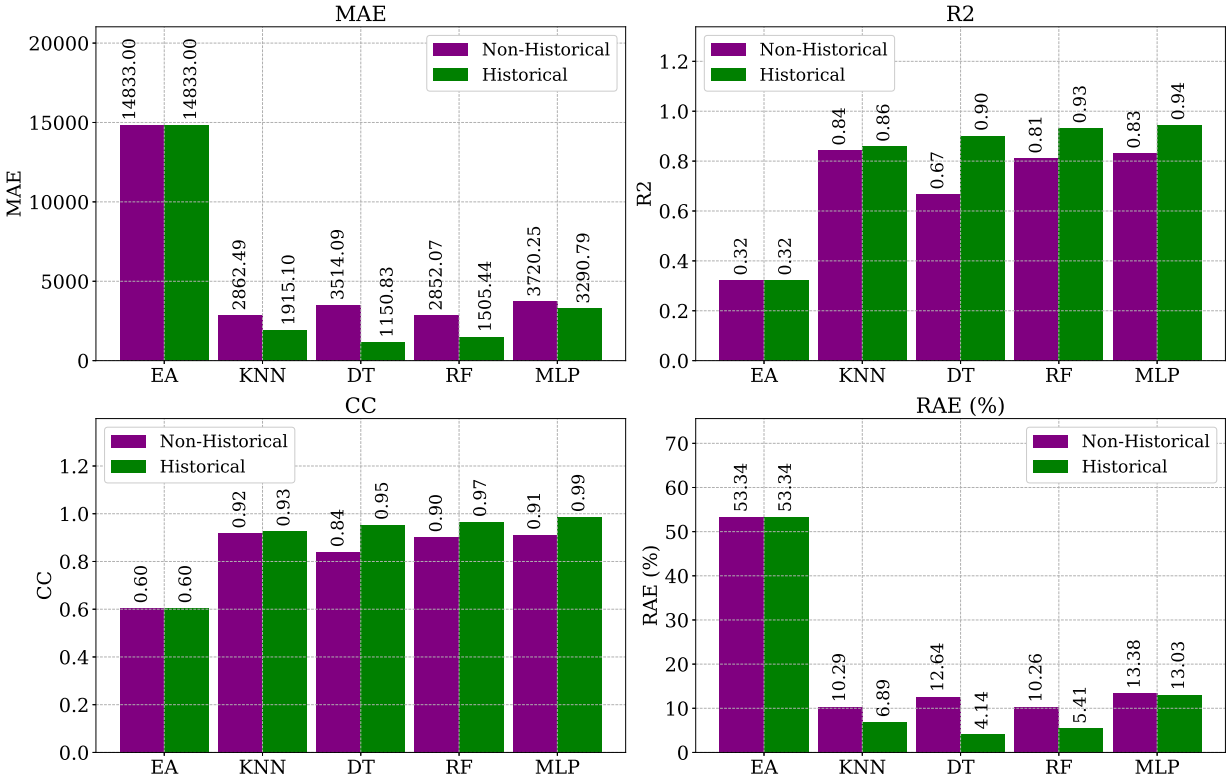

Fig. 4. Model Performance Comparison: Historical vs Non-Historical Attributes based on MAE, R^2, CC, and RAE.

non-historical attributes, ML models still offer a significant predictive advantage over traditional approaches.

D. Inference Time Comparison

Inference latency is crucial for real-time schedulers. Table II presents the average total and per-sample inference times for each method.

TABLE II
COMPARISON OF AVERAGE INFERENCE TIME

Model	Avg. Total Time (s)	Avg. Per-Sample Time (ms)
EA	0.00313	0.003606
MLP	0.021633	0.024900
RF	0.057667	0.066500
KNN	0.011267	0.012933
Decision Tree	**0.002900**	**0.003333**

Although EA remains the fastest due to its $\mathcal{O}(1)$ computation, DT offers better and comparable inference speed with better accuracy. In contrast, KNN and MLP models introduce higher inference latency, which may be prohibitive for time-critical scheduling environments. Therefore, we are considering DT more consistent and reliable for practical implementation, even if KNN and MLP have better accuracy.

E. Confidence Interval Analysis

To assess model stability, each ML model was trained and evaluated five times using different random splits of the dataset. For each configuration, the 95% confidence intervals for the key evaluation metrics (MAE, R^2, CC, and RAE) were computed.

The aggregated results are presented in Table III, showing the mean performance along with the 95% confidence intervals for both non-historical and historical attributes. The results show that **DT** exhibited the lowest variance across runs, with a narrow 95% confidence interval (e.g., $\pm$ 495.10 ms for MAE on non-historical attributes). **RF** displayed slightly higher variability, especially with non-historical attributes.

Overall, the confidence intervals confirm that DT and KNN offer not only superior predictive performance but also greater robustness under varying data partitions, strengthening their suitability for real-world deployment.

F. Data and Code Availability

The codebase developed for this study has been made publicly available. The repository contains all scripts used for data preprocessing, model training, evaluation, and figure generation. The full code and data are accessible at: https://github.com/nsriad/Burst-Time-Prediction-Using-Machine-Learning.git

V. CONCLUSION

Accurate prediction of CPU burst times remains a cornerstone challenge for achieving efficient scheduling in uniprocessing OSs. This study has demonstrated that ML models, particularly DT and KNN, can significantly outperform EA

TABLE III
PERFORMANCE SUMMARY OF KNN, DECISION TREE, AND RANDOM FOREST ON NON-HISTORICAL AND HISTORICAL ATTRIBUTES (MEAN ± 95% CONFIDENCE INTERVAL)

Model	Attribute Type	MAE	R^2	CC	RAE (%)
KNN	Non-Historical	2862.49 ± 524.02	0.843 ± 0.0497	0.918 ± 0.0256	10.29 ± 1.88
	Historical	1542.93 ± 465.32	0.9029 ± 0.0330	0.9503 ± 0.0174	5.62 ± 1.468
Decision Tree	Non-Historical	3514.09 ± 495.10	0.667 ± 0.0581	0.840 ± 0.0306	12.64 ± 1.45
	Historical	782.18 ± 455.34	0.9439 ± 0.0569	0.9720 ± 0.0284	2.83 ± 1.7596
Random Forest	Non-Historical	2852.08 ± 573.81	0.811 ± 0.0579	0.902 ± 0.0305	10.26 ± 1.67
	Historical	1092.86 ± 279.05	0.9415 ± 0.0243	0.9705 ± 0.0122	3.96 ± 0.9168

when trained on readily available non-historical attributes at submission time.

Using the GWA-T-4 AuverGrid dataset, we evaluated multiple ML models, including KNN, DT, RF, and MLP. DT emerged as the most balanced solution, achieving substantial improvements in MAE and CC while maintaining inference times comparable to EA. Through confidence interval analysis, we further validated the robustness and stability of the best-performing models.

These findings suggest a practical pathway toward integrating lightweight ML predictors into real-time CPU schedulers, enabling smarter scheduling decisions without the need for historical execution logs. By leveraging attributes available at process submission, operating systems can potentially approximate burst times more accurately and adaptively, enhancing overall system performance.

Future work can extend this study by exploring incremental and online learning methods that continuously adapt models as new job data becomes available. Furthermore, examining sequential models, such as recurrent neural networks (RNNs) or transformers, may further improve prediction performance by capturing temporal relationships in job submission patterns. Integrating the proposed predictors into a kernel-level scheduling simulator and benchmarking their impact on system-level metrics such as average waiting time, throughput, and fairness will also offer valuable insights for practical deployment.

REFERENCES

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 9th ed. Hoboken, NJ, USA: Wiley, 2013.

[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," in *Proc. 3rd Int. Conf. Artif. Intell., Modelling and Simulation (AIMS)*, 2015, pp. 67–72.

[3] A. Matsunaga and J. A. B. Fortes, "On the Use of Machine Learning to Predict the Time and Resources Consumed by Applications," in *Proc. IEEE 10th Int. Conf. Cluster, Cloud and Grid Comput. (CCGrid)*, 2010, pp. 495–504.

[4] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[5] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[6] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[8] W. Smith, I. Foster, and V. Taylor, "Predicting Application Run Times with Historical Information," *Journal of Parallel and Distributed Computing*, vol. 64, no. 9, pp. 1007–1016, 2004.

[9] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. J. Epema, "The Grid Workloads Archive," *Future Generation Computer Systems*, vol. 24, no. 7, pp. 672–686, 2008.

[10] S. Biswas, M. S. Ahmed, M. J. Rahman, A. Khaer, and M. M. Islam, "A Machine Learning Approach for Predicting Efficient CPU Scheduling Algorithm," in *Proc. 5th Int. Conf. Sustainable Technologies for Industry 5.0 (STI)*, Dhaka, Bangladesh, 2023, pp. 1–6, doi: 10.1109/STI59863.2023.10464816.

[11] S. Pemasinghe and S. Rajapaksha, "Comparison of CPU Scheduling Algorithms: FCFS, SJF, SRTF, Round Robin, Priority Based, and Multilevel Queuing," in *Proc. IEEE 10th Region 10 Humanitarian Technology Conf. (R10-HTC)*, Hyderabad, India, 2022, pp. 318–323, doi: 10.1109/R10-HTC54060.2022.9929533.

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

[13] A. R. Panda, S. Sirmour, and P. K. Mallick, "Real-Time CPU Burst Time Prediction Approach for Processes in the Computational Grid Using ML," in *Advances in Intelligent Computing and Communication*, M. N. Mohanty and S. Das, Eds., vol. 430, Lecture Notes in Networks and Systems. Singapore: Springer, 2022, pp. 665–678, doi: 10.1007/978-981-19-0825-5_58.

[14] A. Pourali and A. M. Rahmani, "A Fuzzy-Based Scheduling Algorithm for Prediction of Next CPU-Burst Time to Implement Shortest Process Next," in *Proc. IACSIT Spring Conf.*, 2009, pp. 217–220.

[15] Vandana, "Revised Formula for Estimating CPU-Burst," *J. Int. Acad. Phys. Sci.*, vol. 22, no. 4, pp. 345–354, 2018.

[16] K. Vayadande *et al.*, "Comparative Study on Calculating CPU Burst Time Using Different Machine Learning Algorithms," in *Proc. Int. Conf. Advancement in Tech. (ICONAT)*, IEEE, 2023, pp. 1–6.

[17] B. T. Nguyen, H. Q. Ta, and T. N. Le, "Requirements of Realistic Simulators for CPU Scheduling Algorithms," in *Proc. Int. Conf. Adv. Technol. Commun. (ATC)*, IEEE, 2024, pp. 191–196.

[18] S. Prasanna, A. S. Gulati, and A. H. C., "Fairness in CPU Scheduling: A Probabilistic Algorithm," in *Proc. Int. Conf. Circuit Power Comput. Technol. (ICCPCT)*, IEEE, 2024, pp. 1031–1036.