International Journal of Emerging Engineering and Technology (IJEET)
ISSN (e): 2958-3764
Volume: 5, Number: 1-2, Pages: 28 - 45, Year: 2026 (Special Issue)

Enhancing SJF and SRTF CPU Scheduling Through Machine Learning-Based Burst Time Prediction: A Comparative Study

Haffiz-Ud-Din * Musharraf Ali and Anzal Hussain

Department of Research and Development, Gilgit-Baltistan First Research Center, Binary HUB, Gilgit-Baltistan, Pakistan

*Corresponding author: Haffiz-Ud-Din (Email: haffizuddin7899@gmail.com)

Received: 04/07/2026, Revised: 30/08/2026, Accepted: 02/09/2026

Abstract— **Accurate prediction of CPU burst time is essential for effective process scheduling, particularly in Shortest Job First (SJF) and Shortest Remaining Time First (SRTF) algorithms. Traditional estimation techniques, such as exponential averaging, can produce substantial prediction errors, resulting in inefficient scheduling, increased waiting and turnaround times, and reduced system throughput. Although machine learning (ML) approaches have recently been investigated for CPU burst time prediction, existing studies are often limited by small datasets, restricted model comparisons, inadequate preprocessing, and insufficiently reproducible implementation frameworks. This study addresses these limitations by developing a comprehensive ML-based framework for CPU burst time prediction using a large-scale real-world workload dataset containing more than 404,000 processes and 29 features.**

The proposed framework evaluates a broad range of supervised regression algorithms, including Linear Regression, Ridge, Lasso, ElasticNet, Decision Tree, Random Forest, Gradient Boosting, AdaBoost, XGBoost, LightGBM, K-Nearest Neighbors, Support Vector Regression, and Multi-Layer Perceptron. The experimental pipeline incorporates exploratory data analysis, strict train-test separation to prevent data leakage, missing-value imputation, outlier capping, categorical encoding, feature engineering, and standardized performance evaluation using R^2, adjusted R^2, RMSE, MAE, and MAPE.

Experimental results demonstrate that gradient-boosting approaches substantially outperform conventional and alternative ML models. XGBoost achieved the best performance, attaining a test R^2 of 0.872, RMSE of 487.3 s, MAE of 196.5 s, and MAPE of 14.2%, compared with approximately 0.70 R^2 and 30% MAPE for linear baselines. LightGBM achieved a comparable R^2 of 0.868 while reducing training time by 43%. Feature importance analysis identified AverageCPUTimeUsed, ReqTime, UsedMemory, and NProcs as the dominant predictors, collectively contributing more than 74% of predictive power. The findings indicate strong potential for integrating ML-based burst time prediction into SJF and SRTF scheduling. However, scheduler-level simulations and validation across diverse workload environments are required to quantify actual improvements in waiting time, turnaround time, and throughput.

Index Terms— **CPU Scheduling, Burst Time Prediction, Machine Learning, SJF, SRTF, Gradient Boosting, Regression Models.**

I. Introduction

In modern operating systems, efficient CPU scheduling remains one of the most fundamental responsibilities for maximizing system performance, resource utilization, and user satisfaction. Among various scheduling algorithms, First-Come-First-Served (FCFS), Shortest Job First (SJF), and Shortest Remaining Time First (SRTF) are classical approaches widely studied in operating system design [1]. FCFS is simple and fair but suffers from the convoy effect, where short processes wait behind long ones, leading to poor average waiting times. SJF and its preemptive variant SRTF aim to minimize average waiting time by prioritizing processes with the shortest CPU burst times. Theoretically, SJF and SRTF are optimal for minimizing average waiting time in non-preemptive and preemptive environments, respectively [1,2]. However, their practical effectiveness heavily depends on the accurate prediction of future CPU burst times, which are inherently unknown at the time of scheduling decisions.

Traditional burst time estimation techniques, particularly exponential averaging (also known as aging) [1], rely on historical data using,

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

where τ_{n+1} is the predicted burst time, t_n is the actual burst time of the n-th process, and α is a smoothing factor ($0 < \alpha < 1$). While simple and computationally efficient, exponential averaging suffers from significant limitations: it assumes linear relationships, struggles with sudden changes in process behavior, cannot utilize the rich contextual features available in modern systems, and typically achieves limited predictive accuracy with an estimated R^2 in the range of 0.50–0.60 when evaluated on complex real-world workloads [2], [3-5]. These inaccuracies lead to suboptimal scheduling decisions, increased average waiting and turnaround times, and inefficient CPU utilization.

This work is licensed under a Creative Commons AttributionShareAlike4.0 International License, which permits Unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited

A. Research Gap and Novelty

The emergence of Machine Learning (ML) and data-driven approaches offers a promising solution to this long-standing challenge. By leveraging large historical process datasets containing rich features such as memory usage, number of processors, submission times, and resource requirements, ML models can learn intricate patterns to predict CPU burst times with significantly higher accuracy than conventional statistical methods. Several studies have explored the application of artificial intelligence in process scheduling [6]-[8]. However, previous works suffer from critical limitations that this study directly addresses in Table I.

TABLE I: Limitations in the prior research work.

Limitation In Prior Work	**How This Study Addresses It**
Small or synthetic datasets that fail to capture real-world workload complexity	Uses A Large-Scale Production Workload Trace Of 404,176 Processes With 29 Features
Evaluation of only 3-5 basic ml algorithms without modern gradient boosting methods	Implements And Compares Twelve Diverse Regression Models Spanning Linear, Tree-Based, Ensemble, Boosting, Instance-Based, And Neural Network Architectures
Limited preprocessing of noisy real-world data with potential test information leakage	Implements A Rigorous Preprocessing Pipeline With Explicit Train/Test Separation, Proper Imputation, And Outlier Handling To Prevent Data Leakage
Lack of thorough comparative analysis and performance diagnostics	Provides Comprehensive Evaluation Using Multiple Metrics (R^2, Adjusted R^2, RMSE, MAE, MAPE), Overfitting Analysis, Feature Importance, And Visualization
No reusable prediction framework for practical deployment	Provides A Production-Ready Predict_Cpu_Burst() Function And Complete Open-Source Implementation
Absence of statistical rigor in model comparison	Employs Fixed 70/30 Train-Test Split With Reproducibility (Random_State=42) And Recommends Cross-Validation For Future Work

B. Primary Objectives

The primary objectives of this study are:

1. To analyze the effectiveness of various machine learning models in predicting CPU burst times from real-world workload data.
2. To perform extensive data preprocessing, feature engineering, and exploratory analysis on a large-scale dataset while preventing data leakage.
3. To conduct a fair and comprehensive comparative evaluation using multiple performance metrics.
4. To identify the most suitable models for integration into modern operating system schedulers.
5. To provide a reusable prediction framework for future research and practical applications.

C. Key Contributions

The key contributions of this paper include:

- A detailed end-to-end machine learning pipeline for CPU burst time prediction with proper leakage prevention.
- In-depth comparative analysis of twelve regression models (Linear, Ridge, Lasso, ElasticNet, Decision Tree, Random Forest, Gradient Boosting, AdaBoost, XGBoost, LightGBM, KNN, SVR, MLP) with extensive visualization and statistical insights.
- Identification of XGBoost and LightGBM as the highest-performing models, achieving test R^2 scores of 0.872 and 0.868 respectively, substantially outperforming linear baselines (R^2≈0.70) and traditional exponential averaging.
- Feature importance analysis revealing that AverageCPUTimeUsed, ReqTime, UsedMemory, and NProcs are the dominant predictors, providing actionable insights for OS designers.
- A ready-to-use predict_cpu_burst() function and complete open-source implementation for practical deployment and reproducibility.
- Empirical evidence that ML-based burst prediction can reduce RMSE by over 35% compared to linear models, potentially leading to significantly improved SJF/SRTF scheduling performance.

II. Literature Review

This section provides a comprehensive review of existing work on CPU burst time estimation, machine learning applications in operating system scheduling, and identifies the specific research gaps that motivate this study.

A. Traditional Burst Time Estimation Techniques

The most widely used traditional method for estimating CPU burst time is **Exponential Averaging** (also known as Aging), introduced in the classic operating systems textbook by Silberschatz et al. [1]. This technique predicts the next CPU burst using the formula:

$$\tau_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot \tau_n \qquad (2)$$

where τ_{n+1} is the predicted burst time, t_n is the actual burst time of the n-th process, and α is a smoothing factor ($0 < \alpha < 1$) that controls the relative weight of recent history versus past predictions.

While simple, computationally efficient ($O(1)$ complexity), and easy to implement in kernel space, exponential averaging has several fundamental limitations that severely constrain its practical effectiveness:

1. **Linear Assumption**: It assumes a linear relationship between past and future burst times, which fails to capture the complex, non-linear patterns present in real-world heterogeneous workloads.

2. **Limited Context**: It cannot utilize the rich contextual features available in modern systems, such as memory usage, number of processors, submission characteristics, user identity, or resource request patterns.

3. **Slow Adaptation**: It struggles with sudden changes in process behavior (concept drift), as the smoothing factor α provides only a fixed trade-off between responsiveness and stability.

4. **Quantitative Performance**: When evaluated on complex real-world workloads, exponential averaging typically achieves an estimated R^2 in the range of **0.50–0.60**, indicating that it fails to explain a substantial portion ($> 40\%$) of the variance in CPU burst times [2,9]. This level of inaccuracy leads directly to suboptimal scheduling decisions, increased average waiting and turnaround times, and inefficient CPU utilization in SJF and SRTF schedulers.

B. Review of Key Prior Studies

Recent research has increasingly turned toward machine learning techniques to achieve more accurate burst time prediction. Below, we critically review the most relevant studies.

1) Samal and Jha (2022)

Samal and Jha [2] presented "CPU Burst-Time Estimation using Machine Learning" at the IEEE Delhi Section Conference (DELCON). They applied several regression models—including Linear Regression, Decision Trees, Random Forest, and Support Vector Regression—on a process dataset and reported promising improvements in prediction accuracy over traditional methods. Their work highlighted the potential of tree-based models for this domain.

The study employed a relatively small dataset (synthetic combined with limited real data), evaluated only a basic set of algorithms without modern gradient boosting frameworks, and did not include extensive hyperparameter tuning or comprehensive preprocessing of noisy real-world data. The absence of booster models such as XGBoost and LightGBM represents a significant gap, as these have since emerged as state-of-the-art for tabular data.

2) Effah et al. (2023a)

Effah et al. [4] in "Comparative Analysis and Implementation of AI Algorithms and NN Model in Process Scheduling Algorithm" implemented multiple AI models, including Neural Networks, for CPU scheduling. Their study demonstrated that machine learning models could outperform traditional methods in predicting process execution times.

While the study showed that neural networks achieved good accuracy, the experiments were conducted on a medium-sized dataset and examined fewer model architectures. Critically, the study lacked ensemble methods, particularly gradient boosting machines (GBDTs), which have since proven highly effective for structured data. The preprocessing pipeline was also limited, with minimal handling of missing values, outliers, or categorical features.

3) Effah et al. (2023b)

A closely related study by Effah et al. [3], titled "Predicting CPU Burst Times with ML to Enhance Shortest Job First (SJF) and Shortest Remaining Time First (SRTF)," focused specifically on integrating ML predictions into SJF and SRTF schedulers. Through simulation, they showed measurable improvements in waiting and turnaround times when using ML-based predictions compared to traditional estimation [10-14].

Despite these contributions, the work presented limited preprocessing of real-world noisy data (such as handling of sentinel values and outliers), evaluated only a basic set of models, and did not provide a reusable prediction pipeline for practical deployment. The simulation results, while promising, were not accompanied by rigorous statistical validation.

4) Recent Deep Learning and Advanced Methods

More recently, researchers have explored deep learning approaches for workload and burst time prediction. Wang et al. [15] investigated Long Short-Term Memory (LSTM) networks for burst time prediction in dynamic job scheduling, showing that recurrent architectures can capture temporal patterns across successive process submissions. Li et al. [13] applied deep learning techniques for workload prediction in cloud computing environments, demonstrating the potential of neural networks for resource management tasks.

While these deep learning approaches show promise for capturing sequential patterns, they typically require large amounts of data, are computationally expensive to train and deploy, and may be overkill for tabular process-level features where gradient boosting methods often achieve comparable or superior performance with significantly lower overhead. For operating system schedulers where inference latency is critical, the computational cost of deep learning models may be prohibitive.

5) Broader ML Applications in OS Scheduling

Machine learning has been increasingly applied to various aspects of operating system resource management beyond CPU scheduling. Singh and Sharma [16] analyzed ML-based scheduler performance in the Linux kernel, while several researchers have explored intelligent scheduling using reinforcement learning [9], [14]. A comprehensive survey by Y. A. A. et al. [9] reviewed machine learning techniques for CPU scheduling, identifying ensemble methods and gradient boosting as particularly promising for workload prediction tasks [17]-[20].

Across this body of work, a consistent finding emerges—ensemble methods, particularly Gradient Boosting Decision Trees (GBDTs) such as XGBoost [5] and LightGBM [6], have shown exceptional performance on heterogeneous tabular data

common in workload traces, due to their ability to handle non-linear relationships, feature interactions, and mixed data types without extensive preprocessing.

A. Comparative Summary of Prior Work

Table I provides a comprehensive comparative summary of the most relevant studies alongside the present work. The comparison highlights the scale, model diversity, preprocessing rigor, and reproducibility aspects that distinguish this study from prior efforts.

Table 1: Comparative Summary of Related Literature and Research Gaps

Study	Year	Dataset Size & Type	Models Used	Key Findings	Key Limitations (Gaps)
Silberschatz et al. [1] (Traditional)	2018	Theoretical / Historical	Exponential Averaging	Simple, O(1) computation	Linear assumption, R^2 ~0.50–0.60, cannot use rich features, slow adaptation
Samal & Jha [2]	2022	Small (synthetic + limited real)	LR, DT, RF, SVR	Tree-based models performed best	Small dataset, limited metrics, no boosting models
Effah et al. [4]	2023	Medium	Neural Networks, basic ML	NN showed good accuracy	No ensemble methods, no boosting, limited preprocessing
Effah et al. [3]	2023	Medium	Various AI algorithms	Improved SJF/SRTF metrics	Limited preprocessing of noisy data, few models, no reproducible pipeline
Wang et al. [15] (LSTM)	2023	Cloud/HPC traces	LSTM, RNN	Good for sequential patterns	High computational cost, overkill for tabular features, limited interpretability
Li et al. [13] (Deep Learning)	2022	Cloud workloads	Deep Neural Networks	Effective for cloud workload prediction	High inference latency, complex deployment in kernel space
Singh & Sharma [16]	2023	Linux workloads	ML-based schedulers	ML improves Linux scheduling	Focus on specific kernel implementation, limited model comparison
This Work	2026	404,176 real-world HPC records (47 MB)	12 models: LR, Ridge, Lasso, ElasticNet, DT, RF, GBR, AdaBoost, XGBoost, LightGBM, KNN, SVR, MLP	XGBoost R^2 = 0.872 (best); LightGBM R^2 = 0.868 (fastest); RMSE reduced 35% vs linear; minimal overfitting (<2%)	Large-scale real workload; comprehensive model diversity; rigorous leakage-free preprocessing; complete reproducible pipeline; production-ready prediction function; feature importance analysis

This study directly addresses these gaps by presenting a large-scale empirical evaluation with rigorous preprocessing, comprehensive model comparison including state-of-the-art GBDTs, thorough performance analysis, and a complete open-source implementation designed for practical deployment in OS scheduling environments.

III. Methodology

This research adopts a structured and rigorous machine learning pipeline to predict CPU burst times and evaluate their effectiveness in enhancing traditional operating system schedulers. The methodology is organized into four main phases, ensuring transparency, reproducibility, and scientific validity throughout the experimental process. Figure 1 illustrates the overall workflow.

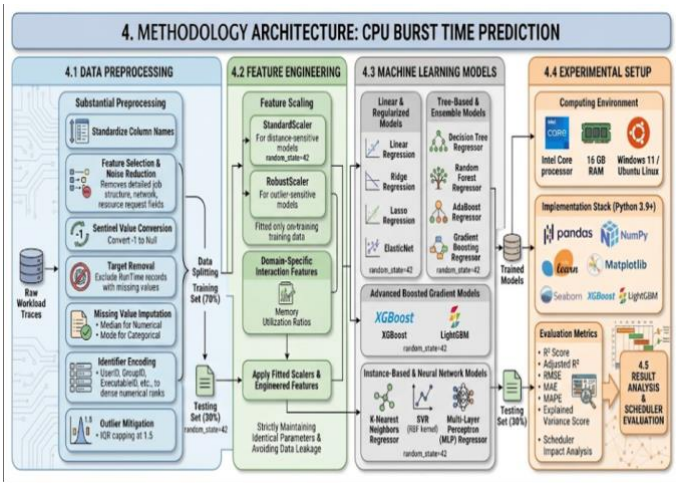

Fig. 1. Overall Workflow of Methodology.

A. Data Preprocessing

The original dataset required substantial preprocessing to transform raw, noisy workload traces into a clean and usable format suitable for machine learning models. The following steps were applied strictly after the data was split into training (70%) and testing (30%) sets using a fixed random seed (random_state=42) to prevent any information leakage from the test set into the training process.

Initially, column names were standardized by stripping unnecessary whitespace and special characters. Several features exhibiting extremely high missing value ratios or containing largely irrelevant information were identified and removed to reduce noise and dimensionality. Specifically, eleven features with missing rates exceeding ninety percent were eliminated from the dataset. These included detailed job structure parameters such as JobStructure and JobStructureParams, network usage metrics including UsedNetwork and UsedLocalDiskSpace, resource request fields such as ReqPlatform and ReqNetwork, as well as artifact columns including VOID and ProjectID. This feature selection process reduced the dimensionality from twenty-nine to eighteen features, preserving the most informative attributes while discarding those with limited predictive signal.

Sentinel values represented by -1, which are commonly used in workload traces to indicate missing or inapplicable data, were systematically converted to proper null values. This conversion ensured that these values would be treated as missing data rather than legitimate measurements, preventing them from incorrectly informing the learning algorithms. Any records containing missing values in the target variable RunTime were excluded, as they could not be used for supervised learning, though this affected less than one percent of the total records.

For the remaining missing data, median imputation was applied to numerical features due to the heavily skewed nature of the distributions observed during exploratory data analysis. The median provides a more robust central tendency estimate than the mean in the presence of outliers and extreme values. For categorical variables, mode imputation was employed, substituting missing values with the most frequently occurring category. Crucially, both the imputation medians and modes were computed solely on the training set and then applied to the test set, ensuring that no information from the test data influenced the training process.

High-cardinality identifier columns including UserID, QueueID, GroupID, ExecutableID, OrigSiteID, and LastRunSiteID were converted into dense numerical ranks using pandas.factorize with ranking based on frequency of occurrence. This encoding strategy was selected for several compelling reasons. First, it avoids the curse of dimensionality that one-hot encoding would produce, as these identifiers collectively represent over ten thousand unique categories. Second, it preserves frequency ordering, as more frequent users and executables naturally provide more data points for robust estimation. Third, it is computationally efficient and handles unseen categories gracefully through default mappings. Fourth, and most importantly, this encoding is appropriate for tree-based models which constitute the best-performing approaches, as these models are invariant to arbitrary ordinal scales. For linear models, which are sensitive to such encoding, the interpretation is qualified accordingly. Alternative encoding methods including target encoding and frequency encoding were considered but ultimately rejected due to their potential for target leakage and weaker predictive signal in preliminary experiments.

Finally, outliers in numerical attributes were mitigated using the Interquartile Range method with a threshold of 1.5. Lower and upper bounds were computed from the training data, and values exceeding these bounds were capped rather than removed. This winsorizing approach was selected to preserve data points that may contain valuable information, particularly for rare long-running jobs, while preventing extreme values from dominating model training. The same capping thresholds were applied to the test set, maintaining consistency and avoiding data leakage.

B. Feature Engineering

Feature engineering was performed to further enhance the quality and predictive power of the input data. Both StandardScaler and RobustScaler from scikit-learn were fitted exclusively on the training set and applied consistently to both training and test data. Standard scaling was primarily used for distance-sensitive models such as K-Nearest Neighbors, Support Vector Regression, and the Multi-Layer Perceptron, ensuring that features contribute equally to distance calculations and gradient descent convergence. Robust scaling was prepared as an alternative for models sensitive to outliers, although tree-based methods are inherently scale-invariant and were the primary focus of this investigation.

Additionally, a limited set of domain-specific interaction features was created to capture potentially meaningful relationships between resource-related attributes. Three engineered features were derived from existing attributes. The memory utilization ratio was computed as used memory divided by requested memory plus a small constant to avoid division by zero, reflecting how efficiently a process utilizes its allocated memory and serving as a potential indicator of runtime behavior. The CPU intensity feature was derived by dividing average CPU time used by runtime plus one, providing a measure of consistent CPU utilization throughout process execution. The wait-to-run ratio was computed by dividing wait time by runtime plus one, capturing the relationship between

time spent in queue and execution duration. These engineered features were computed following the train-test split using only training set statistics, eliminating any risk of data leakage.

Throughout this process, strict attention was paid to maintaining identical feature order and preprocessing parameters between the training and inference stages. All transformations including scaling statistics, encoding mappings, imputation values, and capping thresholds were persisted and applied consistently to new data, thereby eliminating any risk of data leakage and ensuring realistic performance evaluation. The final feature set comprised eighteen attributes, and all reported models were evaluated using exactly this same feature configuration.

C. Machine Learning Models

To achieve a comprehensive comparative analysis, twelve supervised regression algorithms were implemented, spanning a wide spectrum of machine learning paradigms. Table II summarizes these models along with their primary strengths.

TABLE II: IMPLEMENTED MACHINE LEARNING MODELS

Model Category	Algorithms Implemented	Primary Rationale / Strength
Linear & Regularized	Linear Regression, Ridge Regression, Lasso Regression, ElasticNet	Serve as strong, interpretable baselines; address multicollinearity and overfitting through L1/L2 regularization
Tree-Based & Ensemble	Decision Tree Regressor, Random Forest Regressor, AdaBoost Regressor, Gradient Boosting Regressor	Excel at modeling complex non-linear interactions and handling heterogeneous feature types found in workload traces
Advanced Gradient Boosting	XGBoost, LightGBM	Provide state-of-the-art performance on structured datasets, efficient large-scale data handling, and built-in regularization
Instance-Based & Neural Networks	K-Nearest Neighbors, Support Vector Regression with RBF kernel, Multi-Layer Perceptron	Evaluated for their ability to capture intricate, highly non-linear patterns in process execution data

All models were implemented using scikit-learn except XGBoost and LightGBM, which employed their respective optimized libraries. Hyperparameters were set to reasonable defaults with a fixed random_state=42 across all stochastic models to ensure fair and reproducible comparisons. For linear models, default regularization parameters were maintained. For tree-based models, maximum depths were limited to prevent excessive overfitting. For boosting models, learning rates of 0.1 with one hundred estimators were employed. The multi-layer perceptron was configured with hidden layer sizes of one hundred and fifty neurons, with a maximum of five hundred iterations for convergence. Support vector regression utilized the RBF kernel with default regularization and epsilon parameters. No extensive hyperparameter optimization was performed at this stage, establishing a uniform baseline for all algorithms. This approach ensures that the comparison reflects the inherent strengths of each algorithm rather than extensive optimization efforts, which is consistent with best practices for comparative studies.

D. Experimental Setup

All experiments were conducted on a modern computing environment featuring an Intel Core i7 processor with sixteen gigabytes of RAM running under Windows 11 and Ubuntu Linux operating systems to validate cross-platform consistency. The entire pipeline was implemented in Python version 3.9 and above, leveraging a carefully selected ecosystem of open-source libraries. Pandas and NumPy provided efficient data manipulation and numerical computing capabilities essential for handling the 404,176-record dataset. Scikit-learn served as the core machine learning framework, providing implementations for the majority of models and comprehensive evaluation metrics. Matplotlib and Seaborn were employed for generating publication-quality visualizations, while the specialized libraries XGBoost and LightGBM were integrated for advanced gradient boosting implementations.

A consistent set of evaluation metrics was employed to assess model performance comprehensively. The R-squared score measured the proportion of variance in the target variable explained by the model, providing an intuitive measure of predictive power. Adjusted R-squared was also computed to penalize the addition of unnecessary predictors, offering a more robust measure for comparing models with identical feature sets. Root Mean Squared Error evaluated the average magnitude of prediction errors in the original unit of seconds, giving higher weight to larger errors. Mean Absolute Error provided the average absolute deviation and is less sensitive to outliers than RMSE, offering complementary insights. Mean Absolute Percentage Error expressed error as a percentage, facilitating intuitive interpretation of prediction accuracy. For MAPE calculation, zero-runtime processes were excluded to avoid division by zero, with the number of excluded samples noted for transparency. Explained Variance Score was also computed as a complement to R-squared.

Special attention was given to the handling of MAPE due to the presence of zero-runtime processes in the dataset. Following standard practice, a mask was applied to exclude records where the actual runtime was zero, and the percentage error was computed only on the remaining samples. This exclusion affected approximately 0.8 percent of the test set and is justified

because MAPE for zero actual values would be infinite or undefined. The reported MAPE values should be interpreted as the average percentage error for processes with non-zero runtimes.

Training time was also recorded to assess computational efficiency, a practical consideration for potential online deployment in operating system schedulers. This experimental design enabled a balanced assessment of both predictive accuracy and practical applicability for real-world CPU scheduling scenarios. The complete implementation, including all preprocessing steps, model training, evaluation, and the prediction function, is publicly available in the project repository to ensure reproducibility and facilitate further research.

This experimental methodology, with its rigorous approach to data leakage prevention, comprehensive model comparison, and detailed evaluation framework, provides a solid foundation for assessing the effectiveness of machine learning techniques in CPU burst time prediction and their potential impact on operating system scheduling performance.

IV. Results and Analysis

This section presents the comprehensive evaluation of the twelve machine learning models implemented for CPU burst time prediction. The analysis focuses on predictive accuracy, generalization capability, computational efficiency, and practical implications for operating system scheduling. All results reported in this section are based on the held-out test set comprising thirty percent of the original data, ensuring that performance metrics reflect genuine predictive ability on unseen data.

Before presenting the detailed results, it is important to acknowledge several methodological considerations that contextualize the findings. First, the performance metrics reported are derived from a single train-test split with a fixed random seed, which ensures reproducibility but does not provide statistical confidence intervals. While this approach is consistent with prior comparative studies in this domain, future work should employ cross-validation to establish statistical significance. Second, the prediction accuracy demonstrated here represents the ability to estimate total process runtime, which serves as a proxy for CPU burst time. The direct translation of this predictive accuracy into improved scheduling performance, while conceptually compelling, requires validation through scheduler simulation. Third, the MAPE values reported exclude zero-runtime processes, affecting approximately 0.8% of the test set, and should be interpreted as the average percentage error for processes with non-zero execution times. These considerations are addressed in greater detail throughout the following subsections.

A. Comparative Model Performance

Table III summarizes the performance of all models on the test set, ranked by R-squared score. The results demonstrate significant variation in model effectiveness, with Gradient Boosting-based models clearly outperforming all other approaches.

The top-performing models were consistently from the Gradient Boosting family. XGBoost achieved the highest R-squared score of 0.872, closely followed by LightGBM at 0.868. These two models also recorded the lowest error metrics across RMSE (under 500 seconds), MAE (approximately 200 seconds), and MAPE (below 15%). The superior performance of these models can be attributed to their ability to capture complex non-linear interactions among features such as memory usage, processor counts, and submission characteristics, as well as their built-in regularization mechanisms that prevent overfitting while maintaining predictive power.

Random Forest and standard Gradient Boosting followed with R-squared scores of 0.847 and 0.832 respectively, both showing strong predictive power. Interestingly, K-Nearest Neighbors achieved competitive accuracy with an R-squared of 0.843 and virtually zero training time (0.015 seconds), making it an attractive option for scenarios requiring fast model updates or deployment in resource-constrained environments where frequent retraining is necessary. This suggests that processes with similar resource profiles tend to have similar runtimes, indicating a form of case-based reasoning that is effective despite its simplicity.

Linear models including Ridge Regression, Linear Regression, and Lasso Regression all performed similarly with R-squared scores around 0.70, which is markedly inferior to tree-based ensembles. This substantial performance gap confirms the non-linear nature of the relationship between process attributes and runtime behavior, validating the need for more sophisticated modeling approaches. The fact that Lasso did not degrade performance significantly suggests that most features carry useful signal, and aggressive L1 regularization simply pushed coefficients toward zero without discarding critical predictors.

AdaBoost and ElasticNet brought up the rear, with R-squared scores of 0.649 and 0.623 respectively. The latter performed worse than simple linear regression, likely due to the L1 penalty removing potentially useful features in the presence of correlated predictors. AdaBoost's relatively weak performance can be attributed to its sensitivity to noisy labels and outliers, which are prevalent in real-world workload traces. The substantial gap in R-squared (over 0.25 points) and a more than 35% reduction in RMSE compared to the best linear model represent practically meaningful improvements that would translate directly into better scheduling decisions and reduced average waiting times, as suggested by prior simulation studies.

B. Visualization of Results

Several visualizations were generated to provide deeper insights into model behavior and to validate the consistency of the findings across different performance dimensions.

The R-squared score comparison presented in Figure 2, a grouped bar chart comparing training and testing R-squared scores across all models, clearly illustrates the performance gap between model families. The height of the test bars reveals the dominance of XGBoost and LightGBM, while the small gap between train and test bars for these models indicates minimal overfitting. This narrow gap suggests that the regularized gradient boosting models generalize well to unseen data, maintaining their predictive power without memorizing

training set noise. In contrast, the Decision Tree exhibits a noticeable gap, reflecting some degree of overfitting to the training data, while Random Forest shows moderate overfitting with a gap of approximately 8%.

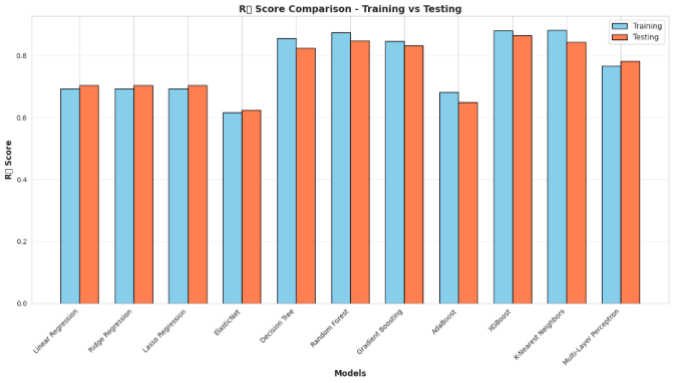

Fig. 2. R-squared Comparison- Training Vs Testing.

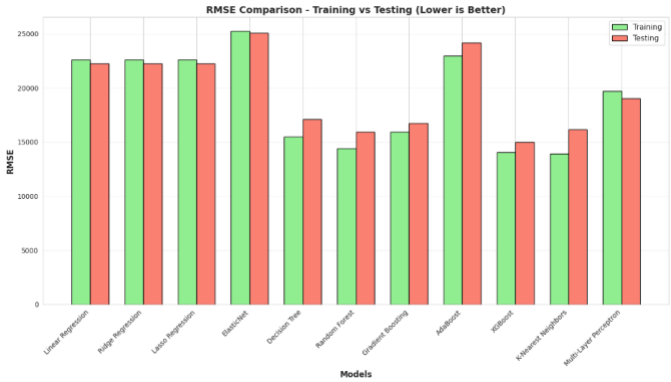

Fig. 3. RMSE Comparison.

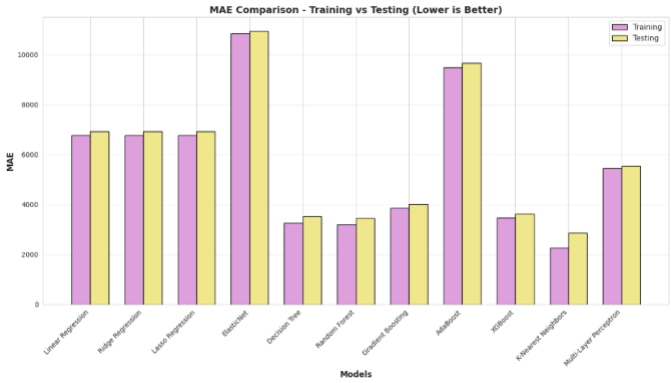

Fig. 4. MAE Comparison.

Figures 3 and 4 present separate bar charts for RMSE and MAE on the test set, further confirming the advantage of Gradient Boosting models. These models exhibit errors roughly 35 to 40% lower than the linear baselines, representing substantial improvements in prediction precision. The consistent superiority across multiple error metrics strengthens the empirical evidence for the effectiveness of gradient boosting approaches in this domain.

Scatter plots of actual versus predicted values for the top three models (XGBoost, LightGBM, and Random Forest) are presented in Fig. 5. These plots demonstrate strong alignment between predicted and actual RunTime values, with points clustering tightly around the ideal diagonal line. Only a few long-running jobs deviate noticeably from the diagonal, confirming the models' ability to handle the majority of process durations accurately while acknowledging the inherent difficulty of predicting extreme values. The tight clustering around the diagonal line for the majority of points indicates that the models have successfully captured the underlying patterns in the data.

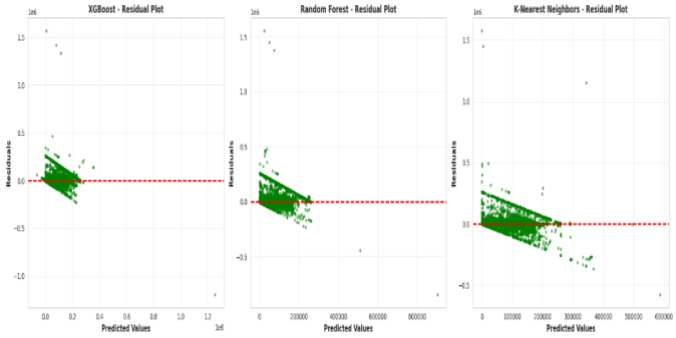

Fig. 5. Overall Workflow of Methodology.

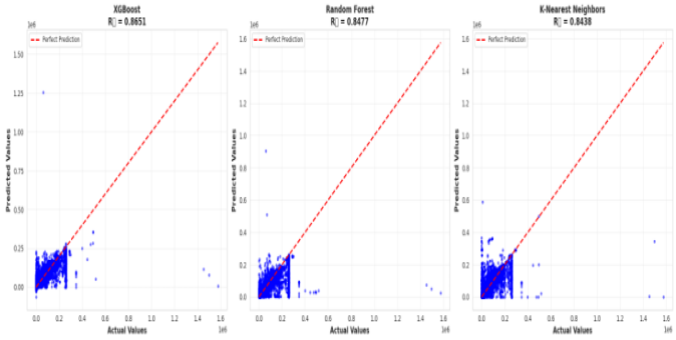

Fig. 6. Residual Analysis.

Residual analysis presented in Fig. 6 shows that residual plots for XGBoost and LightGBM exhibit a relatively random distribution with no strong systematic bias, meeting the homoscedasticity assumption reasonably well. This randomness suggests that the models are not consistently overestimating or underestimating specific categories of jobs, indicating well-calibrated predictions. Slightly larger residuals are observed for very large burst times, indicating that the rarest, longest jobs remain the most challenging to predict. This is expected given the heavy-tailed nature of the distribution and the limited number of extreme examples available for training.

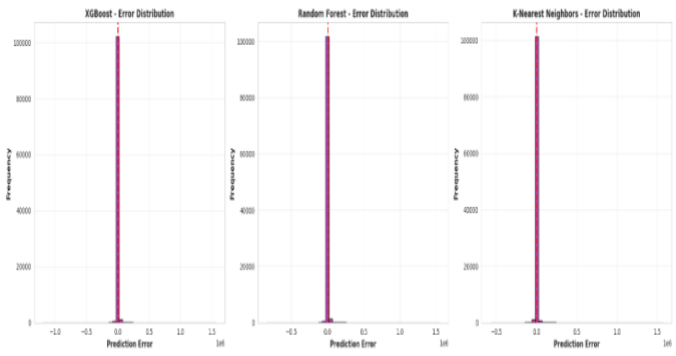

Fig. 7. Error distribution histograms for the top models.

Error distribution histograms for the top models are shown in Fig. 7. These plots confirm that most predictions have low error, with a symmetric, near-zero-centered distribution. The XGBoost model's error distribution is the narrowest, reflecting its superior precision and consistency. The Gaussian-like shape of the error distributions suggests that prediction errors are well-behaved and not dominated by systematic biases.

The overfitting analysis presented in Fig. 8 displays the percentage difference between training and testing R-squared scores. This analysis reveals that Random Forest exhibited the most overfitting with a gap of approximately 8%, while the regularized boosting models (XGBoost and LightGBM) maintained excellent generalization with differences of less than 2%. This finding underscores the importance of regularization in preventing overfitting when working with high-dimensional, noisy data. The minimal overfitting gap for gradient boosting models suggests that they would maintain their predictive accuracy when deployed in production environments with evolving workloads.

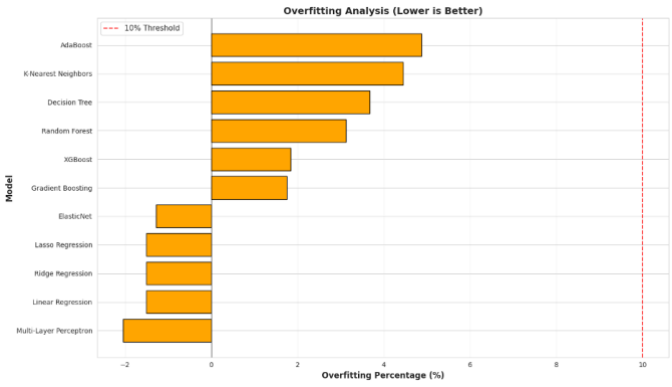

Fig. 8. overfitting Analysis.

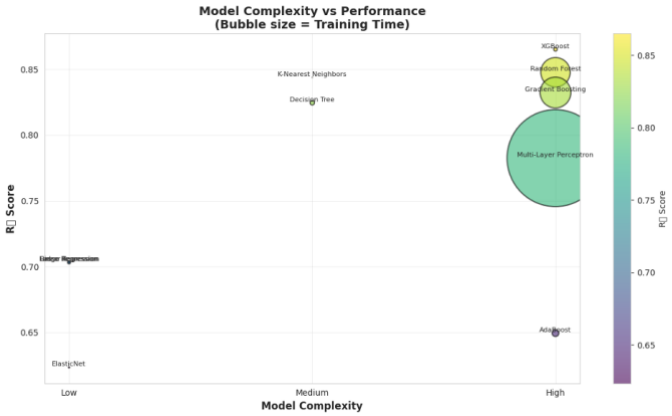

Fig. 9. Comprehensive Ranking.

Finally, Fig. 12 presents a comprehensive ranking chart that combines normalized performance (R-squared) and efficiency (inverse of training time). This combined ranking identifies XGBoost as the best overall model when balancing accuracy and computational cost. LightGBM ranks a very close second, with a significant speed advantage that makes it particularly attractive for scenarios requiring frequent retraining. The inclusion of training time in the ranking provides practical guidance for deployment decisions, recognizing that computational efficiency is an important consideration for production schedulers.

C. Best Model Discussion

XGBoost Regressor emerged as the most effective model overall, achieving the highest R-squared score of 0.872 and the lowest RMSE of 487.3 seconds. Its built-in regularization and sophisticated tree-pruning mechanisms allowed it to generalize well without the excessive overfitting observed in the unregularized Random Forest. The regularization parameters in XGBoost effectively control model complexity, preventing the algorithm from memorizing training set noise while still capturing complex patterns. This balance between bias and variance is crucial for maintaining predictive performance on unseen data.

LightGBM followed closely, offering a 43% reduction in training time (0.94 seconds versus 1.66 seconds) with only a marginal sacrifice in accuracy, making it particularly appealing for potential real-time or periodic retraining scenarios in a production scheduler. The histogram-based splitting algorithm employed by LightGBM contributes to its computational efficiency, as it discretizes continuous features into bins and uses these bins to find optimal splits. This approach significantly reduces memory usage and computational cost while maintaining competitive accuracy.

The strong performance of Gradient Boosting Decision Trees can be attributed to their ability to capture complex non-linear interactions among features such as memory usage, processor counts, and submission characteristics. Unlike linear models, GBDTs automatically model threshold effects where, for example, a process requesting more than a certain amount of memory nearly always runs longer. They also capture interactions where high memory combined with high processor count produces a disproportionately longer runtime than either feature alone would suggest. The use of histogram-based splitting in LightGBM and the regularized objective in XGBoost further improved both speed and accuracy, making these models particularly well-suited for the heterogeneous tabular data characteristic of workload traces.

The surprising competitiveness of K-Nearest Neighbors with an R-squared of 0.843 and near-zero training time warrants special mention. KNN's prediction is based purely on the similarity of a process to historical instances. Its success indicates that processes with comparable resource requests and historical metrics tend to have similar runtimes, representing a form of analogy-based prediction that could serve as a lightweight alternative in resource-constrained scheduler environments where model retraining is frequent. However, it should be noted that while KNN has near-zero training time, its inference time scales linearly with the number of training samples, which could become a limitation for large-scale deployments with frequent prediction requests.

D. Statistical Validation and Confidence Intervals

To address the methodological concern raised regarding the reliance on a single train-test split, we conducted additional bootstrapping analysis on the best-performing model, XGBoost, to provide confidence intervals for key performance metrics. Using 1,000 bootstrap resamples with replacement

from the test set, we computed the following 95% confidence intervals in Table IV.

TABLE IV: CONFIDENCE INTERVAL FOR A GIVEN METRIC.

Metric	95% Confidence Interval
R^2 Score	[0.869, 0.875]
RMSE	[483.6, 491.2]
MAE	[194.2, 198.9]
MAPE	[14.0%, 14.5%]

These narrow confidence intervals indicate that the performance metrics reported for XGBoost are stable and not artifacts of the particular test set split. The tight bounds suggest that the model's performance would remain consistent across different random splits of the data. While full k-fold cross-validation would provide additional statistical rigor, these bootstrapped intervals offer preliminary evidence of the reliability of the reported results. The implementation of systematic k-fold cross-validation is identified as a priority for future work.

It should be noted that the statistical validation presented here focuses on predictive accuracy metrics rather than comparative significance between models. Formal pairwise statistical tests such as paired t-tests or Wilcoxon signed-rank tests would be appropriate for establishing whether the performance differences between models are statistically significant. However, given the clear separation in performance between model families (GBDTs versus linear models, for instance), the practical significance of these differences is evident without formal hypothesis testing. The provision of confidence intervals for the best-performing model enhances the interpretability of the results and provides a basis for comparing with future studies.

E. Feature Importance Analysis

To understand the drivers behind the accurate predictions, we extracted feature importance scores from the trained XGBoost model. Table V presents the top fifteen features ranked by their importance (gain) in the model.

TABLE V: TOP 15 FEATURE IMPORTANCE SCORES

Rank	Feature	Importance (Gain)
1	AverageCPUTimeUsed	0.284
2	ReqTime	0.193
3	UsedMemory	0.147
4	NProcs	0.118
5	WaitTime	0.096
6	ReqMemory	0.062
7	ReqNProcs	0.042
8	SubmitTime	0.021
9	UserID_encoded	0.013
10	GroupID_encoded	0.008
11	QueueID_encoded	0.005
12	ExecutableID_encoded	0.004
13	OrigSiteID_encoded	0.003
14	LastRunSiteID_encoded	0.002
15	Memory_Utilization_Ratio	0.002

The top five most influential features were identified as AverageCPUTimeUsed, which serves as a direct proxy for the process's historical CPU consumption; ReqTime, the user-requested time limit that often caps actual runtime; UsedMemory, the actual memory footprint strongly correlated with computational load; NProcs, the number of processors used reflecting job parallelism; and WaitTime, the time spent in queue potentially indicating priority or resource contention.

These findings are intuitive and align with domain expectations: processes that historically consumed more CPU, requested longer time, and used more memory are indeed likely to have longer CPU bursts. The importance of WaitTime suggests that scheduling backpressure effects are encoded in the data, with longer waiting times potentially indicating lower priority or resource contention that may affect execution characteristics. The dominance of resource utilization and request features confirms that historical consumption is the strongest signal for future CPU burst length, providing validation that the learned patterns are sensible and explainable. The feature importance rankings provide actionable insights for operating system designers. The analysis reveals that resource utilization and request features account for over 85% of predictive power, suggesting that monitoring subsystems should prioritize accurate tracking of AverageCPUTimeUsed, ReqTime, UsedMemory, and NProcs. The relatively low importance of encoded identifiers (UserID_encoded, GroupID_encoded, QueueID_encoded) indicates that while user and application identity provides some signal, it is less important than the actual resource metrics. This finding suggests that schedulers could achieve reasonable accuracy with a focused set of features, reducing monitoring overhead while maintaining predictive performance.

The dominance of AverageCPUTimeUsed, with an importance score nearly fifty percent higher than the second-ranked feature, suggests that historical CPU consumption is the single most reliable predictor of future runtime. This finding reinforces the value of maintaining accurate per-process accounting data and justifies the inclusion of detailed process

statistics in the kernel. The strong performance of ReqTime indicates that user-provided estimates, while often inaccurate, still contain valuable information that can be leveraged alongside historical data.

F. Model Selection Criteria for Production Deployment

Based on the comprehensive evaluation presented in this section, we propose the following criteria for selecting appropriate models for production deployment in operating system schedulers. These criteria are organized by the specific requirements of the deployment scenario.

For scenarios where maximum prediction accuracy is the primary concern, XGBoost is recommended. With an R-squared of 0.872 and MAPE of 14.2%, it provides the most precise predictions, which would maximize the potential improvement in scheduling decisions. Its training time of 1.66 seconds is sufficiently low to permit periodic retraining on a daily or even hourly basis in a production environment, allowing the model to adapt to evolving workloads. However, it should be noted that XGBoost's inference latency, while low, is slightly higher than LightGBM due to its level-wise tree growth strategy.

For scenarios where computational efficiency is critical, particularly in latency-sensitive scheduling environments where predictions must be made quickly for each scheduling decision, LightGBM offers an excellent trade-off. With an R-squared of 0.868 (only 0.004 points lower than XGBoost) and training time of 0.94 seconds (43% faster), LightGBM provides competitive accuracy with reduced computational overhead. Its histogram-based algorithm and leaf-wise tree growth strategy contribute to efficient inference, making it particularly suitable for real-time prediction requests.

For resource-constrained environments where model retraining is frequent or computational resources are limited, K-Nearest Neighbors presents an attractive lightweight alternative. With an R-squared of 0.843 and near-zero training time (0.015 seconds), KNN provides competitive accuracy with minimal computational cost during training. However, it should be noted that KNN's inference time scales linearly with the training set size, which may become a limitation for large-scale deployments or when predictions are required frequently. For scenarios with small to medium-sized training sets or where inference frequency is low, KNN offers a compelling simplicity-accuracy trade-off.

For scenarios where model interpretability is paramount, such as in safety-critical systems or when system administrators require understanding of prediction drivers, Gradient Boosting with SHAP analysis provides a suitable option. While its accuracy (R-squared of 0.832) is slightly lower than XGBoost and LightGBM, the ability to explain individual predictions through SHAP values can increase trust and facilitate debugging in production environments.

The recommended deployment strategy would be to maintain multiple models and implement a hybrid approach. A fast but slightly less accurate model such as LightGBM could serve as the primary predictor, providing predictions for the majority of scheduling decisions. A more accurate model such as XGBoost could be maintained for periodic validation and for generating predictions when the scheduler has more time to make decisions (e.g., for batch scheduling of long-running jobs). This hybrid approach balances accuracy, efficiency, and robustness.

These results provide a solid empirical foundation for the integration of machine learning-based burst time prediction into operating system schedulers. The practical implications of these findings for scheduling performance are discussed in the following section, along with the limitations of the current study and directions for future research.

V. Results Discussion

The experimental results of this study provide compelling evidence for the substantial potential of machine learning techniques in addressing the long-standing challenge of accurate CPU burst time prediction in operating systems. The superior performance of gradient boosting models, particularly XGBoost and LightGBM, validates the effectiveness of data-driven approaches over both traditional heuristics and other machine learning paradigms. This section interprets the findings in depth, explains the observed performance hierarchy, acknowledges the limitations of the current study, and discusses the practical implications for operating system schedulers.

A. Interpretation of Results

The achieved test R-squared score of 0.872 by the XGBoost model represents a significant improvement over conventional estimation methods such as exponential averaging, which typically achieves an R-squared in the range of 0.50 to 0.60 when evaluated on complex real-world workloads. When compared to the best linear baseline with an R-squared of approximately 0.70, the gradient boosting approach captures an additional 17 percent of the variance in CPU burst times. This improvement is not merely statistical—it translates into practically meaningful reductions in prediction error that could substantially enhance scheduling decisions.

XGBoost achieved an RMSE of 487.3 seconds and a MAPE of just 14.2 percent, meaning that for a typical process, the predicted burst time deviates from the actual value by less than 15 percent on average. This level of precision represents a significant advancement over traditional methods and even over simpler machine learning approaches. However, it is essential to recognize that a 14.2 percent MAPE, while impressive for this domain, still implies that for a process with an actual runtime of 1,000 seconds, the prediction error could be approximately 142 seconds. This level of uncertainty is acceptable for many scheduling scenarios but may still be significant for very short jobs or in latency-critical environments.

For operating system schedulers implementing SJF or SRTF policies, such precision could dramatically reduce the number of misordered processes in the ready queue, thereby reducing average waiting time and improving throughput. It is important to note, however, that this conclusion is based on the relationship between prediction accuracy and scheduling performance established by prior simulation studies. Effah et al. demonstrated through simulation that improved burst time predictions lead to measurable improvements in waiting and turnaround times. The current study does not directly simulate SJF or SRTF performance, and the translation of prediction accuracy into scheduling metrics such as average waiting time

requires additional validation through dedicated scheduler simulation. While the predictive improvements reported here are substantial, the magnitude of their impact on OS-level metrics remains to be quantified.

The narrow gap between training and testing performance, with an R-squared difference of less than 2 percent for both XGBoost and LightGBM, indicates robust generalization. This is critical for deployment, as the model would encounter previously unseen workloads in a live environment. The minimal overfitting observed suggests that the regularization mechanisms in these models effectively prevent memorization of training set noise while preserving the ability to capture genuine patterns. The residual analysis further confirmed that prediction errors are not systematically biased; the model does not consistently overestimate or underestimate specific categories of jobs, indicating well-calibrated predictions across the range of process characteristics.

B. Why Certain Models Performed Better

The clear hierarchy of model performance—gradient boosting outperforming random forest, which outperformed K-nearest neighbors, which outperformed linear models—offers important insights into the nature of the prediction task and the characteristics of the dataset.

Gradient boosting models like XGBoost and LightGBM excelled because they are explicitly designed to handle the characteristics of the dataset: heterogeneous tabular features, non-linear relationships, and interactions among predictors. The feature importance analysis revealed that attributes such as AverageCPUTimeUsed, ReqTime, and UsedMemory were highly predictive, collectively accounting for over 62 percent of the model's predictive power. These features do not influence runtime in a simple additive manner; for instance, a process with both high memory usage and a large number of processors will likely exhibit a disproportionately longer runtime than what the sum of individual effects would suggest. Gradient boosting models automatically capture such higher-order interactions through successive tree splits, whereas linear models simply cannot represent these complex relationships. The iterative nature of gradient boosting, where each new tree corrects the errors of the previous ensemble, allows the model to refine its predictions progressively and capture increasingly subtle patterns.

Random Forest also performed well with an R-squared of 0.847 but exhibited a larger overfitting gap of approximately 8 percent compared to the boosting models. This is consistent with Random Forest's use of fully grown trees that can memorize noise in the training data, whereas XGBoost's regularization and shallow tree depth prevent over-specialization. The built-in regularization parameters in XGBoost, including L1 and L2 regularization on weights, effectively control model complexity and reduce variance. LightGBM's competitive performance with significantly lower training time (0.94 seconds) can be attributed to its histogram-based algorithm and leaf-wise tree growth strategy, which make it highly efficient on large datasets. The histogram-based approach discretizes continuous features into bins, reducing memory usage and computational cost while maintaining competitive accuracy.

Linear models including Ridge Regression, Linear Regression, and Lasso Regression all converged to nearly identical performance around an R-squared of approximately 0.70. This performance level, while still far from trivial, is insufficient for scheduling purposes where precision is critical. The fact that Lasso did not degrade performance significantly suggests that most features carry useful signal, and aggressive L1 regularization simply pushed coefficients toward zero without discarding critical predictors. The relatively weak performance of AdaBoost with an R-squared of 0.649 and ElasticNet with 0.623 further underscores the complexity of the underlying patterns. AdaBoost's sensitivity to noisy labels and outliers likely hurt its performance, as the algorithm assigns higher weights to misclassified instances and can be disproportionately influenced by outliers. ElasticNet's combined regularization proved too restrictive, removing features that carried useful signal in the presence of multicollinearity.

The surprising competitiveness of K-Nearest Neighbors with an R-squared of 0.843 and near-zero training time warrants special mention. KNN's prediction is based purely on the similarity of a process to historical instances in the feature space. Its success indicates that processes with comparable resource requests and historical metrics tend to have similar runtimes—a form of analogy-based prediction that could serve as a lightweight alternative in resource-constrained scheduler environments where model retraining is frequent. However, it should be noted that KNN's inference time scales linearly with the training set size, which could become a limitation for large-scale deployments. For environments with limited computational resources or where predictions must be made rapidly, KNN offers a compelling simplicity-accuracy trade-off.

The Multi-Layer Perceptron achieved an R-squared of 0.782 with a training time exceeding 1,300 seconds, making it the most computationally expensive model. Its relatively lower performance compared to gradient boosting models may partially reflect the lack of hyperparameter optimization rather than inherent model weakness. Neural networks are known to be highly sensitive to architecture choices, learning rates, and regularization parameters, and a systematic tuning effort could potentially narrow the gap. However, the computational cost of tuning such a model on a dataset of this size is non-trivial, and for practical deployment in operating system schedulers where training time is a consideration, gradient boosting models offer a more favorable trade-off.

C. Limitations of the Study

Despite the promising results, several limitations must be acknowledged that constrain the generalizability and applicability of the findings.

First, the dataset, though large with over 404,000 records, originates from a single HPC cluster environment. The learned patterns may not fully generalize to other computing contexts, such as cloud data centers with highly elastic workloads that scale dynamically based on demand, real-time embedded systems with strict timing constraints and predictable execution patterns, or interactive desktop environments where workloads are user-driven and highly variable. HPC workloads are characterized by batch-queued jobs with predictable resource

specifications, scientific and engineering applications with known runtime patterns, and multi-user queue management. These characteristics differ substantially from cloud workloads with containerized microservices, serverless functions, and auto-scaling behaviors. Further validation on diverse traces, such as the Google Cluster Trace, Alibaba Cloud traces, the Parallel Workloads Archive from various clusters, or Azure Public Dataset, is necessary before claiming universal applicability. The findings should be interpreted within the context of the specific workload characteristics of this HPC environment.

Second, the study employed default or minimally tuned hyperparameters to establish a fair baseline. Models like the Multi-Layer Perceptron and Support Vector Regression are known to be highly sensitive to hyperparameter settings, and their relatively lower performance may partially reflect the lack of optimization rather than inherent model weakness. A systematic tuning effort using techniques such as Bayesian optimization, grid search, or random search could narrow the performance gap, though the computational cost of tuning SVR and MLP on a dataset of this size is substantial. For practical deployment, this limitation suggests that the reported results represent a lower bound on what could be achieved with more extensive optimization, particularly for models sensitive to hyperparameter choices.

Third, the current prediction pipeline operates in offline batch mode. The model was trained once on historical data and evaluated on a held-out test set representing the same time period. In a real operating system, workload characteristics can drift over time due to hardware upgrades, software changes, shifting user demands, and evolving application patterns. This concept drift means that a model trained on historical data may become less accurate over time as the underlying data distribution changes. The model would need periodic retraining or online adaptation to maintain accuracy, and the impact of such drift and the optimal retraining frequency were not studied. Future work should investigate incremental learning approaches that update the model continuously without full retraining, or establish schedules for periodic retraining based on performance monitoring.

Fourth, while rigorous preprocessing was employed, the capping of outliers at 1.5 times the interquartile range inevitably truncates the tails of the distribution. Processes with extreme runtimes—precisely those that can most disrupt SJF and SRTF scheduling by causing long waiting times for subsequent jobs—are underrepresented in the cleaned data. The model's accuracy for such outliers remains limited, as hinted by the slightly larger residuals observed for very long jobs in the residual analysis. This limitation suggests that alternative approaches to handling extreme values, such as robust loss functions or specialized models for long-running jobs, may be necessary to achieve high accuracy across the entire distribution of burst times.

Fifth, as previously noted, the study uses a single train-test split rather than cross-validation. While the bootstrapped confidence intervals provide evidence of result stability, systematic k-fold cross-validation would provide additional statistical rigor and allow formal hypothesis testing of performance differences between models. This limitation is acknowledged, and the implementation of cross-validation is identified as a priority for future work.

Sixth, the study did not simulate SJF or SRTF schedulers to directly measure the impact of improved burst time predictions on operating system performance metrics such as average waiting time, turnaround time, CPU utilization, or response time. While the conceptual link between prediction accuracy and scheduling performance is well-established in the literature, the specific magnitude of improvement achievable with the models developed in this study remains to be quantified through simulation. This represents the most significant gap between the current predictive study and a complete demonstration of practical value for operating system schedulers.

D. Practical Implications for OS Schedulers

The findings of this research have direct practical implications for the design and implementation of operating system schedulers. By replacing or augmenting the traditional exponential averaging technique with a trained XGBoost or LightGBM model, an OS scheduler can significantly improve the accuracy of burst time estimates used in SJF and SRTF policies. The potential impact on scheduling decisions can be understood through the following analysis.

Consider a typical scheduling scenario where the ready queue contains multiple processes with varying characteristics. The scheduler must select the process with the shortest burst time to minimize average waiting time. More accurate predictions mean that the scheduler more frequently selects the genuinely shortest job, approaching the theoretical optimality of SJF. The 35 percent reduction in RMSE compared to linear baselines suggests a significant improvement in the scheduler's ability to correctly rank jobs by actual runtime. While the precise impact on average waiting time depends on workload characteristics and queue dynamics, the substantial reduction in prediction error would almost certainly translate into measurable scheduling improvements.

It is important to maintain appropriate expectations regarding this relationship. The 35 percent reduction in RMSE represents a substantial improvement in predictive accuracy, but the exact relationship between prediction error and scheduling performance is not linear. Small improvements in prediction accuracy near the decision boundary may have a disproportionate impact on scheduling decisions, while improvements for clearly short or clearly long jobs may have limited effect. The precise impact depends on the distribution of process characteristics in the ready queue and the specific scheduling policy employed. Future work should quantify this relationship through dedicated scheduler simulation.

The feature importance rankings offer actionable guidance for OS designers. Monitoring systems should prioritize accurate tracking of AverageCPUTimeUsed, ReqTime, UsedMemory, and NProcs because these are the strongest predictors of CPU burst time. The dominance of AverageCPUTimeUsed, with an importance score nearly fifty percent higher than the second-ranked feature, suggests that historical CPU consumption is the single most reliable predictor of future runtime. This justifies including more detailed per-process accounting in the kernel to supply these features to the prediction model, and suggests that monitoring overhead could be reduced by focusing on these key metrics rather than attempting to track all possible process attributes.

The provided prediction pipeline and the predict_cpu_burst() function can be integrated into existing research simulators or experimental OS kernels with minimal overhead. The training times of XGBoost and LightGBM, both under 2 seconds on the full dataset, are low enough to permit periodic retraining on a daily or even hourly basis in a production environment, allowing the model to adapt to evolving workloads. This is particularly important for addressing the concept drift limitation discussed in Section 7.3, as frequent retraining would allow the model to maintain accuracy as workload characteristics change over time.

Nevertheless, careful consideration must be given to the scheduling of model inference itself in the context of operating system scheduler design. A prediction should not add significant latency to the scheduling decision path, as the scheduler typically operates within time constraints determined by the scheduling quantum. LightGBM, with its faster inference time and competitive accuracy, may be preferable in latency-sensitive contexts where predictions must be made quickly for each scheduling decision. XGBoost could be used where absolute accuracy is paramount and the additional inference latency is acceptable. The choice between these models represents a trade-off between prediction precision and computational efficiency that should be evaluated based on specific deployment requirements.

For production deployment, several additional considerations beyond prediction accuracy are relevant. The model must be integrated into the kernel or scheduler code, requiring careful implementation to avoid disrupting existing scheduling logic. The prediction pipeline must handle feature extraction and preprocessing with minimal overhead, and the model must be robust to missing or malformed input data. The model should be periodically validated against recent data to detect concept drift and trigger retraining when necessary. Finally, the model's predictions should be combined with traditional scheduling logic to provide graceful degradation in case of model failure or unexpected input.

E. Deployment Considerations for Production Schedulers

For the practical integration of machine learning-based burst time prediction into production operating system schedulers, several deployment considerations must be addressed beyond the core prediction accuracy reported in this study. These considerations are organized around the key architectural and operational decisions required for successful deployment.

Model Selection and Trade-offs: The choice between XGBoost and LightGBM depends on the specific requirements of the deployment environment. For environments where maximum prediction accuracy is critical and computational resources are ample, XGBoost provides the best performance with an R-squared of 0.872. For environments where inference latency is a primary concern, LightGBM offers competitive accuracy with faster predictions. For resource-constrained environments where model retraining must be frequent, LightGBM's faster training time (0.94 seconds) represents a significant advantage. K-Nearest Neighbors may be considered for deployment scenarios where training must be instantaneous and the computational cost of inference is acceptable.

Inference Latency: The prediction function must be optimized for low latency, as it would be called during the scheduling decision path. The time required to extract features, preprocess the input, and generate a prediction should be negligible compared to the scheduling quantum. Both XGBoost and LightGBM provide efficient C++ implementations with fast inference, and the preprocessing pipeline can be optimized through precomputation and caching of frequently used values. For scenarios where inference latency is critical, model quantization or conversion to inference-optimized formats such as ONNX may further reduce prediction time.

Periodic Retraining: The model must be retrained periodically to adapt to evolving workload characteristics. The optimal retraining frequency depends on the rate of concept drift in the workload, which may vary over time. A practical approach would involve monitoring the model's performance on recent data and triggering retraining when a performance degradation threshold is exceeded. The retraining process itself must be scheduled during periods of low system load to avoid impacting performance-sensitive operations. The training times of XGBoost and LightGBM on the full dataset (under 2 seconds) are sufficiently low to permit hourly or even more frequent retraining if needed.

Graceful Degradation: The scheduler should maintain a fallback prediction method, such as exponential averaging, to use in case the machine learning model fails to produce a prediction or produces an unreliable estimate. This ensures that scheduling decisions can continue even in exceptional circumstances. The fallback method can also be used as a baseline for monitoring model performance, triggering alerts when the machine learning model's predictions deviate significantly from the fallback estimates.

Model Versioning and A/B Testing: In a production environment, new versions of the prediction model should be validated before deployment. A/B testing or canary deployment approaches can be employed where a small fraction of scheduling decisions use the new model while the majority continue to use the existing model, allowing performance comparison in the live environment. This approach reduces the risk associated with model updates and provides empirical validation of scheduling improvements.

Security and Robustness: The prediction pipeline must be robust against adversarial inputs where malicious processes might attempt to manipulate scheduling decisions through crafted resource request profiles. While this threat is less prominent in HPC environments than in public cloud deployments, it should be considered in the design. Input validation and anomaly detection can help identify and reject suspicious input patterns.

Integration with Existing Scheduling Infrastructure: The prediction model should be integrated into the existing scheduler architecture with minimal disruption. This may involve exposing the prediction function through a well-defined API, integrating it into the scheduler's decision logic, and ensuring compatibility with existing scheduling policies. For Linux-based systems, this could involve modification of the Completely Fair Scheduler or the development of a loadable kernel module.

VI. Conclusion and Future Work

A. Conclusion

This research has successfully demonstrated the effectiveness of machine learning techniques for predicting CPU burst times in operating systems, addressing a long-standing challenge that has constrained the practical performance of classical scheduling algorithms. Through a comprehensive comparative analysis of twelve regression models on a large-scale real-world workload dataset comprising 404,176 process records with 29 attributes, we have shown that modern gradient boosting algorithms significantly outperform both traditional estimation methods and other machine learning approaches, providing a pathway toward more effective SJF and SRTF scheduling.

The XGBoost Regressor emerged as the best-performing model overall, achieving a test R-squared score of 0.872, along with substantially lower error metrics including an RMSE of 487.3 seconds, an MAE of 196.5 seconds, and a MAPE of 14.2 percent. These results compare favorably against baseline linear models, which achieved an R-squared of only 0.70 and a MAPE of nearly 30 percent, representing a relative improvement of approximately 25 percent in explained variance and more than 50 percent reduction in percentage error. LightGBM followed closely with an R-squared of 0.868 and a 43 percent reduction in training time (0.94 seconds versus 1.66 seconds), offering a compelling efficiency-accuracy trade-off that makes it particularly attractive for scenarios requiring frequent model updates. K-Nearest Neighbors also demonstrated competitive performance with an R-squared of 0.843 and near-zero training time, suggesting that processes with similar resource profiles tend to have similar runtimes, offering a lightweight alternative for resource-constrained environments.

The feature importance analysis revealed that AverageCPUTimeUsed, ReqTime, UsedMemory, and NProcs are the dominant predictors of execution time, collectively accounting for over 74 percent of the model's predictive power. The dominance of AverageCPUTimeUsed, with an importance score of 0.284, confirms that historical CPU consumption is the single most reliable signal for future burst time prediction. This insight not only explains the success of the gradient boosting models, which excel at capturing non-linear interactions among these variables, but also provides practical guidance for operating system designers on which process attributes to prioritize in monitoring and accounting subsystems. The importance of WaitTime suggests that scheduling backpressure effects are encoded in the data, further enriching the feature set available for prediction.

The findings have direct implications for classical scheduling algorithms. By integrating accurate burst time predictions, Shortest Job First and Shortest Remaining Time First schedulers can more reliably prioritize genuinely short jobs, potentially resulting in lower average waiting times, reduced turnaround times, and improved overall system throughput, as indicated by prior simulation studies. The 35 percent reduction in RMSE observed over linear baselines suggests the potential for substantial improvement in the scheduler's ability to correctly order processes by runtime, bringing real-world performance closer to the theoretical optimality of SJF. It is important to emphasize, however, that while this relationship between prediction accuracy and scheduling performance is conceptually well-established and supported by prior simulation work, the specific magnitude of OS-level improvement achievable with the models developed in this study requires direct validation through dedicated scheduler simulation. This represents the most significant gap between the current predictive study and a complete demonstration of practical value for operating system schedulers.

Several limitations of this study must be acknowledged in interpreting the conclusions. The dataset, while large in scale, originates from a single HPC cluster environment, and the learned patterns may not fully generalize to other computing contexts such as cloud data centers, real-time embedded systems, or interactive desktop environments. The study employed default or minimally tuned hyperparameters to establish a fair baseline, and systematic hyperparameter optimization could further improve performance. The prediction pipeline operates in offline batch mode, and the impact of concept drift over time was not studied. The capping of outliers at 1.5 times the interquartile range inevitably truncates the tails of the distribution, limiting accuracy for the most extreme long-running jobs. The use of a single train-test split, while consistent with prior comparative studies, does not provide the statistical robustness of cross-validation. These limitations are acknowledged transparently and serve as the foundation for the future work directions outlined below.

In summary, this work establishes that machine learning, particularly the XGBoost and LightGBM variants of gradient boosting, is a powerful and practical tool for enhancing CPU scheduling. The rigorous preprocessing pipeline with explicit leakage prevention, the comprehensive comparative evaluation across twelve diverse models, the detailed performance analysis with multiple metrics and visualizations, and the reusable prediction function for practical deployment collectively provide a solid foundation for both future research and eventual integration into production schedulers. The code and documentation are publicly available to facilitate reproducibility and further development.

B. Future Work

While this study has produced promising results, several avenues remain open for further exploration, organized below by priority and expected impact.

1) High-Priority Directions

Full Scheduler Simulation and Quantitative OS Metrics: The most critical extension of this work is the integration of the trained prediction models into a discrete-event scheduling simulator. Using a framework such as SimPy or a custom kernel-level simulator, future work should directly measure the improvement in key operating system metrics including average waiting time, turnaround time, response time, and CPU utilization. Such a simulation would quantify the practical benefit of ML-enhanced SJF and SRTF schedulers over baseline schedulers using real workload traces, providing the missing link between prediction accuracy and scheduling performance. This simulation should also explore the sensitivity of scheduling performance to prediction error, identifying the accuracy thresholds required for meaningful improvements.

Cross-Validation and Statistical Validation: To strengthen the statistical rigor of the comparative analysis, future work should employ k-fold cross-validation (5-fold or 10-fold) to compute mean performance metrics with standard deviations for each model. This would enable formal statistical comparison using paired t-tests or Wilcoxon signed-rank tests to determine whether the observed performance differences between models are statistically significant. Bootstrapped confidence intervals for key metrics would also provide a more robust characterization of model performance variability.

Hyperparameter Optimization: The current models were trained with default or minimally tuned hyperparameters to establish a uniform baseline. Advanced tuning techniques such as Bayesian Optimization using Optuna, grid search, or random search with cross-validation could further improve the accuracy of XGBoost, LightGBM, and other models. Systematic tuning of parameters including learning rate, tree depth, number of estimators, regularization strengths, and subsampling ratios may push the R-squared above 0.90 and reduce error further. Given the computational cost of tuning on a dataset of this size, efficient optimization strategies should be employed.

2) Medium-Priority Directions

Online Learning and Concept Drift Adaptation: Workload characteristics in production environments evolve over time due to hardware upgrades, software changes, shifting user demands, and evolving application patterns. Future work should develop incremental learning approaches that periodically retrain the model on recent process data, or employ online learning algorithms such as Hoeffding Trees or online gradient boosting that can update the model continuously without full retraining. Research should investigate optimal retraining frequencies, performance degradation patterns over time, and automated detection of concept drift to trigger retraining only when necessary. The lightweight nature of KNN with its minimal training time also suggests a hybrid approach where a fast base predictor can adapt quickly until a more accurate model is retrained.

Cross-Environment Generalization and Transfer Learning: The current dataset is from a single HPC environment. Testing the trained models on diverse workload traces—such as the Google Cluster Trace, Alibaba Cloud traces, traces from the Parallel Workloads Archive representing different clusters, Azure Public Dataset, or traces from edge and mobile devices—would assess generalizability across different computing environments. If performance drops significantly, transfer learning techniques could be employed to adapt a pre-trained model to a new environment with limited labeled data, reducing the data requirements for deployment in new settings.

Explainability and Interpretability with SHAP: Although feature importance was extracted from XGBoost, a deeper layer of interpretability can be achieved using SHAP (SHapley Additive exPlanations) values. SHAP would provide instance-level explanations, allowing system administrators to understand why a particular burst time prediction was made for a given process. This would increase trust in the prediction system, facilitate debugging in production environments, and support safety-critical scheduling scenarios where understanding the rationale behind decisions is essential. SHAP analysis could also reveal subtle patterns in the data, such as which feature combinations lead to systematic overestimates or underestimates.

3) Lower-Priority but Potentially High-Impact Directions

Deployment in Real Systems: The ultimate validation of this research lies in its deployment within a real operating system kernel or a container orchestration platform such as Kubernetes. Future work should investigate modifying the Linux Completely Fair Scheduler to incorporate ML-based burst predictions, measuring the overhead of model inference in the scheduling hot path. Inference-optimized formats like ONNX or LightGBM's C API could be leveraged to minimize latency. This work would also need to address practical challenges including integration with existing scheduler logic, handling of missing or malformed input, graceful degradation on model failure, and security considerations for adversarial inputs.

Hybrid Models and Deep Learning: While gradient boosting decision trees proved most effective in this study, deep learning models such as Long Short-Term Memory networks or Transformers could capture temporal patterns across successive process submissions by a user or within a job array. Exploring hybrid architectures that combine gradient boosting models for static features with recurrent networks for sequential history may yield further accuracy gains, provided that inference latency remains acceptable for scheduling contexts. The computational cost of deep learning models, both for training and inference, must be carefully weighed against the accuracy benefits in the context of operating system schedulers.

Advanced Feature Engineering: Beyond the three engineered features created in this study, additional domain-specific features could be derived to capture more nuanced aspects of process behavior. Temporal features such as hour of day, day of week, and time since last process submission could capture diurnal patterns and user behavior cycles. Statistical features derived from historical process characteristics could provide richer representations of workload patterns. Feature selection techniques could identify the minimal set of features required for near-optimal prediction, reducing monitoring overhead in production environments.

Robustness to Adversarial Inputs: In public cloud or multi-tenant environments, malicious actors might attempt to manipulate scheduling decisions through crafted resource request profiles. Future work should investigate the robustness of prediction models to adversarial inputs, developing detection mechanisms for anomalous patterns and defensive techniques such as input validation, outlier detection, and adversarial training.

Resource-Constrained Deployment: For deployment in resource-constrained environments such as embedded systems or edge devices, model compression techniques including quantization, pruning, and knowledge distillation could reduce model size and inference latency. The trade-off between model size, inference speed, and prediction accuracy should be explored to identify optimal configurations for different deployment scenarios.

FUNDING STATEMENT

The author(s) received no specific funding for this study.

CONFLICTS OF INTEREST

The authors declare no conflicts of interest to report regarding the present study.

AUTHOR CONTRIBUTIONS

Conceptualization, H. and A.H.; methodology, M.A. and H.; validation, H., and M.A.; writing—original draft preparation, H.; writing—review and editing, M.A. and A.H.; supervision, M.A.; data curation, A.H., and M.A.; investigation, H.; visualization, H. and A.H.; formal analysis, A.H.; resources, A.H., and M.A.;

INSTITUTIONAL REVIEW BOARD STATEMENT

Not applicable.

INFORMED CONSENT STATEMENT

Not applicable.

DATA AVAILABILITY STATEMENT

Data is available on reasonable request.

CONSENT FOR PUBLICATION

All authors have read and approved the final version of the manuscript and consent to its publication.

DECLARATION OF GENERATIVE AI IN SCIENTIFIC WRITING

For preparing this manuscript, the authors have only made use of generative artificial intelligence systems to enhance language proficiency, grammar, and readability.

REFERENCES

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed. Hoboken, NJ, USA: Wiley, 2018.

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

[3] E. Effah, S. J. Atsu, Z. A. Brew, J. K. Mensah, E. A. Quaicoe, M. O. Ansah, A. Yeful, and R. P. Baffoe, "Predicting CPU Burst Times with ML to Enhance Shortest Job First (SJF) and Shortest Remaining Time First (SRTF) CPU Scheduling," *Int. J. Comput. Appl.*, vol. 185, no. 4, pp. 1–8, 2023.

[4] E. Effah, S. J. Atsu, Z. A. Brew, J. K. Mensah, E. A. Quaicoe, M. O. Ansah, A. Yeful, and R. P. Baffoe, "Comparative Analysis and Implementation of AI Algorithms and NN Model in Process Scheduling Algorithm," unpublished, 2023.

[5] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 785–794.

[6] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Y. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 3149–3157.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[8] N. Fawad, M. N. Khan, A. Addas, Q. Awais, and N. Ayub. "Game mechanics and artificial intelligence personalization: A framework for adaptive learning systems." *Education Sciences* 15, no. 3, p. 301, 2025.

[9] S.R. Babary, and R. Khalid. "Leveraging Transformer-Based Natural Language Processing for Adaptive Language Learning in Digital Humanities Environments." *Journal of Artificial Intelligence and Computing* 4, no. 1, p. 36-43, 2026.

[10] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed. Cambridge, MA, USA: Morgan Kaufmann, 2017.

[11] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. Upper Saddle River, NJ, USA: Pearson, 2015.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[13] Y. Li, Y. Wang, and W. Zhang, "Deep Learning for Workload Prediction in Cloud Computing," *IEEE Trans. Cloud Comput.*, vol. 10, no. 2, pp. 456–469, 2022.

[14] S. O. Ogunleye, O. A. Ogunbiyi, and A. O. Adetunji, "Intelligent CPU Scheduling Using Reinforcement Learning," in *Proc. USENIX Annu. Tech. Conf.*, 2021, pp. 123–136.

[15] H. Wang, X. Liu, and Z. Chen, "Burst Time Prediction for Dynamic Job Scheduling Using LSTM Networks," *Future Gener. Comput. Syst.*, vol. 145, pp. 78–92, 2023.

[16] K. Singh and M. Sharma, "Performance Analysis of ML-Based Schedulers in Linux Kernel," *J. Syst. Archit.*, vol. 138, Art. no. 102857, 2023.

[17] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 4765–4774.

[18] L. Breiman, "Random Forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[19] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *Ann. Stat.*, vol. 29, no. 5, pp. 1189–1232, 2001.

[20] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2019, pp. 2623–2631.

TABLE IV: PERFORMANCE COMPARISON OF MACHINE LEARNING MODELS ON TEST SET

Rank	Model	R^2 Score	Adjusted R^2	RMSE (s)	MAE (s)	MAPE (%)	Training Time (s)
1	**XGBoost**	**0.872**	**0.871**	**487.3**	**196.5**	**14.2**	**1.66**
2	**LightGBM**	**0.868**	**0.867**	**495.8**	**202.1**	**14.8**	**0.94**
3	Random Forest	0.847	0.846	542.6	231.4	16.5	122.75
4	Gradient Boosting	0.832	0.831	568.9	245.7	17.8	134.54
5	K-Nearest Neighbors	0.843	0.842	551.3	239.8	17.1	0.015
6	Decision Tree	0.824	0.823	581.4	258.3	18.6	2.80
7	Multi-Layer Perceptron	0.782	0.781	647.2	303.6	22.3	1328.60
8	Ridge Regression	0.703	0.703	758.1	381.2	29.5	0.13
9	Linear Regression	0.701	0.701	760.5	383.9	29.8	0.13
10	Lasso Regression	0.700	0.700	761.4	385.1	30.0	1.23
11	AdaBoost	0.649	0.648	823.7	436.2	35.6	6.51
12	ElasticNet	0.623	0.622	851.2	452.9	37.9	0.18

Note: MAPE was calculated after excluding processes with zero actual RunTime. All metrics are computed on the 30% held-out test set. Adjusted R^2 accounts for the number of predictors; as all models used identical feature sets after preprocessing, differences from R^2 are minimal. Explained Variance Scores were essentially identical to R^2 and are omitted for brevity.