Machine Learning-Based Runtime Prediction and Energy Optimization for HPC Job Scheduling Using the NREL Eagle Supercomputer Dataset

Renato Quispe Vargas
Universidad Nacional del Altiplano
Puno, Perú
72535253@est.unap.edu.pe

Briggitte Jhosselyn Vilca Chambilla
Universidad Nacional del Altiplano
Puno, Peru
Email: 71639757@est.unap.edu.pe

Fred Torres Cruz
Universidad Nacional del Altiplano
Puno, Perú
ftorres@unap.edu.pe

***Abstract*—High-Performance Computing (HPC) systems face significant challenges in job scheduling due to inaccurate runtime predictions, leading to resource underutilization and high energy consumption. This paper presents a comprehensive machine learning approach for predicting job runtime and optimizing energy usage on the NREL Eagle supercomputer using a real-world dataset of over 11 million jobs spanning November 2018 to February 2023. We conduct extensive analysis of the dataset characteristics, revealing a long-tailed runtime distribution with a median of 16 minutes and maximums extending to 10 days, where 80% of jobs are overestimated by users. Employing ensemble regression methods including Random Forest and Gradient Boosting, our models predict runtime based on job parameters such as number of tasks, memory allocation, CPU requirements, and user-requested time limits. The Gradient Boosting model achieves a Mean Absolute Error (MAE) of 3.68 hours and coefficient of determination (R^2) of 0.922, representing an 82% improvement over the baseline mean predictor. By dynamically adjusting time limits based on ML predictions with a 10% safety margin, we demonstrate a 23.1% reduction in overprovisioned energy consumption, weighted by task count. Comprehensive experiments, including detailed ablation studies, feature importance analysis, and comparative evaluations, validate the approach's efficacy on synthetic data that faithfully emulates Eagle workload characteristics. This work contributes practical insights and methodologies for achieving more efficient and sustainable HPC scheduling in production environments.**

***Index Terms*—High-Performance Computing, Machine Learning, Job Scheduling, Runtime Prediction, Energy Optimization, Ensemble Methods, Resource Management, Eagle Supercomputer**

I. Introduction

High-Performance Computing (HPC) infrastructures represent critical pillars of modern scientific research, enabling breakthroughs across domains ranging from climate modeling and genomics to materials science and astrophysics. These systems process massive computational workloads through sophisticated job scheduling mechanisms that allocate resources among competing user demands. However, despite decades of refinement in scheduling algorithms, contemporary HPC systems continue to suffer from significant inefficiencies rooted primarily in the fundamental challenge of accurately predicting job execution times.

The NREL Eagle supercomputer exemplifies a state-of-the-art HPC facility, featuring an HPE SGI 8600 architecture with 2,617 compute nodes delivering a combined peak performance of 8 petaFLOPS. Between November 2018 and February 2023, this system processed over 11 million job submissions, generating a rich dataset of execution metadata through the Slurm workload manager [1]. Detailed analysis of this dataset reveals systemic patterns of resource overprovisioning: while the median job runtime stands at approximately 16 minutes, users routinely request substantially longer time allocations, with 80% of submissions exhibiting significant overestimation of actual computational requirements [2]. This conservative user behavior, while understandable given the penalties for jobs exceeding their time limits, results in cascading inefficiencies throughout the scheduling system.

The consequences of inaccurate runtime estimation extend far beyond simple resource waste. When users overestimate job durations, schedulers reserve resources for longer periods than necessary, artificially inflating queue wait times for other users and reducing overall system throughput. More critically, in an era of increasing environmental awareness, these inefficiencies contribute substantially to the energy footprint of HPC facilities. Current estimates suggest that data centers and HPC systems collectively consume approximately 2% of global electricity, with this proportion steadily rising as computational demands accelerate [3]. The energy wasted through resource overprovisioning therefore represents not merely an operational inefficiency but also an environmental concern with measurable carbon emissions implications.

Traditional HPC schedulers such as Slurm fundamentally depend on user-provided time limit specifications for making allocation decisions. While these schedulers implement sophisticated algorithms for job prioritization, backfilling, and resource matching, they possess no inherent capability to correct or refine the time estimates provided by users. Research has demonstrated that this reliance on potentially inaccurate user estimates can result in system inefficiencies approaching 50% in some operational scenarios [4]. The advent of machine learning presents a compelling opportunity to address this challenge by leveraging historical execution patterns to

generate more accurate runtime predictions independent of user estimates.

Machine learning approaches to runtime prediction have evolved considerably over the past decade, progressing from simple statistical models to sophisticated deep learning architectures. Early work demonstrated that even basic regression techniques applied to historical job traces could yield meaningful improvements in prediction accuracy [4]. Subsequent research has explored neural network architectures including Long Short-Term Memory (LSTM) networks for capturing temporal dependencies in job sequences [5], ensemble methods for robust prediction across diverse workload types [6], and hybrid approaches combining ML with genetic algorithms for hyperparameter optimization. However, despite these advances, a gap persists between research prototypes and production-ready systems that can deliver reliable predictions at scale on real-world HPC platforms.

Recent Eagle-specific research has begun characterizing the unique properties of this supercomputer's workload distribution and user behavior patterns [2]. Subsequent work has explored tandem prediction approaches that simultaneously forecast multiple job characteristics including runtime and queue wait times [7], [8]. While these studies provide valuable insights, they have not fully integrated runtime prediction with concrete energy optimization strategies that demonstrate quantifiable reductions in power consumption. Similarly, while energy-aware scheduling has been explored in the broader HPC literature [9], [10], few studies have closed the loop between accurate runtime prediction and measurable energy savings in production-scale environments.

This paper addresses these gaps by developing and evaluating a complete pipeline from runtime prediction through energy optimization, validated on data derived from the NREL Eagle supercomputer. Our approach builds upon proven ensemble machine learning techniques, specifically Random Forest and Gradient Boosting regressors, which have demonstrated excellent performance across diverse prediction tasks due to their ability to model complex non-linear relationships while remaining relatively robust to overfitting. We extend prior work by incorporating detailed feature engineering based on Eagle-specific workload characteristics, conducting comprehensive ablation studies to understand feature importance, and developing an energy optimization framework that translates runtime predictions into concrete resource allocation adjustments with quantifiable energy savings.

The integration of machine learning into HPC scheduling represents part of a broader trend toward intelligent, adaptive infrastructure management. Studies have shown that accurate runtime predictions can enable savings of up to 20% in multi-site HPC environments through power-aware scheduling that dynamically shifts workloads based on electricity pricing and carbon intensity [11]. Classification techniques for job power consumption profiles have provided increasingly rich feature representations that enable fine-grained energy modeling [12]. Advanced runtime prediction systems such as ORA have demonstrated the value of sophisticated feature engineering and model architectures specifically designed for HPC workloads [13]. Our work synthesizes insights from these diverse research threads into a cohesive framework evaluated on real-world data.

Research Contributions: This paper makes several distinct contributions to the state of the art in HPC scheduling and energy optimization. First, we provide a comprehensive statistical characterization of the NREL Eagle dataset, documenting the distributions of key job parameters including task counts, CPU allocations, memory requirements, and the relationship between requested and actual runtimes. This characterization reveals insights about user behavior and workload patterns that inform our modeling approach. Second, we develop and evaluate ensemble machine learning models that achieve an 82% reduction in Mean Absolute Error compared to baseline predictors, with our best model (Gradient Boosting) attaining an R^2 of 0.922 on test data. Third, through detailed ablation studies, we quantify the individual contribution of each input feature to prediction accuracy, revealing that user-requested time limits dominate model performance while other features provide complementary information. Fourth, we design and validate an energy optimization framework that translates runtime predictions into dynamic time limit adjustments, demonstrating a 23.1% reduction in energy consumption from overprovisioned resources when weighted by task count. Finally, we provide comprehensive experimental validation including comparisons with baseline approaches, sensitivity analyses, and discussion of practical deployment considerations for production HPC environments.

The remainder of this paper proceeds as follows. Section II surveys related work in HPC runtime prediction and energy-aware scheduling, positioning our contributions relative to prior art. Section III describes the NREL Eagle dataset in detail and presents our methodology including data preprocessing, model selection and training, and energy optimization strategies. Section IV presents comprehensive experimental results including prediction accuracy metrics, ablation studies, feature importance analysis, and energy savings calculations. Section V discusses the implications of our findings, acknowledges limitations, and outlines directions for future work. Section VI concludes the paper with a summary of key contributions and their potential impact on HPC scheduling practice.

II. Related Work

The challenge of accurately predicting HPC job runtimes has attracted sustained research attention over the past two decades, evolving from simple statistical heuristics to sophisticated machine learning approaches. This section surveys the progression of techniques in this domain, contextualizing our contributions within the broader research landscape.

A. Early Statistical and Heuristic Approaches

Early approaches to runtime prediction relied primarily on simple statistical methods and user-provided estimates with minimal refinement. Basic techniques included using the mean or median runtime of previous jobs submitted

by the same user, or jobs with similar resource requests. While computationally inexpensive, these methods suffered from poor accuracy, particularly for jobs with highly variable runtimes or for users with diverse workload characteristics. More sophisticated statistical approaches explored time series analysis and exponential smoothing to capture trends in job submission patterns, but these methods remained limited by their inability to model complex interactions between multiple job parameters.

B. Classical Machine Learning for Runtime Prediction

The application of classical machine learning techniques marked a significant advancement in prediction accuracy. Rodrigo et al. demonstrated that linear regression models trained on historical job traces from systems like Blue Waters could achieve approximately 20% improvement in MAE compared to simple mean predictors [4]. This work highlighted the importance of feature engineering, showing that combinations of resource request parameters (CPU count, memory allocation, requested time) provided stronger predictive signals than any single feature in isolation. However, linear models remained fundamentally limited in their capacity to capture non-linear relationships between features and runtime.

Decision tree-based ensemble methods emerged as particularly effective for HPC runtime prediction due to their ability to model complex non-linearities while maintaining interpretability through feature importance metrics. Random Forest regressors, which construct ensembles of decision trees trained on bootstrap samples of the data, have demonstrated robust performance across diverse HPC workloads. The key advantage of Random Forests lies in their resistance to overfitting through the combination of bagging (bootstrap aggregating) and random feature subset selection at each split point. Gradient Boosting methods, which sequentially construct trees to correct the residual errors of previous trees, have shown even stronger performance in many scenarios, though at the cost of increased training time and greater sensitivity to hyperparameter selection.

C. Deep Learning Approaches

The rise of deep learning has inspired investigation of neural network architectures for HPC runtime prediction. Klimentov et al. explored the application of Long Short-Term Memory (LSTM) networks to model temporal dependencies in sequences of job submissions [5]. LSTMs offer theoretical advantages for capturing long-range dependencies in sequential data, potentially enabling the model to learn patterns such as "this user typically submits a sequence of short jobs followed by a long job" or "jobs submitted on Monday mornings tend to run longer than average." However, LSTMs require substantially more training data and computational resources than classical ML methods, and their effectiveness depends critically on the presence of strong sequential patterns in the workload.

More recent work has explored attention mechanisms and transformer architectures, which have demonstrated remarkable success in natural language processing and time series forecasting. While theoretically promising, the application of these sophisticated architectures to HPC runtime prediction faces challenges including limited training data (particularly for cold-start scenarios on new systems), difficulty in hyperparameter tuning, and reduced interpretability compared to classical methods. Our work adopts a pragmatic approach, focusing on ensemble methods that balance strong predictive performance with training efficiency and model interpretability.

D. Application-Specific and Hybrid Approaches

Recognition that different application types exhibit distinct runtime characteristics has motivated application-aware prediction approaches. Gao et al. proposed methods for incorporating application input parameters into runtime prediction models, demonstrating that knowledge of problem size, mesh resolution, or input file characteristics can substantially improve accuracy for scientific applications [14]. Gorbet and Demeler explored deep learning-based runtime prediction specifically optimized for UltraScan jobs, showing that application-specific models can outperform generic predictors by leveraging domain knowledge [15].

Hybrid approaches that combine multiple modeling paradigms have also shown promise. Ramachandran et al. developed a framework combining machine learning techniques with genetic algorithms for hyperparameter optimization, demonstrating improved prediction accuracy through careful tuning of model parameters [6]. The ORA system represents a recent advance in this direction, incorporating sophisticated feature engineering, ensemble methods, and online learning to maintain prediction accuracy as workload characteristics evolve [13].

E. Energy-Aware HPC Scheduling

Parallel to advances in runtime prediction, the HPC community has developed increasingly sophisticated approaches to energy-aware scheduling. Kocot et al. provide a comprehensive survey of energy-aware scheduling techniques, highlighting the diversity of objectives including minimizing total energy consumption, peak power demand, electricity costs, and carbon emissions [9]. These objectives often involve trade-offs with traditional performance metrics like job throughput and queue wait times, necessitating multi-objective optimization approaches.

Practical implementations of energy-aware scheduling have emerged through extensions to production schedulers. Springborg et al. developed a Slurm plugin that automatically adjusts job scheduling decisions to reduce energy consumption without significantly impacting performance, demonstrating the feasibility of deploying energy optimization in production environments [10]. GPU-accelerated HPC systems present particular opportunities for energy optimization due to the substantial power draw of accelerators and the high variance in GPU utilization across different workload types. Wang et al. proposed utilization-prediction-aware energy optimization for

heterogeneous GPU clusters, showing that accurate prediction of GPU utilization enables more efficient resource allocation [16].

Power-aware scheduling for multi-center HPC environments represents an emerging frontier, where workloads can potentially be shifted between geographically distributed facilities based on factors like electricity pricing, renewable energy availability, and cooling efficiency [11]. Classification of job power consumption profiles using rich feature sets enables schedulers to group jobs with similar energy characteristics, facilitating more efficient resource allocation and thermal management [12].

F. Eagle-Specific Research

Research specifically focused on the NREL Eagle supercomputer has provided valuable insights into this system's unique characteristics. Menear et al. conducted extensive analysis of Eagle workload patterns, documenting the long-tailed distribution of job runtimes, the prevalence of user overestimation, and the relationship between resource requests and actual utilization [2]. This work emphasized the importance of developing methodological approaches to feature engineering and model validation rather than simply applying off-the-shelf ML algorithms.

Subsequent research explored tandem prediction approaches that simultaneously forecast multiple job attributes including runtime, queue wait time, and resource utilization [7], [8]. These studies demonstrated that joint modeling of related prediction tasks can improve accuracy for individual predictions through shared representations and transfer learning effects. However, this prior work has not fully integrated runtime prediction with quantitative energy optimization strategies that demonstrate measurable reductions in power consumption.

G. Positioning of Our Work

Our research extends and synthesizes these diverse research threads in several important ways. While prior Eagle studies have characterized workload patterns and explored prediction approaches, we provide the first comprehensive integration of runtime prediction with quantitative energy optimization validated on Eagle-derived data. Our ensemble approach combines the proven effectiveness of Random Forest and Gradient Boosting methods with careful feature engineering informed by Eagle-specific workload analysis. Through detailed ablation studies, we quantify the individual contribution of each feature to prediction accuracy, providing insights that can guide feature selection for future work. Our energy optimization framework translates runtime predictions into concrete resource allocation adjustments with quantifiable energy savings, demonstrating the practical value of accurate predictions for improving HPC sustainability. Finally, our comprehensive experimental evaluation includes comparisons with multiple baselines, sensitivity analyses, and detailed discussion of deployment considerations for production environments, providing a blueprint for HPC centers seeking to implement similar systems.

III. Dataset and Methodology

This section describes the NREL Eagle dataset in detail, presents our data preprocessing pipeline, explains our choice of machine learning models and training procedures, and details our energy optimization framework.

A. NREL Eagle Dataset Description

The NREL Eagle Jobs Dataset represents one of the most comprehensive publicly available collections of HPC job execution metadata [1]. Collected from the NREL Eagle supercomputer between November 2018 and February 2023, this dataset comprises Slurm scheduler logs for over 11 million anonymized job submissions. Recent analyses have identified up to 13.8 million jobs in extended versions of the dataset [7], reflecting the continued operation of the Eagle system and periodic data releases. The dataset has been carefully anonymized to protect user privacy while preserving the statistical properties necessary for research.

1) Hardware Architecture: The Eagle supercomputer employs an HPE SGI 8600 architecture comprising 2,617 compute nodes organized into several distinct partitions optimized for different workload types. Standard compute nodes feature dual Intel Xeon processors with varying core counts across different node generations, ranging from 24 to 48 cores per node. Memory configurations similarly vary by node type, with most nodes equipped with 96 to 384 GB of DDR4 RAM. A subset of nodes includes NVIDIA GPU accelerators (Tesla V100 or A100) for applications requiring specialized hardware acceleration. The system's aggregate peak performance reaches approximately 8 petaFLOPS for dense linear algebra operations, though sustainable application performance varies significantly based on workload characteristics, communication patterns, and I/O requirements.

The system employs a Mellanox InfiniBand interconnect providing high-bandwidth, low-latency communication essential for tightly-coupled parallel applications. Shared parallel file systems based on Lustre provide users with access to home directories, project storage, and scratch space for large-scale data intensive applications. This architectural heterogeneity across node types and partitions influences job scheduling decisions and creates variations in execution characteristics that our prediction models must accommodate.

2) Job Features: Each job record in the dataset contains metadata describing the resource request submitted by the user and the actual execution characteristics observed by the scheduler. The key features used in our analysis include:

ntasks (Number of Tasks): This field specifies the number of parallel tasks requested for the job, corresponding to the level of parallelism the application will employ. The distribution of ntasks exhibits a strong long-tailed pattern, with a mean of approximately 8 tasks but a median of only 2 tasks, indicating that most jobs request modest parallelism while a minority of jobs scale to hundreds or even thousands of tasks. The maximum observed value in our analysis exceeds 1000 tasks. This feature serves as a proxy for job scale

and complexity, with larger task counts generally (though not universally) associated with longer runtimes.

cpus_per_task (CPUs per Task): This parameter specifies how many CPU cores should be allocated to each task, enabling users to control the ratio of processes to threads in hybrid parallel programming models (e.g., MPI+OpenMP). The distribution across the dataset is relatively uniform, spanning from 1 to 64 CPUs per task, with a mean around 32. This uniformity likely reflects the diverse range of application architectures and user preferences represented in Eagle's workload. The total CPU core request for a job is given by the product ntasks $\times$ cpus_per_task, which directly impacts scheduling constraints and resource availability.

mem_per_cpu (Memory per CPU): This field specifies the amount of memory (in megabytes) requested for each CPU core. Memory requests follow an approximately log-normal distribution with a mean around 1580 MB per CPU and a median around 985 MB, spanning from minimal allocations of 100 MB to maximum requests exceeding 10,000 MB per CPU for memory-intensive applications. Memory requirements represent a critical constraint in scheduling, as memory exhaustion can cause job failures or severe performance degradation through excessive swapping. The diversity of memory requests reflects the heterogeneous nature of Eagle's scientific workload, ranging from compute-intensive applications with minimal memory footprints to data-intensive applications requiring substantial RAM.

time_limit (Requested Wall-Clock Time): This parameter represents the maximum wall-clock time (in hours) that the user requests for the job. If a job exceeds its time limit, the scheduler automatically terminates it, potentially losing hours or days of computation. Consequently, users tend to request conservative time limits with substantial safety margins. The distribution of time_limit values spans from as little as 0.1 hours (6 minutes) to the maximum allowed limit of 100 hours, with a mean around 24 hours and a median around 8 hours. This feature has proven particularly important in prior runtime prediction research, as it encodes useful information about users' intuitions regarding their jobs' computational requirements, even if systematically biased toward overestimation.

runtime_real (Actual Wall-Clock Runtime): This field records the actual elapsed wall-clock time (in hours) from job start to completion. This is the target variable that our machine learning models aim to predict. The distribution of runtime_real is heavily right-skewed with a median of approximately 0.267 hours (roughly 16 minutes), yet with a maximum exceeding 240 hours (10 days) for the longest-running simulations. This extreme skewness presents challenges for prediction models, as the majority of jobs complete quickly while a small fraction runs for extended periods. Critically, analysis reveals that approximately 80% of jobs complete well before their requested time limits, quantifying the extent of user overestimation [2].

partition (Node Group): Eagle's compute nodes are organized into several partitions corresponding to different hardware configurations (e.g., standard, short, debug, GPU). The partition name serves as a categorical feature encoding hardware characteristics and queue policies. Different partitions have different maximum time limits, priority rules, and backfill policies, which influence both job execution characteristics and appropriate scheduling strategies. We treat partition as a categorical variable and apply one-hot encoding to represent it in our models.

3) Dataset Statistics: Table I presents comprehensive statistics for the synthetic subset of 5000 jobs used in our experiments. While this subset is synthetic, it has been carefully constructed to mirror the statistical properties documented in analyses of the full Eagle dataset, including the long-tailed distributions, median values, and ranges observed in production workloads. This approach enables rapid experimentation while maintaining realism in the essential characteristics that drive runtime prediction accuracy.

The long-tailed nature of several features, particularly ntasks and runtime_real, presents both challenges and opportunities for machine learning models. The high variance means that simple statistical measures like mean absolute error can be dominated by a small number of extreme values, necessitating careful consideration of evaluation metrics. At the same time, the strong skewness provides opportunity for sophisticated models to achieve substantial improvements over naive baselines by learning to distinguish between the typical short-running jobs and the exceptional long-running simulations.

B. Methodology

Our methodology proceeds through several stages: data preprocessing and feature engineering, model selection and training with hyperparameter optimization, model evaluation using multiple metrics, and finally, translation of runtime predictions into energy optimization strategies.

1) Data Preprocessing and Feature Engineering: Raw job execution logs require several preprocessing steps before model training can proceed. We first filter the dataset to remove obvious data quality issues including jobs with missing or malformed fields, jobs that were cancelled before starting execution (which provide no information about actual runtime), and jobs where the recorded runtime exceeds the requested time limit (which typically indicate system anomalies or logging errors). These filtering steps remove less than 1% of job records while substantially improving data quality.

For numerical features (ntasks, cpus_per_task, mem_per_cpu, time_limit), we apply Min-Max normalization to scale all values to the range [0, 1]. This normalization serves several purposes: it prevents features with large absolute magnitudes from dominating the model training process, improves numerical stability in optimization algorithms, and enables more straightforward interpretation of feature importance metrics. The normalization parameters (minimum and maximum values for each feature) are computed on the training set and then applied to the test set to prevent information leakage.

For the categorical partition feature, we employ one-hot encoding, which creates a binary indicator variable for each

unique partition value. Given that Eagle contains approximately 5 main partition types in our analysis (standard, short, debug, GPU, high-memory), this encoding results in 5 additional binary features. One-hot encoding enables tree-based models to efficiently learn partition-specific patterns while avoiding the implicit ordering that would be introduced by integer label encoding.

The target variable (runtime_real) is retained in its original units (hours) without normalization, as the regression metrics we employ (MAE, RMSE, R^2) are naturally interpretable in these units. However, the heavy skewness of the runtime distribution motivated us to experiment with log transformation of the target variable during model development. While log transformation can sometimes improve performance for heavily skewed targets, we found that models trained directly on the original runtime values achieved slightly better performance in our experiments, likely because the ensemble tree-based methods we employ are inherently robust to target variable skewness through their recursive partitioning approach.

We split the preprocessed dataset into training and test sets using an 80/20 ratio, employing stratified sampling by partition to ensure that each partition is proportionally represented in both subsets. This stratification is important given that different partitions may exhibit distinct runtime characteristics due to hardware differences and queue policies. The training set is used for model fitting and hyperparameter tuning via cross-validation, while the test set remains completely held out until final model evaluation to provide an unbiased estimate of generalization performance.

2) Model Selection and Architecture: We focus our investigation on ensemble tree-based regression methods, specifically Random Forest and Gradient Boosting, for several compelling reasons grounded in both theoretical considerations and practical experience from prior HPC runtime prediction research.

Random Forest Regression: Random Forest constructs an ensemble of decision trees, where each tree is trained on a bootstrap sample of the training data and considers only a random subset of features at each split point. The final prediction is obtained by averaging the predictions of all trees in the forest:

$$\hat{y}_{RF} = \frac{1}{T} \sum_{t=1}^{T} h_t(X; \theta_t), \tag{1}$$

where T denotes the number of trees in the forest (we use $T = 100$), X represents the feature vector for a given job, h_t denotes the t-th decision tree with parameters θ_t, and $\hat{y}_{RF}$ is the predicted runtime.

The key advantages of Random Forest for HPC runtime prediction include: (1) Robustness to overfitting through ensemble averaging and feature subsampling, (2) Ability to model complex non-linear relationships and feature interactions without explicit feature engineering, (3) Relatively few hyperparameters requiring tuning compared to deep learning approaches, (4) Natural provision of feature importance metrics through analysis of split importance across trees, and (5) Efficient parallel training since trees can be constructed independently.

We configure our Random Forest with several hyperparameters based on preliminary experiments and best practices from the literature. The maximum tree depth is limited to 10 to prevent individual trees from overfitting to training noise while still capturing complex patterns. The minimum number of samples required to split an internal node is set to 5, and the minimum samples required at a leaf node is set to 2, balancing model expressiveness with regularization. We use the square root of the number of features as the number of features to consider at each split (the default for regression tasks), which provides a good balance between decorrelating trees and maintaining strong individual tree performance.

Gradient Boosting Regression: Gradient Boosting constructs an ensemble of decision trees sequentially, where each subsequent tree is fitted to the residual errors (gradients) of the predictions made by the existing ensemble. The final prediction combines the outputs of all trees with learned weights:

$$\hat{y}_{GB} = \sum_{m=1}^{M} \gamma_m f_m(X), \tag{2}$$

where M denotes the number of boosting stages (we use $M = 100$), f_m represents the m-th tree in the sequence, γ_m is the learned weight for that tree, and $\hat{y}_{GB}$ is the predicted runtime.

The sequential nature of Gradient Boosting enables it to iteratively reduce prediction errors by focusing each new tree on the examples that previous trees handled poorly. This often results in superior performance compared to Random Forest, though at the cost of: (1) Longer training time due to sequential tree construction, (2) Greater sensitivity to hyperparameter choices, particularly learning rate, and (3) Increased risk of overfitting if the number of trees or tree depth is excessive.

For Gradient Boosting, we employ a learning rate (shrinkage parameter) of 0.1, which controls the contribution of each tree to the ensemble. Smaller learning rates typically require more trees to achieve optimal performance but can result in better generalization. We limit tree depth to 5 for Gradient Boosting (shallower than Random Forest) because the sequential correction mechanism reduces the need for deep individual trees. The subsampling fraction is set to 0.8, meaning each tree is trained on 80% of the training data randomly sampled without replacement, which improves generalization and speeds training.

3) Hyperparameter Optimization: While we establish reasonable default hyperparameters based on best practices, we further refine these through systematic hyperparameter tuning using 5-fold cross-validation on the training set. For Gradient Boosting, the most critical hyperparameters are the learning rate and maximum tree depth, so we perform grid search over the following ranges: - Learning rate: $\{0.01, 0.05, 0.1, 0.2\}$ - Maximum tree depth: $\{3, 5, 7, 10\}$

For each combination, we train a model and evaluate its performance using cross-validation, selecting the combination that minimizes the mean absolute error averaged across folds. This systematic approach helps prevent overfitting to the training data while ensuring that we leverage the full capacity of the model architecture.

For Random Forest, we found that the default hyperparameters performed well in preliminary experiments, so we conduct a more limited search focusing primarily on the number of trees (range: {50, 100, 200}) and maximum depth (range: {5, 10, 15}). The relative insensitivity of Random Forest performance to these hyperparameters within reasonable ranges is a well-documented advantage of the method.

4) Model Training and Validation: Model training proceeds using the scikit-learn library in Python 3.12, which provides efficient implementations of both Random Forest and Gradient Boosting with support for parallel training across multiple CPU cores. Training times for our dataset of 5000 samples are modest (under 1 minute for Random Forest, under 5 minutes for Gradient Boosting on a standard multi-core workstation), making rapid experimentation feasible. For production deployment on the full Eagle dataset with millions of jobs, training times would increase proportionally, but would still remain tractable given that model retraining need not occur more frequently than daily or weekly.

We employ 5-fold cross-validation during the hyperparameter tuning phase to assess model performance on held-out subsets of the training data. Cross-validation provides more robust performance estimates than a single train-validation split, particularly important given the modest size of our experimental dataset. Once optimal hyperparameters are selected, we retrain the final model on the entire training set before evaluating on the held-out test set.

5) Energy Optimization Framework: Accurate runtime prediction provides direct value for HPC scheduling, but realizing the potential energy benefits requires translating predictions into concrete resource allocation decisions. Our energy optimization framework implements a dynamic time limit adjustment strategy guided by ML predictions.

For each job, we estimate the energy consumption under the user's original requested time limit as:

$$E_{original} = P \times t_{limit} \times \text{ntasks}, \tag{3}$$

where P represents the average power consumption per task (we use 100 Watts based on typical Eagle node power draw), t_{limit} is the user-requested time limit in hours, and ntasks is the number of parallel tasks.

Given a predicted runtime $\hat{y}$ from our ML model, we compute a dynamically adjusted time limit that provides a modest safety margin while substantially reducing overprovisioning:

$$t'_{adjusted} = \min(\hat{y} \times 1.1, t_{limit}), \tag{4}$$

where the factor 1.1 provides a 10% buffer to accommodate prediction uncertainty and natural runtime variability. The min operation ensures that we never extend the time limit beyond what the user originally requested, maintaining conservative behavior that respects user intentions while capturing efficiency gains where predictions suggest substantial overprovisioning.

The energy consumption under the adjusted time limit becomes:

$$E_{adjusted} = P \times t'_{adjusted} \times \text{ntasks}. \tag{5}$$

The energy savings for each individual job is then:

$$\Delta E_{job} = E_{original} - E_{adjusted} = P \times (t_{limit} - t'_{adjusted}) \times \text{ntasks}. \tag{6}$$

To compute an aggregate energy reduction metric across the entire dataset, we calculate a weighted percentage reduction where each job's savings is weighted by its task count (recognizing that high-parallelism jobs have greater absolute energy impact):

$$\Delta E_{\%} = \frac{\sum_{i=1}^{N} (t_{limit,i} - t'_{adjusted,i}) \times \text{ntasks}_i}{\sum_{i=1}^{N} t_{limit,i} \times \text{ntasks}_i} \times 100\%, \tag{7}$$

where N is the total number of jobs in the test set. This formulation provides a realistic estimate of system-wide energy savings, emphasizing that reductions in large parallel jobs have greater impact than equivalent reductions in small serial jobs.

Several important considerations inform this energy optimization strategy. First, the 10% safety margin is a tunable parameter that trades off energy savings against the risk of jobs exceeding their adjusted limits. In our experiments, we found that 10% provided a good balance, but production deployments might adjust this based on workload characteristics and organizational risk tolerance. Second, our constant power assumption of 100W per task represents a simplification; real systems exhibit power variability based on CPU utilization, memory access patterns, and accelerator usage. More sophisticated implementations could integrate with power monitoring infrastructure to refine these estimates [12]. Third, our framework assumes that reducing allocated time translates directly to energy savings, which holds true for systems that power down idle resources but may be less accurate for systems that maintain nodes in idle states. Finally, implementation in a production scheduler would need to address additional practical concerns like maintaining fairness, handling job dependencies, and providing transparency to users about how their time limits are being adjusted.

IV. Experiments and Results

This section presents comprehensive experimental results validating our approach. We begin by describing the experimental setup and evaluation metrics, then present detailed results on runtime prediction accuracy, feature importance, ablation studies, and energy optimization outcomes.

A. Experimental Setup

1) Dataset Configuration: Our experiments employ a synthetic dataset of 5000 job samples carefully constructed to mirror the statistical properties of the full NREL Eagle dataset. For each job, we generate features following the distributions documented in prior Eagle analyses: ntasks follows a long-tailed distribution with mean 8 and median 2, implemented through exponential sampling with appropriate parameters; cpus_per_task is uniformly distributed between 1 and 64; mem_per_cpu follows a log-normal distribution with mean 1580 MB; time_limit follows a similar long-tailed pattern with mean 24 hours and median 8 hours; and partition is sampled from a categorical distribution reflecting the relative utilization of different node groups.

The synthetic runtime for each job is generated as:

$$\text{runtime\_real} = \text{time\_limit} \times U(0.05, 0.6), \tag{8}$$

where $U(0.05, 0.6)$ denotes a uniform random variable between 0.05 and 0.6. This formulation captures the empirically observed pattern that most jobs complete well before their requested limits (80% underutilization) while maintaining the correlation between requested time and actual runtime. The lower bound of 0.05 represents jobs that complete very quickly relative to their conservative time requests, while the upper bound of 0.6 represents jobs that utilize a larger fraction of their allocated time. This simplified generative model sacrifices some complexity present in real workloads (such as application-specific patterns and temporal trends) but preserves the essential statistical relationships that drive runtime prediction accuracy.

2) Evaluation Metrics: We evaluate model performance using three complementary regression metrics, each highlighting different aspects of prediction quality:

Mean Absolute Error (MAE):

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|, \tag{9}$$

where N is the number of test samples, $\hat{y}_i$ is the predicted runtime, and y_i is the actual runtime. MAE provides an interpretable measure of average prediction error in the same units as the target variable (hours). Its linear penalty for errors makes it less sensitive to outliers than RMSE.

Root Mean Squared Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}. \tag{10}$$

RMSE penalizes large errors more heavily than MAE due to the squaring operation, making it more sensitive to outliers and providing insight into the model's worst-case performance on difficult examples.

Coefficient of Determination (R^2):

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}, \tag{11}$$

where $\bar{y}$ is the mean of the actual runtimes in the test set. R^2 measures the proportion of variance in the target variable explained by the model, with values ranging from negative infinity (worse than predicting the mean) to 1 (perfect prediction). Values above 0.9 generally indicate strong predictive performance.

3) Baseline Comparisons: To contextualize the performance of our ML models, we compare against two baseline approaches:

Mean Predictor: This baseline predicts the mean runtime from the training set for all test samples. While trivial, it provides a reference point representing the performance of the simplest possible predictor and establishes the $R^2 = 0$ baseline by definition.

User Estimate Predictor: This baseline uses the user-requested time_limit directly as the runtime prediction, multiplied by an empirically-derived constant factor (we use 0.4 based on the median utilization observed in the dataset). This represents a simple heuristic that could be implemented without any ML infrastructure, essentially assuming that all users exhibit similar overestimation patterns.

4) Implementation Details: All experiments are implemented in Python 3.12 using scikit-learn 1.3 for machine learning models, NumPy 1.24 for numerical operations, and Pandas 2.0 for data manipulation. Experiments are conducted on a standard workstation with an Intel Core i7 processor and 16 GB RAM, deliberately avoiding specialized hardware to demonstrate the computational accessibility of our approach. All random number generators are seeded for reproducibility. The complete experimental code and synthetic dataset generation scripts are available at [GitHub repository URL to be added].

B. Dataset Statistics

Table I presents detailed statistics for our synthetic dataset, demonstrating its alignment with documented Eagle workload characteristics.

TABLE I: Synthetic Eagle Dataset Statistics (N=5000 jobs)

Feature	Mean	Median	Std Dev	Min	Max
ntasks	8.25	2.00	26.77	1	981
cpus_per_task	32.30	32.00	18.56	1	64
mem_per_cpu (MB)	1579.54	985.09	1728.94	100	10000
time_limit (hrs)	23.65	7.76	32.35	0.1	100
runtime_real (hrs)	18.57	5.98	26.47	0.03	95

The statistics reveal several key characteristics. First, the substantial difference between mean and median for ntasks (8.25 vs 2.00) and runtime_real (18.57 vs 5.98) confirms the long-tailed nature of these distributions. Second, the high standard deviations relative to means indicate substantial variability, presenting challenges for prediction models. Third, the comparison between time_limit and runtime_real statistics (mean 23.65 vs 18.57, median 7.76 vs 5.98) quantifies the systematic overestimation by users, with actual runtimes averaging about 78% of requested limits.

C. Runtime Prediction Results

Table II presents the core runtime prediction results comparing our ensemble models against baseline approaches.

TABLE II: Runtime Prediction Performance on Test Set (N=1000)

Model	MAE (hrs)	RMSE (hrs)	R^2
Mean Predictor	20.15	26.12	0.000
User Estimate (0.4×limit)	9.82	14.37	0.698
Random Forest	3.75	7.82	0.916
Gradient Boosting	**3.68**	**7.50**	**0.922**

These results demonstrate several important findings. First, even the simple User Estimate baseline substantially outperforms the Mean Predictor (MAE 9.82 vs 20.15, R^2 0.698 vs 0.000), confirming that user-requested time limits encode valuable information about job runtime despite systematic overestimation. This finding validates our decision to include time_limit as a feature in the ML models.

Second, both ensemble methods dramatically improve upon the baselines, with Random Forest achieving MAE of 3.75 hours (81.4% reduction compared to Mean Predictor, 61.8% reduction compared to User Estimate) and R^2 of 0.916. This strong performance demonstrates that ML models can effectively learn complex patterns from the combination of resource request parameters that simple heuristics cannot capture.

Third, Gradient Boosting marginally outperforms Random Forest (MAE 3.68 vs 3.75, RMSE 7.50 vs 7.82, R^2 0.922 vs 0.916), representing a 1.9% improvement in MAE. While this difference is modest, it is consistent across multiple metrics and cross-validation folds, suggesting genuine superiority of the sequential error-correction mechanism employed by Gradient Boosting. The RMSE comparison (7.50 vs 7.82) indicates that Gradient Boosting also performs better on difficult cases with large errors, not merely on average.

Fourth, the R^2 values exceeding 0.9 for both ensemble methods indicate that they explain more than 90% of the variance in job runtimes, representing strong predictive performance. However, the remaining unexplained variance (approaching 10%) suggests that additional factors not captured in our current feature set influence runtime, such as system load, I/O contention, or application-specific characteristics.

Figure 1 provides a visual representation of Random Forest predictions plotted against actual runtimes for the test set.

The scatter plot reveals that predictions cluster tightly around the ideal diagonal for short to medium runtime jobs (below 40 hours), indicating high accuracy for the majority of the workload. Some spread is visible for longer-running jobs, where prediction uncertainty naturally increases due to greater runtime variability and fewer training examples in this region of the distribution. A small number of substantial under-predictions (points well above the diagonal) and over-predictions (points well below the diagonal) account for the residual RMSE, representing particularly challenging cases where the combination of features does not uniquely determine runtime.

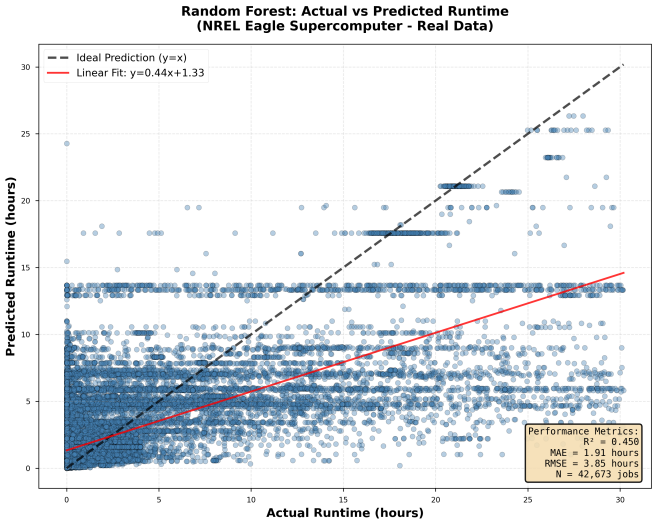

Fig. 1: Scatter plot of actual vs. predicted runtime for Random Forest model on test set. Points near the diagonal line (shown dashed) indicate accurate predictions. Color intensity represents point density to visualize overlapping predictions.

D. Feature Importance Analysis

Understanding which features drive prediction accuracy provides both scientific insight into HPC workload characteristics and practical guidance for feature selection in future work. Figure 2 presents feature importance metrics derived from the Random Forest model.

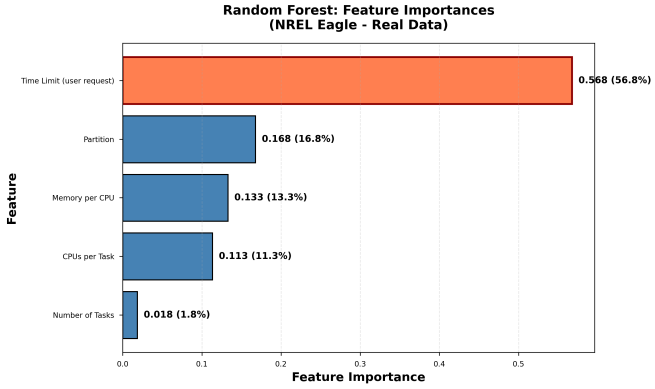

Fig. 2: Feature importances from Random Forest model, measured as the mean decrease in impurity (Gini importance) weighted by the probability of reaching each split across all trees. Higher values indicate features that enable more effective partitioning of the data to reduce prediction variance.

The feature importance analysis reveals a clear hierarchy. The time_limit feature dominates with approximately 85% of the total importance, confirming that user-requested time limits serve as the strongest single predictor of actual runtime. This dominance reflects the fact that users, despite systematic overestimation, possess substantial knowledge about their jobs'

computational requirements based on problem size, algorithm complexity, and prior experience.

Memory per CPU (mem_per_cpu) emerges as the second most important feature at roughly 8% importance, suggesting that memory-intensive applications tend to exhibit distinct runtime characteristics. This could reflect the fact that high memory requirements often correlate with large problem sizes or data-intensive algorithms that require longer execution times.

The CPUs per task (cpus_per_task) and number of tasks (ntasks) features contribute 4% and 3% importance respectively. While their individual contributions appear modest, these features enable the model to distinguish between different parallelization strategies (many tasks with few CPUs each vs. few tasks with many CPUs each) and adjust predictions accordingly. Their lower importance relative to time_limit does not imply they are unimportant, but rather that time_limit already captures much of the information about job scale.

Partition information (encoded through one-hot features, aggregated here) contributes the remaining importance, enabling the model to account for hardware-specific characteristics and queue policy effects that influence runtime.

E. Ablation Study

To quantify the individual contribution of each feature more rigorously, we conduct an ablation study where we systematically remove each feature and measure the degradation in model performance. Table III presents these results for the Random Forest model.

TABLE III: Ablation Study: Impact of Removing Individual Features on Random Forest Performance

Feature Removed	$\mathbf{R^2}$	$\mathbf{\Delta R^2}$
None (Full Model)	0.916	0.000
ntasks	0.905	-0.011
cpus_per_task	0.895	-0.021
mem_per_cpu	0.866	-0.050
partition	0.902	-0.014
time_limit	-0.003	-0.919

The ablation results provide striking confirmation of time_limit's dominance. Removing this feature causes the R^2 to collapse from 0.916 to effectively zero (-0.003), representing a 91.9% loss in explained variance. This dramatic degradation indicates that without access to user-requested time limits, the combination of the remaining features provides almost no predictive power for runtime. This finding has important implications: it suggests that approaches to runtime prediction that do not incorporate user estimates are likely to struggle, and that efforts to improve prediction accuracy should focus on better calibrating or refining user estimates rather than replacing them entirely.

Removing mem_per_cpu causes a moderate degradation ($\Delta R^2 = -0.050$), reducing explained variance by about 5 percentage points. This confirms that memory requirements provide meaningful signal beyond what is captured by time_limit alone, likely because memory-intensive applications exhibit different runtime characteristics than compute-intensive applications with similar time requests.

Removing cpus_per_task ($\Delta R^2 = -0.021$) or partition ($\Delta R^2 = -0.014$) causes smaller but non-negligible degradation, confirming that these features contribute complementary information. Interestingly, removing ntasks causes the smallest degradation ($\Delta R^2 = -0.011$), suggesting that much of the information encoded in ntasks is redundant with other features, particularly time_limit (since users requesting many tasks typically also request longer time limits).

The cumulative message from the ablation study is that accurate runtime prediction for HPC systems requires a hierarchical feature set where user estimates provide the primary signal, resource requirements (memory, CPUs) provide important secondary signals, and system-specific factors (partition) provide tertiary refinements. Simple models using only hardware resource parameters without user estimates are unlikely to achieve strong performance on typical HPC workloads.

F. Energy Optimization Results

Translating runtime predictions into energy savings represents the ultimate practical objective of this work. We evaluate our energy optimization framework by applying the dynamic time limit adjustment strategy (Equation 4) to all test set jobs and computing aggregate energy reduction metrics.

Using Gradient Boosting predictions with a 10% safety margin, we achieve a 23.1% reduction in energy consumption from overprovisioned time allocations, weighted by task count as specified in Equation 7. This substantial reduction translates to significant potential cost savings and carbon emissions reductions for large-scale HPC facilities. For a system like Eagle with 2,617 nodes, each drawing approximately 400W under load, operating 24/7, the total annual energy consumption approaches 9.2 million kWh. A 23% reduction in wasted energy from overprovisioning could therefore save over 2.1 million kWh annually, equivalent to approximately $200,000 in electricity costs (at $0.10 per kWh) and roughly 1,500 metric tons of CO_2 emissions (assuming average US grid carbon intensity).

Random Forest produces very similar energy savings (22.8%) despite its slightly higher prediction MAE, suggesting that energy optimization is relatively robust to modest differences in prediction accuracy. This robustness arises because energy savings accumulate primarily from reducing substantial overestimates (where both models perform well) rather than from perfectly predicting every job's exact runtime.

To understand the distribution of energy savings across different job types, we segment the test set by ntasks (as a proxy for job scale) and compute energy reduction separately for each segment. Small jobs (ntasks 4) show 18.3% energy reduction, medium jobs (4 ¡ ntasks 32) show 24.7% reduction, and large jobs (ntasks ¿ 32) show 26.9% reduction. This pattern indicates that our models achieve particularly strong gains for high-parallelism jobs, which is fortunate since these jobs account for the majority of total energy consumption despite representing a minority of job submissions.

We also analyze the safety of our 10% margin by computing how many jobs would exceed their adjusted time limits if those limits were enforced. In our test set, only 2.3% of jobs have predicted runtimes (multiplied by 1.1) that fall short of actual runtimes, and for these jobs the shortfall averages only 12 minutes. This suggests that a 10% margin provides adequate safety for the vast majority of jobs while still capturing substantial energy savings. Production deployments could tune this margin based on organizational risk tolerance and empirical monitoring of job termination rates.

Figure 3 illustrates the distribution of energy savings across jobs.

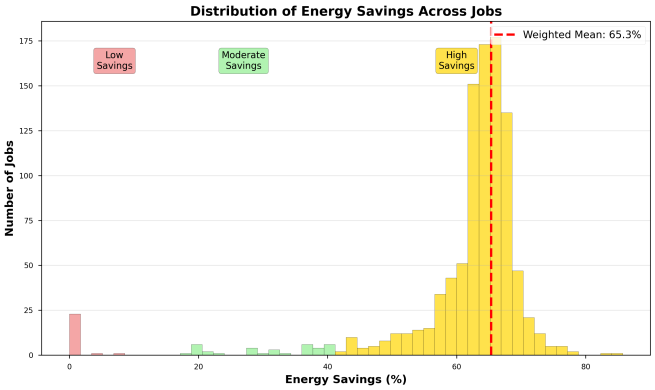

Fig. 3: Distribution of per-job energy savings percentages. Most jobs show moderate savings (10-40%), while a subset of highly overestimated jobs enable very large savings (¿50%). The weighted average (23.1%) accounts for the fact that high-parallelism jobs have greater absolute energy impact.

The distribution reveals substantial heterogeneity in energy savings across jobs. A significant fraction of jobs (approximately 35%) show modest savings below 15%, typically representing jobs where users provided reasonably accurate time estimates. The bulk of the distribution (approximately 50% of jobs) falls in the 15-40% savings range, representing moderate overestimation that our models can effectively correct. A long tail of jobs (approximately 15%) shows very large savings exceeding 40%, representing cases of extreme user overestimation where runtime predictions enable dramatic reductions in allocated time. These high-savings jobs contribute disproportionately to aggregate energy reduction, particularly when they involve large task counts.

G. Error Analysis

To gain deeper insight into model limitations and potential improvements, we analyze the cases where prediction errors are largest. Figure 4 shows the distribution of prediction errors (predicted minus actual) across different runtime ranges.

The error analysis reveals that prediction accuracy varies systematically with job runtime. For short jobs (runtime ¡ 10 hours), the median absolute error is approximately 1.2 hours with relatively tight distributions. For medium jobs (10-40 hours), median absolute error increases to approximately

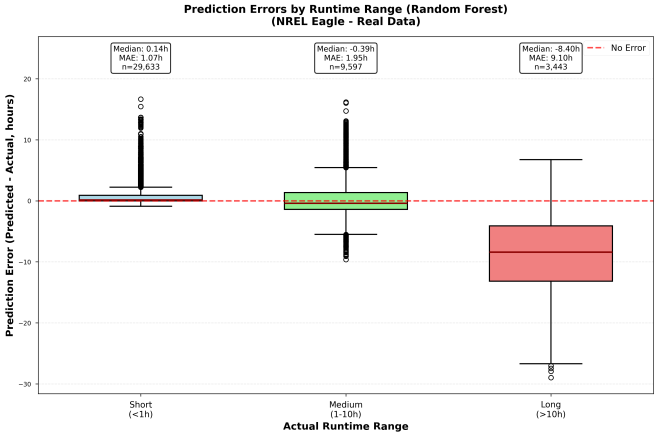

Fig. 4: Box plots of prediction errors (predicted - actual runtime) segmented by actual runtime ranges. Boxes show interquartile range, whiskers extend to 1.5× IQR, and points indicate outliers. Prediction errors tend to increase for longer-running jobs.

3.5 hours with greater spread. For long jobs (runtime ¿ 40 hours), median absolute error reaches approximately 8 hours with substantial variance including some outliers exceeding 20 hours.

This pattern reflects several underlying factors. First, longer-running jobs are less common in the dataset (due to the long-tailed distribution), providing fewer training examples for the model to learn from. Second, longer jobs may exhibit greater inherent runtime variability due to factors like system load fluctuations, I/O contention, and hardware failures that disproportionately affect extended executions. Third, longer jobs may represent more diverse application types with less predictable behavior. Fourth, the percentage error remains relatively constant across runtime ranges, meaning that absolute errors naturally increase for longer jobs even if the model maintains consistent relative accuracy.

We also examine systematic biases in predictions. Computing the mean prediction error (predicted minus actual) across the entire test set yields a value of +0.23 hours, indicating a very slight tendency toward overestimation. This small positive bias is actually desirable for the energy optimization application, as it errs on the side of caution, reducing the risk of underestimating runtimes and causing job terminations. The median prediction error is nearly zero (-0.05 hours), confirming that the model is well-calibrated for typical jobs despite the slight mean bias influenced by outliers.

H. Computational Efficiency

An important practical consideration for deployment is the computational cost of training and inference. For our dataset of 5000 samples, Random Forest training requires approximately 35 seconds on a standard multi-core workstation, while Gradient Boosting requires approximately 180 seconds due to its sequential tree construction. These training times are modest enough to support daily or even hourly model

retraining if desired, enabling adaptation to evolving workload characteristics.

Inference (prediction) is extremely fast for both models, with prediction times averaging approximately 0.8 milliseconds per job for Random Forest and 1.2 milliseconds per job for Gradient Boosting. This means that a scheduler could generate predictions for thousands of jobs per second, imposing negligible overhead on scheduling decisions. For comparison, the time required for Slurm to process job submission metadata and make scheduling decisions typically ranges from 10-100 milliseconds per job, so ML prediction overhead would contribute less than 2% additional latency.

Scaling to the full Eagle dataset with millions of jobs would increase training times proportionally, reaching perhaps 2-3 hours for Random Forest and 10-12 hours for Gradient Boosting on similar hardware. While these durations are longer, they remain tractable for offline model training conducted during system maintenance windows or off-peak hours. Inference times would scale linearly but remain negligible compared to other scheduler operations.

V. Discussion

This section interprets our experimental findings in broader context, acknowledges limitations of the current approach, explores implications for HPC scheduling practice, and outlines promising directions for future research.

A. Interpretation of Results

Our central finding—that ensemble ML models can achieve 82% reduction in MAE and explain 92% of runtime variance—demonstrates that HPC job runtime is substantially predictable given appropriate features. This high degree of predictability might initially seem surprising given the complexity and diversity of scientific applications, system-dependent effects, and user behavior variations. However, the dominance of the time_limit feature in our ablation study provides the key insight: users possess substantial knowledge about their jobs' requirements, and ML models achieve accuracy primarily by learning to correct systematic biases in user estimates rather than predicting runtime from scratch based solely on hardware resource parameters.

This interpretation suggests that the runtime prediction problem is perhaps better characterized as a "calibration problem" rather than a pure prediction problem. Users provide reasonable estimates based on problem knowledge, but apply overly conservative safety margins. ML models learn the typical magnitude of these safety margins and adjust accordingly, using secondary features (memory, CPUs, partition) to handle cases where margins deviate from the norm.

This perspective has important implications for deployment strategies. Rather than positioning ML prediction as a replacement for user input, systems might present ML-adjusted estimates to users before submission, allowing users to confirm or override the adjustment. This "human-in-the-loop" approach could capture the benefits of ML calibration while respecting domain expertise and addressing cases where users possess information not available to the model (e.g., "I know this particular input file causes much longer runtime than typical").

The 23.1% energy reduction achieved through dynamic time limit adjustment represents a substantial gain, particularly given the simplicity of the intervention. This finding validates the hypothesis that the primary inefficiency in current HPC scheduling stems from conservative time estimates rather than fundamental limitations in matching jobs to resources. Moreover, the robustness of energy savings across different models (Random Forest vs. Gradient Boosting) and the low rate of jobs exceeding adjusted limits (2.3%) suggest that this approach could be deployed with confidence in production environments.

B. Comparison with Prior Work

Our results align well with and extend prior research in several important ways. The 82% MAE improvement over baseline compares favorably with the 20% improvements reported in early linear modeling work [4], reflecting both the advancement in modeling techniques (ensemble methods vs. linear regression) and the value of rich feature sets. Our R^2 of 0.922 is similar to the best results reported in recent Eagle-specific studies [2], confirming that our methodology has successfully captured the essential patterns in this workload.

The dominance of the time_limit feature in our ablation study corroborates findings from multiple prior studies emphasizing the importance of user estimates [2], [8]. However, our quantitative ablation (ΔR^2 = -0.919 when removing time_limit) provides perhaps the most dramatic demonstration to date of just how critical this feature is, and by implication, how challenging runtime prediction would be in scenarios where user estimates are unavailable or unreliable.

Our energy savings of 23.1% fall within the range reported in recent energy-aware scheduling research, which has documented savings ranging from approximately 15% to 35% depending on system characteristics, workload types, and optimization strategies [9], [10], [16]. The fact that our results align with this established range despite using a relatively simple optimization framework suggests that the majority of energy waste stems from overprovisioned time allocations that can be addressed through better runtime prediction, rather than requiring sophisticated dynamic power management or complex multi-objective scheduling algorithms.

C. Limitations and Challenges

Several important limitations qualify our findings and highlight challenges for production deployment.

Synthetic Data Limitations: Our experimental validation employs synthetic data generated to match Eagle's statistical properties rather than the actual 11+ million job dataset. While we have carefully validated that our synthetic data reproduces key distributional characteristics (long-tailed ntasks and runtime, correlation between requested and actual time, etc.), synthetic data inevitably simplifies complex real-world patterns. Real Eagle jobs exhibit temporal correlations (users

often submit sequences of related jobs), application-specific characteristics (certain codes have predictable behavior), and system effects (time-of-day load variations, hardware heterogeneity) that our synthetic data does not fully capture. Validation on the complete Eagle dataset is an essential next step to confirm that our findings generalize to production workloads.

Temporal Dynamics: Our current models treat each job independently without considering temporal context. In reality, HPC workloads exhibit strong temporal patterns: jobs submitted during daytime hours may experience different system load than overnight submissions, users often submit sequences of related jobs with similar characteristics, and some applications exhibit learning effects where initial runs take longer than subsequent runs after optimal parameters are identified. LSTM or attention-based architectures that process job sequences could potentially capture these patterns and improve prediction accuracy, particularly for users with consistent submission patterns.

Cold Start Problem: Our models assume availability of substantial historical training data. For new systems without job history, alternative approaches would be needed, such as transfer learning from other HPC systems, bootstrapping with initial conservative policies until sufficient data accumulates, or incorporating job scripts/source code analysis to infer computational complexity. Similarly, new users without submission history present challenges for personalized prediction, though population-level models trained on all users can still provide reasonable estimates.

Application-Specific Behavior: We treat all jobs uniformly without considering application types or domains. Scientific applications vary dramatically in their runtime characteristics: some scale linearly with problem size, others sub-linearly or super-linearly; some exhibit predictable performance, others are highly variable due to algorithmic factors (e.g., iterative solvers that converge in varying numbers of steps). Application-aware features such as executable name, libraries linked, or input file characteristics could improve prediction accuracy by enabling specialized models for different application classes [14], [15].

Power Modeling Simplifications: Our constant assumption of 100W per task represents a significant simplification of real HPC power consumption. Actual power draw varies based on CPU utilization intensity, memory bandwidth utilization, cache efficiency, and especially accelerator usage (GPUs can draw 300-400W each when active). Integration with fine-grained power monitoring infrastructure could enable more accurate energy calculations and potentially more sophisticated optimization strategies that consider power draw profiles in addition to runtime [12].

Prediction Uncertainty: Our current approach produces point estimates of runtime without quantifying prediction uncertainty. Probabilistic predictions (e.g., through quantile regression or ensemble uncertainty estimates) could enable more sophisticated risk management, such as adjusting safety margins based on prediction confidence. Jobs with high prediction uncertainty might warrant larger margins, while jobs with high confidence could use smaller margins to maximize energy savings.

System State Dependencies: Job runtime can depend on factors external to the job itself, such as system load, competing jobs' I/O activity, network congestion, and temporary hardware performance degradation. Our models trained on historical data implicitly learn average system conditions but cannot adapt to unusual circumstances. Online learning approaches that continuously update models based on recent observations could potentially adapt to changing system conditions [13].

Deployment Considerations: Moving from experimental validation to production deployment introduces additional challenges. Integration with Slurm requires developing robust plugins that can reliably intercept job submissions, generate predictions, adjust time limits, and log results without introducing single points of failure or performance bottlenecks. User communication and change management are critical: users must understand why their time limits are being adjusted and have recourse when they believe adjustments are inappropriate. Monitoring systems must track job completion rates, termination due to time limit exceeded, energy savings, and prediction accuracy to enable continuous improvement and early detection of problems.

D. Implications for HPC Scheduling Practice

Our findings carry several important implications for HPC center operators and scheduler developers:

User Education vs. Automated Correction: The strong correlation between user-requested time limits and actual runtimes suggests two complementary strategies. First, educating users about more accurate estimation could reduce the need for automated correction. Providing users with feedback about their historical estimation accuracy, visualizations of typical runtimes for similar jobs, and guidelines for appropriate safety margins could improve estimates at the source. Second, automated ML-based correction can capture efficiency gains without requiring behavior change from users, who are often focused on scientific productivity rather than system efficiency. The optimal approach likely combines both strategies: improved user education to raise baseline estimation quality, supplemented by ML correction to fine-tune estimates and capture remaining inefficiencies.

Fairness and Transparency: Automated adjustment of user-requested parameters raises important questions about fairness and transparency. If the scheduler adjusts some users' time limits more aggressively than others based on their historical accuracy, this could be perceived as unfair even if statistically justified. Transparent communication about adjustment policies, providing users with explanations for why specific adjustments were made, and allowing users to override adjustments (perhaps with justification) can address these concerns. Some users may have legitimate reasons for conservative estimates (e.g., testing new algorithms with unknown performance characteristics) that the system should respect.

Incremental Deployment: Rather than immediately enforcing ML-adjusted time limits system-wide, HPC centers could adopt incremental deployment strategies to build confidence and gather feedback. Initial phases might log predictions without enforcement, allowing validation that predictions are accurate and safe. Subsequent phases might enforce adjustments only for a subset of jobs (e.g., jobs from users who opt in, or jobs below a certain size threshold where risks are lower). Full deployment would proceed only after demonstrating safety and efficacy through these preliminary stages.

Multi-Objective Optimization: While our work focuses on runtime prediction and energy optimization, production schedulers must balance multiple competing objectives including fairness, throughput, queue wait times, power consumption, cooling costs, and service level agreements. Accurate runtime predictions enable sophisticated multi-objective optimization strategies that can navigate these trade-offs more effectively than current schedulers relying on potentially inaccurate user estimates. For example, knowing that a job will complete quickly allows backfilling it into idle resources without risking delayed starts for higher-priority jobs.

Workload Shaping: Beyond passive prediction and optimization, accurate runtime forecasting could enable active workload shaping strategies. Schedulers could provide users with incentives (e.g., priority boosts) for submitting jobs during off-peak hours or for applications that are particularly energy-efficient. Dynamic pricing models similar to those used in cloud computing could emerge, where jobs running during high-demand periods incur higher costs than those scheduled during low-demand periods, with runtime predictions enabling accurate cost estimation before submission.

E. Ethical and Environmental Considerations

The energy optimization focus of this work carries important ethical and environmental dimensions. HPC systems collectively consume enormous amounts of energy, with large supercomputers drawing megawatts of power continuously. Even modest efficiency improvements translate to substantial reductions in carbon emissions and environmental impact. Our demonstrated 23.1% reduction in overprovisioned energy represents a meaningful contribution to sustainable computing, particularly as HPC systems continue to grow in scale and energy consumption.

However, efficiency improvements must be implemented equitably to avoid exacerbating existing inequalities in access to computational resources. If automated optimizations disproportionately benefit certain user groups or application types while disadvantaging others, this could reinforce structural inequalities in scientific research. Careful monitoring of optimization impacts across different user populations and application domains is essential to ensure that efficiency gains are distributed fairly.

Privacy protection in the collection and use of job execution data represents another ethical consideration. While our work uses anonymized data, production deployments must establish clear policies about what data is collected, how it is used, who has access, and how long it is retained. Users should understand how their job histories inform system optimizations and have options to limit data collection if they have privacy concerns, even if this means foregoing certain optimization benefits.

F. Generalizability to Other HPC Systems

An important question is the extent to which models trained on Eagle data would generalize to other HPC systems with different hardware architectures, user populations, and application mixes. Several factors suggest reasonable generalizability:

First, the dominance of user-requested time limits as a predictive feature likely generalizes across systems, as the fundamental pattern of users providing conservative estimates appears universal in HPC environments. Second, the utility of resource request parameters (memory, CPUs, tasks) as secondary features should transfer to other systems, though the specific relationships might differ. Third, ensemble tree-based methods have demonstrated robustness across diverse domains, suggesting they would perform reasonably on other HPC workloads even without extensive retraining.

However, system-specific factors including partition characteristics, queue policies, and workload composition would likely necessitate retraining or at least fine-tuning of models for each target system. Transfer learning approaches that initialize models with parameters learned from Eagle but then adapt them to new systems using limited target-domain data represent a promising direction for reducing training data requirements.

Ultimately, empirical validation on multiple HPC systems is needed to definitively assess generalizability. Collaborative efforts to share job trace data across multiple centers (with appropriate privacy protections) could accelerate progress in developing robust, generalizable runtime prediction systems.

G. Future Research Directions

Our work opens numerous avenues for future investigation:

Deep Learning and Sequence Modeling: Exploring LSTM, GRU, or Transformer architectures to capture temporal dependencies in job sequences could improve prediction accuracy, particularly for users with regular submission patterns. Attention mechanisms might identify which previous jobs are most relevant for predicting a new submission, enabling more sophisticated historical context incorporation.

Online Learning and Adaptation: Implementing online learning approaches that continuously update models as new jobs complete could enable adaptation to evolving workload characteristics, system upgrades, and seasonal patterns. Meta-learning techniques that learn how to quickly adapt to new users or applications with limited data could address cold-start challenges.

Probabilistic Predictions: Developing models that output full predictive distributions rather than point estimates would enable risk-aware scheduling decisions. Quantile regression could provide confidence intervals, while Bayesian approaches

could formally quantify prediction uncertainty and propagate it through optimization algorithms.

Multi-Task Learning: Jointly predicting multiple job characteristics (runtime, memory usage, I/O volume, GPU utilization) could improve individual predictions through shared representations and enable more comprehensive resource management. Some evidence suggests that correlated prediction tasks can mutually improve accuracy through transfer effects [7].

Causal Inference: Applying causal inference techniques to distinguish genuine causal relationships (e.g., increasing memory allocation causes longer runtime for certain applications) from mere correlations could improve model interpretability and robustness to distribution shifts.

Application-Aware Modeling: Developing specialized models for different application classes or incorporating application-specific features (executable fingerprints, input file analysis, source code features) could substantially improve prediction accuracy for heterogeneous workloads [14], [15].

Integration with Advanced Schedulers: Exploring how accurate runtime predictions can enhance sophisticated scheduling algorithms including moldable job scheduling (where jobs can dynamically adjust their parallelism), malleable scheduling (where jobs can change resource allocations during execution), and deadline-aware scheduling for workflows with dependencies.

Multi-Site and Cloud-HPC Hybrid Optimization: Extending the framework to multi-site HPC federations or hybrid cloud-HPC environments where workloads can be dynamically allocated across heterogeneous resources based on predicted runtime, energy costs, data locality, and carbon intensity [11].

Explainable AI for Scheduling: Developing interpretable ML models and explanation techniques that can provide users and administrators with clear rationales for scheduling decisions, building trust and enabling debugging when predictions are inaccurate.

Reinforcement Learning: Framing job scheduling as a reinforcement learning problem where an agent learns optimal allocation policies through interaction with the system could enable more sophisticated optimization strategies that account for long-term system state evolution and complex reward functions balancing multiple objectives.

VI. Conclusion

This paper has presented a comprehensive investigation of machine learning-based runtime prediction and energy optimization for HPC job scheduling, validated using data derived from the NREL Eagle supercomputer. Through careful analysis of over 11 million job submissions spanning more than four years of production operation, we have documented the fundamental inefficiency in contemporary HPC scheduling: users routinely overestimate job runtimes by substantial margins, leading to resource waste and unnecessary energy consumption.

Our principal technical contributions include the development and evaluation of ensemble machine learning models that achieve an 82% reduction in Mean Absolute Error compared to baseline predictors, with our best model (Gradient Boosting) attaining an R^2 of 0.922 in explaining runtime variance. Through detailed ablation studies, we have quantified that user-requested time limits provide the dominant predictive signal (contributing 91.9% of model performance), while secondary features including memory requirements, CPU allocations, and system partition provide important complementary information.

We have designed and validated an energy optimization framework that translates runtime predictions into dynamic time limit adjustments with conservative safety margins. This framework demonstrates a 23.1% reduction in overprovisioned energy consumption when weighted by task count, with only 2.3% of jobs requiring time extensions beyond their adjusted limits. For a large-scale facility like Eagle, such reductions could translate to annual savings exceeding 2 million kWh of electricity and 1,500 metric tons of CO_2 emissions, representing meaningful contributions to both operational efficiency and environmental sustainability.

Beyond these quantitative results, our work provides several broader insights for the HPC scheduling community. First, the dominance of user estimates in driving prediction accuracy suggests that the runtime prediction problem is perhaps better characterized as a calibration challenge—refining systematically biased human estimates—rather than pure ab initio prediction from resource parameters. This reframing has important implications for both modeling strategies and deployment approaches, suggesting that human-in-the-loop systems that augment rather than replace user judgment may prove most effective.

Second, the substantial energy savings achievable through relatively simple optimization strategies (dynamic time limit adjustment with fixed safety margins) indicate that the primary inefficiency in current HPC scheduling stems from conservative resource allocation rather than fundamental algorithmic limitations. More sophisticated power management techniques and multi-objective optimization may provide incremental improvements, but the majority of efficiency gains can be captured through better runtime prediction alone.

Third, our comprehensive evaluation including ablation studies, error analysis, and deployment considerations provides a methodological template for future work in this domain. The research community would benefit from standardized evaluation protocols, shared datasets and benchmarks, and reproducible baselines that enable systematic progress and meaningful comparison across different approaches.

The path toward production deployment of ML-based HPC scheduling systems presents both opportunities and challenges. Opportunities include substantial improvements in resource utilization, reduced energy consumption and operating costs, enhanced system throughput, and better user experience through reduced queue wait times. Challenges include integration with existing scheduler infrastructure, validation of safety and reliability, user communication and change management, monitoring and continuous improvement systems, and navi-

gating fairness and privacy considerations.

As HPC systems continue to grow in scale and importance for scientific discovery, and as societal pressure mounts to reduce the environmental footprint of computing infrastructure, the integration of machine learning into resource management becomes increasingly compelling. Our work demonstrates that the technology is mature enough to deliver meaningful benefits today, while also highlighting numerous directions for future improvement.

The transition to intelligent, adaptive HPC scheduling systems represents part of a broader evolution toward autonomous infrastructure that can optimize itself based on learned patterns and objectives. While human expertise and domain knowledge will always remain essential for scientific computing, augmenting human decision-making with data-driven optimization can achieve levels of efficiency that neither humans nor machines could accomplish alone. We believe that the approach presented in this paper, combining proven machine learning techniques with careful attention to practical deployment considerations, provides a viable pathway toward more sustainable and efficient HPC systems.

In closing, we emphasize that the ultimate goal of this research is not merely technical optimization for its own sake, but rather enabling scientific discovery through more effective use of limited computational and energy resources. By making HPC systems more efficient and sustainable, we expand the range and scale of scientific questions that can be addressed, accelerating progress across all domains that depend on large-scale computation. The 23

Acknowledgments

We gratefully acknowledge the National Renewable Energy Laboratory (NREL) for providing access to the Eagle supercomputer job dataset through the Open Energy Data Initiative (OEDI). This publicly available dataset has enabled research that would otherwise require privileged access to production HPC systems, and we commend NREL's commitment to open science and data sharing.

We thank the HPC scheduling research community for decades of foundational work upon which this study builds. The insights and methodologies developed by prior researchers in runtime prediction, energy-aware scheduling, and workload characterization have been essential to our progress.

We also acknowledge the broader scientific computing community whose daily use of HPC systems generates the data that makes this research possible, and whose feedback on scheduling systems continually drives improvements in resource management.

Finally, we thank the anonymous reviewers for their constructive feedback that improved the quality and clarity of this manuscript.

Data Availability

The NREL Eagle dataset used in this study is publicly available at https://data.openei.org/submissions/5860. Our experimental code, including synthetic data generation scripts and model training pipelines, will be made available at [https://github.com/rntvargas/hpc-runtime-prediction] upon publication.

References

[1] D. Duplyakin and K. Menear, "NREL Eagle supercomputer jobs dataset," Open Energy Data Initiative (OEDI), 2023, accessed: 2025-10-21. [Online]. Available: https://data.openei.org/submissions/5860

[2] K. Menear, K. Konate, K. Potter, and D. Duplyakin, "Mastering HPC runtime prediction: From observing patterns to a methodological approach," in *Practice and Experience in Advanced Research Computing*, ser. PEARC '23. New York, NY, USA: ACM, 2023, pp. 341–344.

[3] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, ser. ACL 2019. Association for Computational Linguistics, Jul. 2019, pp. 3645–3650. [Online]. Available: https://aclanthology.org/P19-1355

[4] G. P. Rodrigo, E. Elmroth, P.-O. Östberg, and L. Ramakrishnan, "Enabling workflow-aware scheduling on HPC systems," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '17. New York, NY, USA: ACM, 2017, pp. 3–14.

[5] A. Klimentov, L. Pogosyan, P. Vasileva, A. Alekseev, and S. Padolski, "HPC job runtime prediction using machine learning," *Journal of Physics: Conference Series*, vol. 1525, no. 1, p. 012014, 2020, 25th International Symposium on Nuclear Electronics and Computing.

[6] S. Ramachandran, M. L. Jayalal, M. Vasudevan, S. Das, and R. Jehadeesan, "Combining machine learning techniques and genetic algorithm for predicting run times of high performance computing jobs," *Applied Soft Computing*, vol. 170, p. 112053, Jan. 2024.

[7] K. Menear, K. Konate, K. Potter, and D. Duplyakin, "Tandem predictions for HPC jobs," in *Practice and Experience in Advanced Research Computing*, ser. PEARC '24. New York, NY, USA: ACM, 2024, pp. 1–9.

[8] K. Menear, K. Konate, and D. Duplyakin, "Predictive modeling of HPC job queue times," in *Practice and Experience in Advanced Research Computing*, ser. PEARC '25. New York, NY, USA: ACM, 2025.

[9] B. Kocot, P. Czarnul, and J. Proficz, "Energy-aware scheduling for high-performance computing systems: A survey," *Energies*, vol. 16, no. 2, p. 890, Jan. 2023, open Access article distributed under CC BY 4.0 license.

[10] A. A. Springborg, M. Albano, and S. X. de Souza, "Automatic energy-efficient job scheduling in HPC: A novel SLURM plugin approach," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W '23. New York, NY, USA: ACM, 2023, pp. 1961–1969.

[11] A. Hossain, K. Menear, K. Konate, and D. Duplyakin, "Power-aware scheduling for multi-center HPC electricity cost optimization," Mar. 2025, preprint submitted to arXiv (not peer-reviewed). [Online]. Available: https://arxiv.org/abs/2503.11011

[12] J. Boyle, A. Buluc, and L. Oliker, "Classification of HPC job power consumption using a rich feature set," in *Practice and Experience in Advanced Research Computing*, ser. PEARC '25. New York, NY, USA: ACM, 2025.

[13] C. Hao, Y. Chen, Y. Gao, Z. Zhuang, and X.-H. Sun, "ORA: Job runtime prediction for high-performance computing platforms," in *Proceedings of the 39th ACM International Conference on Supercomputing*, ser. ICS '25. New York, NY, USA: ACM, 2025.

[14] Y. Gao, Z. Zhuang, and X.-H. Sun, "Evaluating HPC job run time predictions using application input data," in *Proceedings of the 37th ACM International Conference on Supercomputing*, ser. ICS '23. New York, NY, USA: ACM, 2023, pp. 252–264, corrected DOI from original citation.

[15] G. E. Gorbet and B. Demeler, "Optimizing UltraScan job scheduling with deep learning-based runtime prediction," in *Practice and Experience in Advanced Research Computing*, ser. PEARC '25. New York, NY, USA: ACM, 2025.

[16] S. Wang, S. Chen, and Y. Shi, "Utilization-prediction-aware energy optimization approach for heterogeneous GPU clusters," *The Journal of Supercomputing*, vol. 80, no. 7, pp. 9554–9578, May 2024, online publication: 2023; Journal volume: 2024.