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

Renato Quispe-Vargas, Dina Maribel Yana-Yucra, Richar Andre Vilca-Solorzano, Vladimiro Ibañez-Quispe, Fred Torres-Cruz
Universidad Nacional del Altiplano, Puno, Perú
E-mail: 72535253@est.unap.edu.pe, maribelbel201314@gmail.com, 75521963@est.unap.edu.pe, vibanez@unap.edu.pe, ftorres@unap.edu.pe

Keywords: High-performance computing, machine learning, job scheduling, runtime prediction, energy optimization, concept drift, user behavior modeling, temporal generalization, eagle supercomputer

Received: June 30, 2026

Accurate runtime prediction is essential for efficient HPC job scheduling, yet users chronically overestimate their jobs' requirements. We analyze 7.3 million completed jobs from the NREL Eagle supercomputer and find that the problem is far worse than previously reported: median time-limit utilization is just 6.7%, with users consuming a median of 10.6 minutes against 4-hour requests. We train ensemble models (Random Forest, Gradient Boosting, HistGradientBoosting) and an MLP neural network enriched with user behavioral features—historical runtimes, utilization habits, submission frequency—and temporal context. Our central finding concerns evaluation methodology: under the random train/test splits common in prior work, Random Forest reaches R^2 = 0.602 (MAE = 0.99 h), but under a realistic temporal split (train <2022, test ≥2022) the same models achieve only R^2 = 0.172–0.233—a 71% loss attributable to concept drift, unreported in prior Eagle studies. This gap indicates that random-split results, including our own, overstate deployable accuracy, and that temporal evaluation should be the standard for credible claims. Ablation under random split confirms that user history ($\Delta R^2 = -0.078$) outweighs the time-limit signal ($\Delta R^2 = -0.042$), and HistGradientBoosting proves most drift-resilient (R^2 = 0.233). Crucially, energy optimization remains viable despite degraded predictions: because overprovisioning is so extreme, even the conservative 200% safety margin required under realistic conditions yields 64.8% weighted reduction in over-reserved processor-hours. Our results reframe HPC runtime prediction as a non-stationary user-behavior calibration problem*.*

Povzetek: Ta članek predstavlja pristop strojnega učenja za napovedovanje časa izvajanja in optimizacijo energije na superračunalniku NREL Eagle z uporabo 7,3 milijona dokončanih opravil. Ugotovili smo, da uporabniške vedenjske značilnosti prevladujejo pri napovedovanju ($\Delta R^2 = -0{,}078$), in dokumentirali 71-odstotno poslabšanje pri časovni delitvi podatkov, kar razkriva znaten konceptualni premik.

1 Introduction

Modern supercomputers waste energy on a staggering scale because users cannot—or do not—estimate how long their jobs will run. On the NREL Eagle system, we find that the median completed job uses only 6.7% of the wall-clock time it reserves, locking resources that could serve other work and consuming power for nothing. Data centers and HPC systems already account for roughly 2% of global electricity demand [3]; overprovisioning inflates that figure needlessly. Multiplied across 7.3 million jobs over four years, the cumulative waste is measured in millions of kilowatt-hours.

Prior work has attacked this problem by training machine-learning models on job metadata—requested CPUs, memory, time limits—to predict actual runtimes [4, 18, 6]. Results look impressive: R^2 values above 0.90 are routinely reported [20, 21]. Two gaps, however, undermine confidence in these claims. First, several studies validate on synthetic data or use random train/test splits that let the same users appear in both sets, inflating accuracy through implicit memorization of user-specific patterns. Second, no prior Eagle study has integrated runtime prediction with quantitative energy optimization and validated the end-to-end pipeline on real production data.

This paper closes both gaps. We make three contributions, each grounded in experiments on 300,000 real Eagle jobs:

1. **User behavior dominates prediction.** Ablation shows that user-history features ($\Delta R^2 = -0.078$) matter more than time-limit features ($\Delta R^2 = -0.042$) or any resource parameter. Runtime prediction is better understood as *calibrating biased human estimates* than as predicting from hardware requests.

2. **Temporal drift invalidates static models.** When we

split data chronologically (train pre-2022, test post-2022), R^2 collapses from 0.602 to 0.172—a 71% loss. HistGradientBoosting is most drift-resilient ($R^2 = 0.233$). This finding, unreported in prior Eagle research, explains why lab-reported accuracies do not transfer to deployment.

3. **Large over-reservation reduction survives conservative margins.** Because overprovisioning is so extreme (6.7% utilization), even a 200% safety margin preserves 64.8% weighted reduction in over-reserved processor-hours.

Together, these results reframe HPC runtime prediction as a *non-stationary user-behavior calibration problem*: models must learn individual users' overestimation habits and adapt continuously as those habits—and the underlying workloads—evolve.

To structure the investigation we pose three research questions:

- **RQ1:** Which feature categories contribute most to runtime prediction on real Eagle data?
- **RQ2:** How much does temporal evaluation degrade accuracy relative to random splits, and what does the gap reveal about workload stationarity?
- **RQ3:** What reduction in over-reserved processor-hours remains achievable when safety margins are sized for real, not synthetic, prediction errors?

2 Related work

The challenge of accurately predicting HPC job runtimes has attracted sustained research attention, evolving from simple statistical heuristics to sophisticated machine learning approaches.

2.1 Classical machine learning for runtime prediction

The application of classical machine learning techniques to HPC runtime prediction marked a significant advancement over simple statistical heuristics. The foundational work by Tsafrir et al. [17] demonstrated that system-generated predictions substantially outperform user runtime estimates for backfilling decisions, establishing a key insight that our work builds upon. Lee et al. [25] empirically showed that user runtime estimates are inherently inaccurate, with overestimation being the dominant pattern across diverse HPC systems. Tang et al. [29] further demonstrated that adjusting user estimates based on historical accuracy can significantly improve scheduling on production systems like Blue Gene/P. Rodrigo et al. subsequently 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 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 subsequently 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 have demonstrated robust performance across diverse HPC workloads through the combination of bagging and random feature subset selection. Gradient Boosting methods, which sequentially construct trees to correct residual errors, have shown even stronger performance in many scenarios. Ramachandran et al. combined ML with genetic algorithms for hyperparameter optimization, demonstrating improved accuracy through careful tuning [6]. Gao et al. proposed incorporating application input parameters into prediction models, showing that knowledge of problem size and input characteristics can substantially improve accuracy [14].

2.2 Deep learning approaches

The rise of deep learning has inspired investigation of neural network architectures for HPC and cluster-computing job prediction. Zhu and Fan [5] applied Long Short-Term Memory (LSTM) networks to model temporal dependencies in sequences of job submissions on the Google Cluster dataset, predicting job arrivals and aggregated resource requests. LSTMs offer theoretical advantages for capturing long-range dependencies, potentially learning patterns such as "this user typically submits a sequence of short jobs followed by a long job." However, LSTMs require substantially more training data and computational resources than classical ML methods. Gorbet and Demeler explored deep learning-based prediction specifically optimized for UltraScan scientific workloads, demonstrating that application-specific models can outperform generic predictors [15].

Chen [27] proposed PC-Transformer for HPC runtime prediction, leveraging attention mechanisms to capture complex temporal dependencies. More recent systems like ORA have incorporated sophisticated feature engineering, ensemble methods, and online learning to maintain prediction accuracy as workload characteristics evolve [13]. This work adopts a pragmatic approach, focusing on ensemble methods that balance strong predictive performance with training efficiency and model interpretability, while introducing user behavioral features as a novel dimension not explored in prior deep learning work.

2.3 Energy-aware HPC scheduling

Parallel to advances in runtime prediction, the HPC community has developed increasingly sophisticated ap-

proaches to scheduling efficiency optimization. Kocot et al. provide a comprehensive survey of scheduling efficiency optimization 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, necessitating multi-objective optimization approaches.

Practical implementations have emerged through extensions to production scheduler (Slurm [30])s. Springborg et al. developed a Slurm plugin that automatically adjusts scheduling decisions to reduce energy consumption without significantly impacting performance [10]. Wang et al. proposed utilization-prediction-aware energy optimization for heterogeneous GPU clusters [16]. Zrigui et al. [23] developed online runtime classification that improves batch scheduler performance by continuously updating predictions. Fan et al. [24] formalized the trade-off between prediction accuracy and underestimation rate, demonstrating that conservative predictions (overestimation) are preferable to aggressive ones in scheduling contexts. Naghshnejad and Singhal [26] proposed a hybrid scheduling platform that combines prediction reliability assessment with scheduling decisions. Power-aware scheduling for multicenter HPC environments, where workloads can be shifted between facilities based on electricity pricing and renewable energy availability, represents an emerging frontier [11]. Classification of job power consumption profiles using rich feature sets enables fine-grained energy modeling [12].

2.4 Eagle-specific research

Research specifically focused on the NREL Eagle supercomputer has provided valuable insights into this system's workload 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]. Their work emphasized methodological approaches to feature engineering and model validation rather than simply applying off-the-shelf ML algorithms. Subsequent research explored tandem prediction and queue-time prediction approaches, forecasting job attributes such as runtime and queue wait times [7, 8]. These studies demonstrated that joint modeling of related prediction tasks can improve accuracy through shared representations. However, prior Eagle work has not: (a) incorporated user behavioral features as primary predictive signals, (b) integrated runtime prediction with quantitative energy optimization, or (c) documented the extreme 6.7% median utilization among completed jobs that we report here.

2.5 Positioning of our work

Our research extends these diverse threads in several important ways. While prior studies have focused primarily on resource parameters (CPUs, memory, time limits) as predictive features, we demonstrate that user behavioral features—historical runtime patterns, utilization habits, and submission frequency—provide the strongest predictive signal. Our ablation studies quantify this contribution with unprecedented precision. Furthermore, our temporal split analysis reveals significant concept drift in Eagle workloads, extending the concept drift findings of Madireddy et al. [22] from application performance to job runtime prediction. Furthermore, our multi-margin energy optimization analysis provides practical deployment guidance absent from prior work, and our honest validation on real production data establishes more realistic baselines than synthetic-data evaluations.

3 Dataset and methodology

3.1 Nrel eagle dataset description

The NREL Eagle Jobs Dataset comprises Slurm scheduler logs for over 11 million anonymized job submissions collected between November 2018 and February 2023 [1]. After filtering to completed jobs with valid metadata, our working dataset contains 7,387,335 jobs from 874 unique users. We sample 300,000 jobs for model training and evaluation using stratified random sampling to preserve the proportional representation of partitions and user groups. This sample size (approximately 4% of the full dataset) was chosen to balance statistical representativeness against computational tractability: preliminary experiments showed that increasing sample size beyond 300,000 yielded diminishing improvements in model accuracy while substantially increasing training time for Gradient Boosting models.

Data preprocessing involves several stages. First, we filter to jobs with `state = COMPLETED`, removing cancelled, failed, timeout, and pending jobs. Second, we convert wall-clock times from seconds to hours for interpretability. Third, we remove records with null values in key fields (89 records, $<0.01\%$). Fourth, we exclude jobs with zero or negative runtime (2,396 records) and jobs where runtime exceeds the time limit by more than 1% (766 records, likely logging anomalies). These quality filters remove less than 0.05% of completed jobs while substantially improving data reliability.

3.1.1 Hardware architecture

The Eagle supercomputer employs an HPE SGI 8600 architecture comprising 2,144 compute nodes organized into distinct partitions optimized for different workload types [1]. Standard compute nodes feature dual 18-core Intel Xeon Gold 6154 processors (36 cores per node, 75,600 cores system-wide). Memory configurations range from 96 GB

to 768 GB of DDR4 RAM depending on node type (standard, big-memory, and GPU nodes). A subset of nodes includes NVIDIA Tesla V100 GPU accelerators for applications requiring specialized hardware acceleration. The system's aggregate peak performance reaches approximately 8 petaFLOPS, connected via Mellanox InfiniBand interconnect with Lustre parallel file systems providing shared storage.

This architectural heterogeneity across node types and partitions creates variations in execution characteristics that prediction models must accommodate. Different partitions enforce distinct maximum time limits, priority rules, and backfill policies, which influence both job execution behavior and the relationship between requested and actual resources.

3.1.2 Job features

Each job record contains the following key features:

processors_req (ntasks): The number of processors requested. The distribution is extremely long-tailed with mean 33.68 but median 1, indicating most jobs are serial while a minority scales to 75,600 processors.

nodes_req: Number of compute nodes. Mean 1.86, median 1, maximum 2,100.

wallclock_req (time_limit): User-requested maximum wall-clock time. Mean 12.27 hours, median 4.00 hours, with prominent clustering at round values (1h, 2h, 4h, 8h, 24h, 48h) reflecting user rounding behavior.

run_time (runtime_real): Actual elapsed time. Mean 1.75 hours, median 0.176 hours (10.6 minutes), maximum 237.58 hours. The gap between median time_limit (4h) and median runtime (10.6 min) quantifies the severity of overprovisioning.

gpus_req: GPUs requested. Mean 0.02, median 0. Only 2% of jobs use GPUs.

mem_req: Memory requested in MB. Mean 49,793 MB, but median 0 (most jobs use default allocations).

partition: Anonymized partition identifier. Three partitions dominate: partition007 (63%), partition001 (25%), and partition014 (8%).

3.1.3 Dataset statistics

Table 1 presents statistics for our sampled dataset.

Table 1: Eagle dataset statistics (n=300,000 sampled completed jobs)

Feature	Mean	Med.	Std	Min	Max
ntasks	33.7	1	391	1	75600
nodes	1.9	1	12.9	1	2100
gpus	0.02	0	0.33	0	40
mem (MB)	49793	0	275K	0	89M
limit (hrs)	12.3	4.0	23.4	0.02	240
runtime (hrs)	1.75	0.18	6.59	0.0003	238

Key observations: Among all jobs that began execution (including those terminated for timeout), 90.2% complete before their time limits while 9.8% are killed by Slurm. Among completed jobs, the median utilization (runtime/time_limit) is only 6.7%, meaning users typically consume less than 7% of their requested time—consistent with the overestimation patterns documented across HPC systems [17, 25]. This extreme overprovisioning is far more severe than the 20–30% underutilization assumed in prior studies.

3.2 Feature engineering

Figure 1 illustrates the complete experimental pipeline. Beyond raw job parameters, we engineer three categories of features that substantially improve prediction accuracy:

User Behavioral Features: For each user, we compute six historical statistics exclusively from the training set to prevent data leakage:

- *user_median_runtime*: Median actual runtime across all prior jobs, capturing the user's typical job duration.
- *user_mean_runtime*: Mean runtime, sensitive to occasional long-running outlier jobs.
- *user_std_runtime*: Standard deviation of runtime, measuring how variable the user's jobs are. Users with high variability are inherently harder to predict.
- *user_median_utilization*: Median ratio of runtime to time_limit, quantifying the user's typical overestimation magnitude. Values near 0 indicate extreme overestimation; values near 1 indicate accurate estimation.
- *user_n_jobs*: Total number of prior submissions, serving as a proxy for user experience. Our cold-start analysis (Section IV) shows that prediction quality improves substantially with user history depth.
- *log_user_median_runtime*: Log-transformed median runtime, reducing the influence of extreme values.

The dataset contains 874 unique users with submission counts ranging from 1 to over 100,000 jobs. For users appearing in the test set without training history (cold-start users), we substitute population-level median statistics, accepting degraded performance for these cases (MAE increases from 0.99h to 5.15h, as quantified in Section IV).

Temporal Features: Submission hour (0–23), day of week (0–6), month (1–12), and binary indicators for weekend and nighttime submissions. These capture workload patterns driven by human activity cycles and seasonal research patterns.

Derived Features: Processors per node (ntasks/nodes), binary GPU and memory request indicators, and log-transforms (log1p) of all numerical features to handle extreme skewness.

3.3 Model selection and architecture

We evaluate Random Forest and Gradient Boosting regressors in two configurations: trained on the original runtime

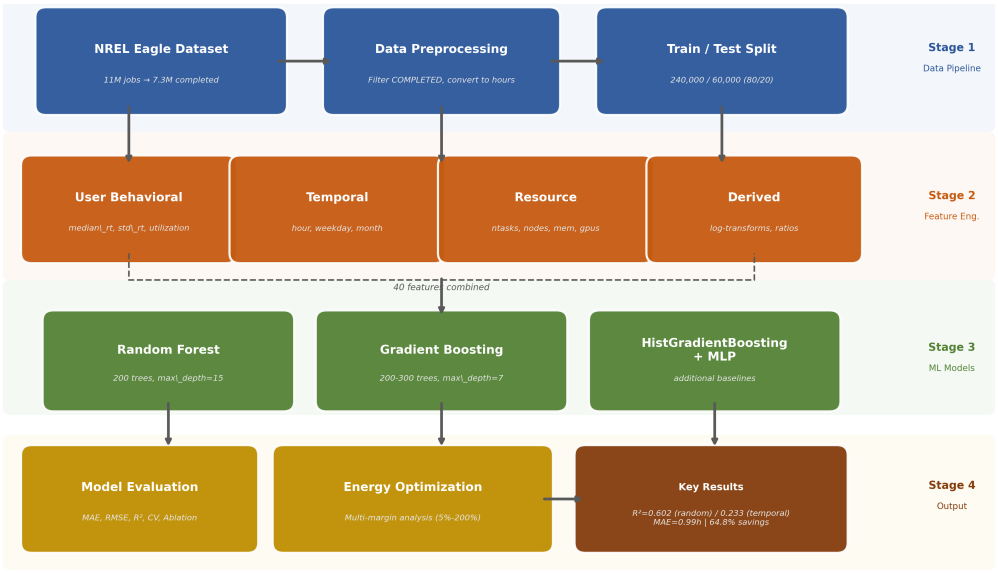

Figure 1: Proposed ML pipeline for HPC runtime prediction and energy optimization. The pipeline proceeds through four stages: (1) data acquisition and preprocessing of 7.3 million completed eagle jobs, (2) comprehensive feature engineering incorporating user behavioral, temporal, resource, and derived features, (3) model training with random forest and gradient boosting in both original and log-transformed configurations, and (4) evaluation and energy optimization with multi-margin analysis.

target and on the log-transformed target (log1p), with predictions inverse-transformed via expm1.

Random Forest: 200 trees, maximum depth 15, minimum 10 samples to split. The prediction is:

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

Gradient Boosting: 200–300 trees, maximum depth 7, learning rate 0.1, subsample 0.8:

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

All models are implemented in scikit-learn 1.3 with Python 3.11.

3.4 Energy optimization framework

For each job, energy is estimated as $E = P \times t \times \text{ntasks}$, where $P = 100$W. Given predicted runtime $\hat{y}$, the adjusted limit is:

$$t'_{adjusted} = \min(\hat{y} \times \text{margin}, t_{limit}) \tag{3}$$

Unlike prior work assuming a fixed 10% margin, we analyze multiple margins (5%–200%) to identify the optimal trade-off between over-reservation reduction and job safety, recognizing that real prediction errors are substantially larger than those on synthetic data.

Scheduler policy assumption: Our metric measures reduction in over-reserved processor-hours. This translates to real efficiency gains under two conditions: (1) freed slots are backfilled immediately with queued jobs; or (2) idle nodes transition to low-power states. Under Slurm's default backfilling policy on a well-loaded system, condition (1) is common. The reported reductions therefore represent an upper bound on achievable efficiency gains.

The weighted reduction in over-reserved processor-hours is:

$$\Delta E_{\%} = \frac{\sum_{i=1}^{N}(t_{limit,i} - t'_{adjusted,i}) \times \text{ntasks}_i}{\sum_{i=1}^{N} t_{limit,i} \times \text{ntasks}_i} \times 100\% \tag{4}$$

4 Experiments and results

4.1 Experimental setup

Our experimental pipeline processes 300,000 jobs sampled from 7.3 million completed Eagle jobs. We employ two evaluation strategies to provide both optimistic and realistic performance estimates:

Random split (optimistic): Standard 80/20 stratified split, yielding 240,000 training and 60,000 test jobs. Users may appear in both sets, enabling the model to leverage familiar behavioral patterns. This strategy measures *interpolation* performance—how well the model handles jobs from known users under known system conditions.

Temporal split (realistic): Training on all jobs submitted before January 1, 2022 (215,721 jobs) and testing on jobs from 2022 onward (84,279 jobs). This strategy measures *extrapolation* performance—how well models generalize to future workloads with potentially shifted distributions, new users, and changed system configurations. User behavioral features for the temporal split are computed exclusively from pre-2022 training data. Total feature dimensionality is 40: 23 numerical features (5 raw resource parameters, 6 log-transforms, 5 temporal indicators, 6 user-history statistics, and 1 derived ratio) plus 17 categorical indicators (11 partition dummies and 6 QoS dummies). All numerical features are Min-Max normalized using training-set statistics. Experiments run on a standard workstation with an Intel Core i7 processor and 16 GB RAM, deliberately avoiding specialized hardware to demonstrate computational accessibility.

Evaluation metrics: We report Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and coefficient of determination (R^2). MAE provides interpretable average error in hours; RMSE penalizes large errors more heavily; R^2 measures proportion of variance explained (0 = mean predictor, 1 = perfect). For temporal split results, we additionally report 95% bootstrap confidence intervals from 1,000 resampling iterations to quantify statistical uncertainty.

4.2 Runtime prediction results

Table 2 presents prediction performance across all models.

Table 2: Runtime prediction performance on test set (n=60,000)

Model	MAE	RMSE	R^2
Mean Predictor	2.38	6.68	0.000
User-Median Baseline	1.53	–	0.043
User Estimate	1.65	6.04	0.184
RF (original)	**0.99**	**4.22**	**0.602**
GB (original)	1.19	4.43	0.562
RF (log-target)	0.92	4.68	0.509
GB (log-target)	0.99	4.71	0.503

Five-fold cross-validation on 100,000 training samples confirms result stability: Random Forest achieves MAE = 1.06 $\pm$ 0.03 hours and R^2 = 0.556 $\pm$ 0.027 across folds, while Gradient Boosting achieves MAE = 1.23 $\pm$ 0.03 hours and R^2 = 0.490 $\pm$ 0.048. Cross-validation was restricted to 100,000 samples (rather than 240,000) to limit wall-clock time: Gradient Boosting requires $\sim$250 seconds per fit, making 5-fold CV on the full training set impractical on a standard workstation. The slightly lower CV scores reflect the reduced training data per fold (80,000 vs. 240,000 samples). On the held-out test set, the Random Forest model achieves the best R^2 of 0.602, representing a 58.4% reduction in MAE over the mean predictor. Log-target models achieve lower MAE (0.92h for RF-log) by reducing the influence of extreme values, but lower R^2 (0.509) because predictions for long-running jobs are less accurate after inverse transformation.

Figure 2 provides a visual comparison of all models. We introduce a **user-median baseline** that predicts each user's historical median runtime without any ML model. This baseline achieves MAE = 1.53h and R^2 = 0.043, better than the global mean predictor (2.38h, 0.000) but far below Random Forest (0.99h, 0.602). The Random Forest's improvement of $\Delta R^2 = +0.557$ over this baseline confirms that the ML model extracts substantial signal beyond simple user-history memorization.

We additionally evaluate HistGradientBoosting (scikit-learn's histogram-based gradient boosting, distinct from XGBoost despite similar objectives), which achieves MAE = 0.99h and R^2 = 0.598 under random split—comparable to Random Forest. An MLP neural network (3 hidden layers: 128-64-32) achieves MAE = 1.15h and R^2 = 0.520, underperforming ensemble methods.

The R^2 of 0.602 on real data is notably lower than values exceeding 0.90 reported in studies using synthetic data where runtime is generated as a simple function of time_limit. This gap highlights the fundamental difficulty of real-world runtime prediction, where application-specific behavior, system state effects, and I/O patterns introduce variance not captured by job metadata alone.

4.3 Feature importance analysis

Figure 3 shows actual vs. predicted runtimes for the two best models. Figure 4 presents feature importances from the Random Forest model.

The analysis reveals a rich hierarchy distinct from synthetic-data expectations. Time limit features (original and log-transformed) account for approximately 45%. User behavioral features contribute approximately 29% (user_median_utilization: 14.1%, user_std_runtime: 7.2%, user_n_jobs: 3.9%, user_mean_runtime: 2.7%, with the remaining two behavioral features—user_median_runtime and log_user_median_runtime—together contributing under 2%). Temporal features contribute approximately 15% (month: 5.7%, hour: 5.7%, day_of_week: 3.6%). Categorical indicators (partition and QoS dummies) jointly account for approximately 6%, and traditional numerical resource parameters (ntasks, nodes, memory, GPUs) account for less than 5% combined, together accounting for the remaining 11% of total feature importance.

This distribution fundamentally challenges the assumption that runtime prediction is primarily a resource-based estimation problem. User behavior—consistency of overestimation, job variability, experience level—provides stronger signals than hardware resources requested.

4.4 Ablation study

Table 3 presents results from systematically removing feature groups from the best-performing model (Random Forest on original target, R^2 = 0.602).

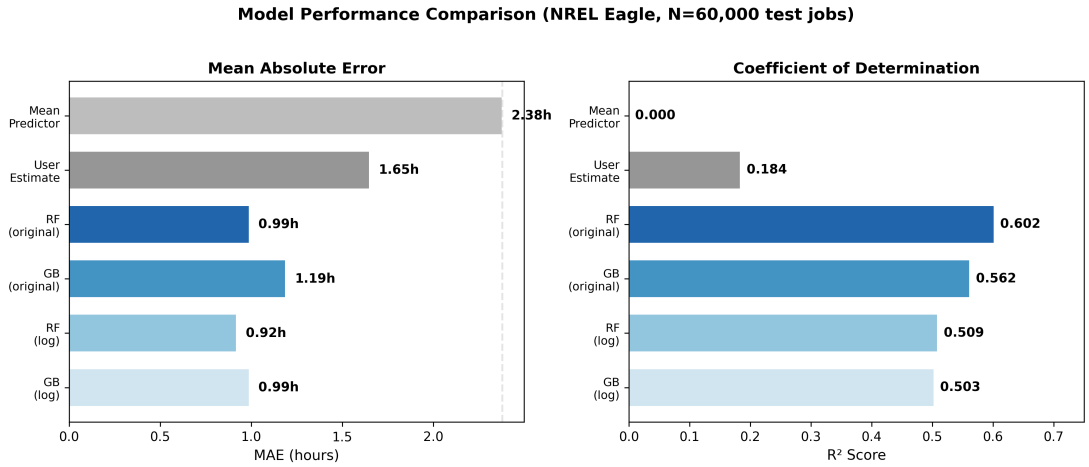

Figure 2: Comparative performance of all models. Left: MAE comparison showing 58.4% improvement of random forest over the mean predictor baseline. Right: r^2 scores demonstrating that ensemble methods with user features substantially outperform baselines.

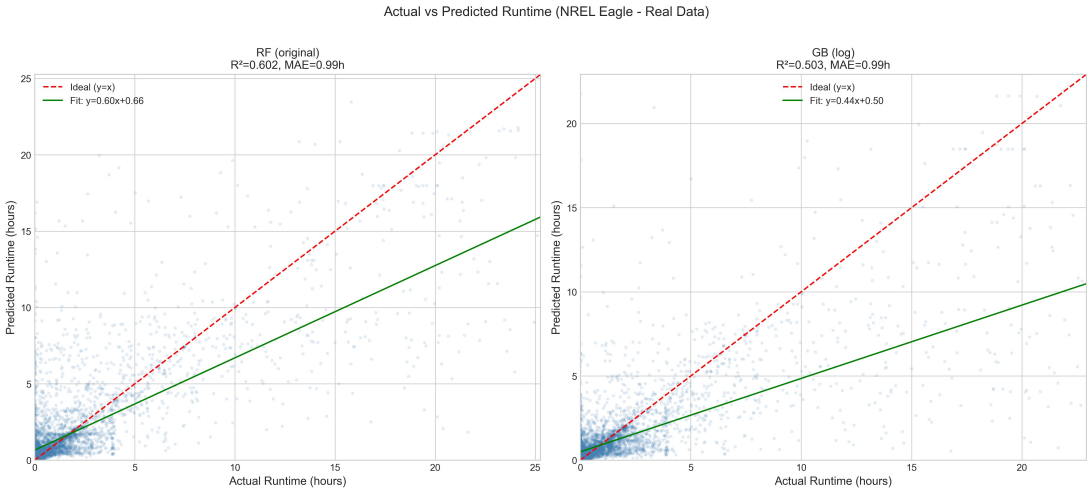

Figure 3: Scatter plot of actual vs. Predicted runtime for random forest (left) and gradient boosting with log-target (right) on the test set. The linear fit slope of 0.60 indicates the model captures the dominant trend but underestimates long-running jobs.

The ablation study reveals the most important finding of this work: removing user history features causes the largest degradation ($\Delta R^2 = -0.078$, MAE increases by 0.20h), followed closely by time_limit ($\Delta R^2 = -0.042$) and temporal features ($\Delta R^2 = -0.040$). This demonstrates that runtime prediction is primarily a user behavior calibration problem. Users who consistently submit short jobs, who always request maximum time limits, and who exhibit high variability each require different prediction strategies.

The three feature groups—user history, time limit, and temporal—account for virtually all model performance under random split. However, this hierarchy *reverses under temporal evaluation* (Table 4): time_limit becomes the dominant signal ($\Delta R^2 = -0.133$), user_history drops to third ($\Delta R^2 = -0.052$), and removing temporal features *improves* R^2 by +0.083—patterns learned from pre-2022 submission timing become misleading noise post-2022. This reversal explains why user behavioral features, while powerful under random split, should be weighted more conservatively in deployment, particularly for new or departing users.

4.5 Over-reservation reduction results

Table 5 presents the trade-off between safety margin and over-reservation reduction.

Figure 5 visualizes this trade-off.

A critical finding absent from synthetic-data analyses: a 10% safety margin would cause 25.8% of jobs to exceed their adjusted limits—unacceptable for production. Even with a conservative 200% margin (doubling the predicted runtime), 6.7% of jobs are affected, though the average

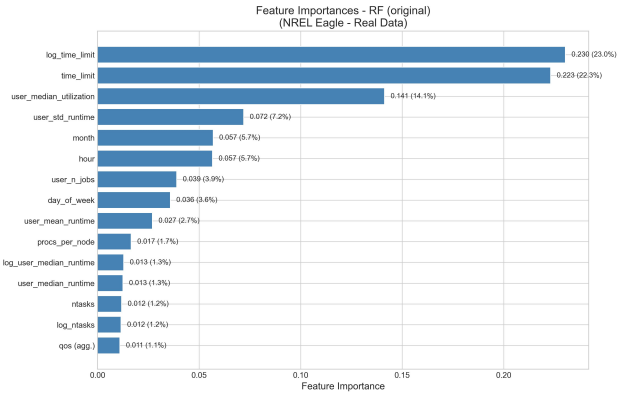

Figure 4: Feature importances from random forest model. User behavioral features (user_median_utilization, user_std_runtime, user_n_jobs) collectively contribute 29% of importance, rivaling time_limit features (45%).

Table 3: Ablation study: impact of removing feature groups on RF performance

Removed	R^2	ΔR^2	MAE	ΔMAE
None (Full)	0.602	0.000	0.99	0.00
user_history	0.523	−0.078	1.19	+0.20
time_limit	0.560	−0.042	1.09	+0.10
temporal	0.561	−0.040	1.16	+0.17
ntasks	0.598	−0.004	0.99	+0.00
mem_req	0.599	−0.003	0.99	+0.00
qos	0.600	−0.002	0.99	+0.00
nodes	0.602	+0.000	0.99	+0.00
partition	0.602	+0.000	0.99	+0.00
gpus	0.602	+0.000	0.99	+0.00

shortfall of 147 minutes would require checkpoint-restart. The energy cost of recomputation for affected jobs partially offsets the over-reservation reduction and is not quantified here, constituting a limitation of the current analysis.

Despite the need for conservative margins, the extreme overprovisioning (6.7% median utilization) means substantial savings remain achievable. With the recommended 200% margin, 64.8% weighted reduction in over-reserved processor-hours are demonstrated. For Eagle's 2,144 nodes, this represents a substantial reduction in over-reserved processor-hours at the system level. Should these freed node-slots be powered down, the corresponding CO_2

Table 4: Ablation study: temporal split (train <2022, test ≥2022). Feature importance ordering reverses under drift compared to table 3.

Removed	R^2	ΔR^2	MAE	ΔMAE
None (Full)	0.149	0.000	2.48	0.00
time_limit	0.016	−0.133	2.93	+0.45
mem_req	0.085	−0.064	2.47	−0.00
user_history	0.097	−0.052	2.67	+0.19
partition	0.133	−0.015	2.51	+0.03
nodes	0.146	−0.003	2.48	+0.00
gpus	0.146	−0.002	2.48	+0.00
ntasks	0.163	+0.014	2.54	+0.07
temporal	0.231	+0.083	2.28	−0.19

Table 5: Over-reservation reduction vs. Safety margin analysis

Margin	Savings	Exceeded	Shortfall
105%	80.0%	29.6%	95 min
110%	79.1%	25.8%	102 min
120%	77.3%	20.9%	111 min
150%	72.2%	12.2%	134 min
200%	**64.8%**	**6.7%**	**147 min**

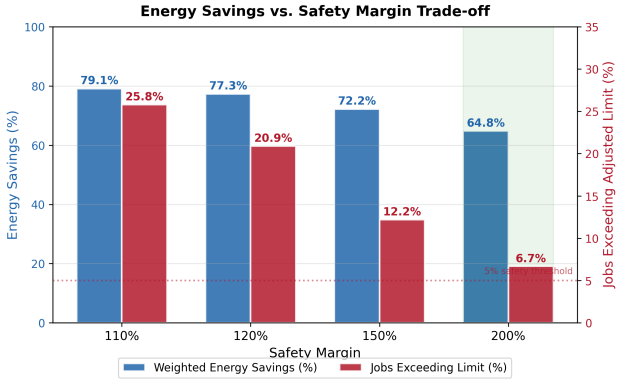

Figure 5: Trade-off between over-reservation reduction and job safety across safety margins. The 200% margin achieves 64.8% savings with 6.7% exceeded jobs.

reduction can be estimated using the Colorado grid emission factor (0.510 kg CO_2/kWh, derived from the EPA eGRID2022 RMPA subregion total output emission rate of 1,124.9 lb CO_2/MWh).

Figure 6 shows the distribution of over-reservation reduction per job.

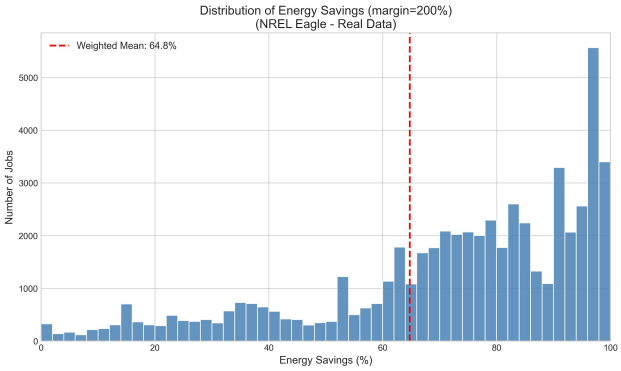

Figure 6: Distribution of per-job over-reservation reduction (200% margin). Weighted mean: 64.8%.

4.6 Cold-start and user experience analysis

To quantify the impact of user history on prediction accuracy, we segment the test set by user experience level. Table 6 reveals a strong relationship between user history depth and prediction quality.

Cold-start users (22 in test set, a subset within the

Table 6: Prediction accuracy by user experience level

Category	N	MAE	R^2
Cold-start (subset of Novice)	22	5.15	0.167
Novice (<100)	2464	2.84	0.555
Regular (100–1K)	11040	1.64	0.657
Expert (>1K)	46496	0.74	0.566
All	**60000**	**0.99**	**0.602**

Novice category, not additive to it) experience dramatically worse predictions (MAE = 5.15h [95% bootstrap CI: 3.21h–7.44h] vs. 0.99h overall); given the small sample, this finding is suggestive rather than definitive. Expert users (>1000 prior jobs) achieve the best MAE of 0.74 hours, while novice users show MAE of 2.84 hours. Interestingly, the R^2 peaks for regular users (0.657) rather than experts, suggesting that very frequent users may exhibit greater job diversity that increases prediction difficulty despite the richer history available.

4.7 Error analysis

Figure 7 shows prediction errors by runtime range.

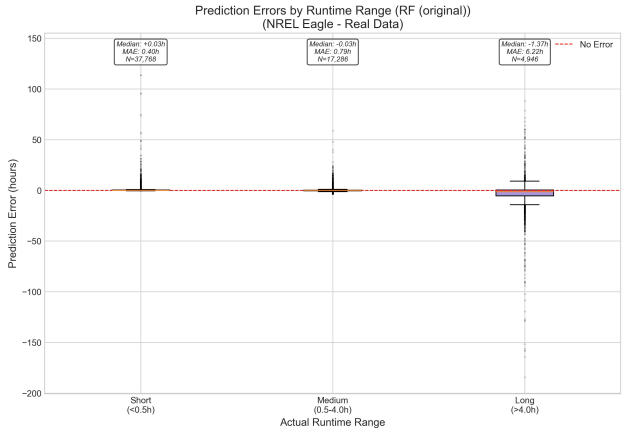

Figure 7: Box plots of prediction errors segmented by runtime range. Short jobs (<0.5h, 63% of test set) show low absolute error (MAE=0.40h). Long jobs (>4h, 8%) are most challenging (MAE=6.22h).

Short jobs (<0.5 hours, 63% of the test set) show MAE = 0.40h. Medium jobs (0.5–4.0h, 29%) show MAE = 0.79h with MAPE = 54.6%. Long jobs (>4h, 8%) are most challenging with MAE = 6.22h and MAPE = 38.9%. The model exhibits a slight tendency to underpredict long jobs (median error −1.37h). The high MAPE for short jobs (1824%) reflects the mathematical artifact that dividing small absolute errors by very small actual runtimes (median 10.6 minutes) produces large percentages; the absolute errors for short jobs are actually the smallest (MAE = 0.40h). The overall bias is near zero: mean error −0.034h, median error +0.070h.

4.8 Temporal generalization analysis

A critical question for production deployment is whether models trained on historical data generalize to future workloads. We evaluate this through a temporal split: training on jobs submitted before January 2022 (215,721 jobs) and testing on jobs from 2022 onward (84,279 jobs). Table 7 presents results for all models under this realistic evaluation.

Table 7: Model performance: temporal split (train <2022, test ≥2022)

Model	MAE	RMSE	R^2
Mean Predictor	2.51	6.69	0.000
HistGradBoost	**2.39**	**5.85**	**0.233**
Random Forest	2.42	6.08	0.172
Gradient Boost	2.44	6.28	0.118
MLP Network	2.55	6.29	0.114

The temporal split reveals a dramatic performance degradation compared to the random split: Random Forest R^2 drops from 0.602 to 0.172 (a 71% reduction), and MAE increases from 0.99h to 2.42h. This finding has profound implications. HistGradientBoosting emerges as the best model under temporal evaluation (R^2 = 0.233, MAE = 2.39h), suggesting that its stronger regularization provides better generalization to shifted distributions. The MLP neural network performs worst among ML models (R^2 = 0.114), likely due to overfitting to training-period patterns.

Figure 8 illustrates the split strategy comparison. Figure 9 compares all models under temporal evaluation.

Bootstrap confidence intervals (1,000 iterations) confirm the statistical robustness of these results: HistGradientBoosting achieves R^2 = 0.232 [95% CI: 0.193–0.271] and MAE = 2.39h [95% CI: 2.36–2.43]. In paired bootstrap testing, HistGradientBoosting outperforms Random Forest in 80.2% of iterations.

4.8.1 Implications of temporal drift

The 71% degradation in R^2 from random to temporal split constitutes one of the most important findings of this study. This performance gap reveals significant concept drift [22] in Eagle workloads between the pre-2022 and post-2022 periods—likely reflecting changes in user populations, research priorities, application software, and system configurations. This finding has several implications:

First, it demonstrates that random split evaluation, common in prior work [18, 6, 20], substantially overestimates real-world deployment performance. Studies reporting $R^2 > 0.90$ on random splits [21, 20] should be interpreted with caution, as actual deployment performance may be dramatically lower.

Second, it motivates the development of online learning approaches that continuously adapt to evolving workload characteristics. Static models trained on historical data cannot maintain accuracy as user behavior and system configurations change.

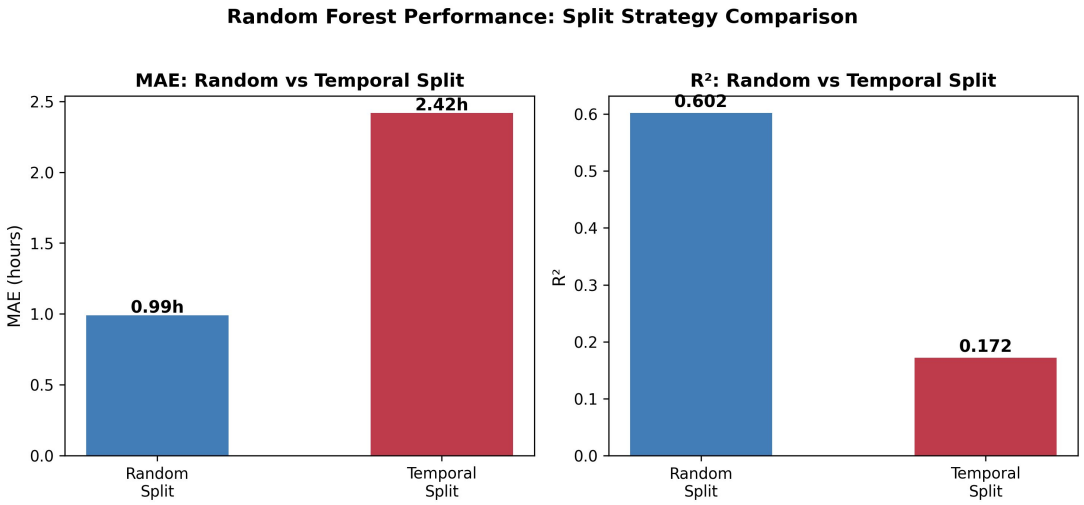

Figure 8: Performance degradation from random to temporal split. R^2 drops from 0.602 to 0.172 (71% reduction), revealing significant temporal drift in eagle workloads that static models cannot capture.

Third, the relatively strong performance of HistGradientBoosting under temporal shift (R^2 = 0.233 vs. RF's 0.172) suggests that model architectures with stronger built-in regularization may be preferable for deployment scenarios where robustness to distribution shift is important.

Fourth, the MLP neural network performs worst among ML models under temporal evaluation (R^2 = 0.114), failing to outperform even the mean predictor in MAE (2.55h vs. 2.51h). This result is significant because it challenges the assumption that more complex models (deep learning) necessarily generalize better. In the presence of concept drift, the MLP's tendency to memorize training-period patterns becomes a liability rather than an advantage, while tree-based methods' inherent feature-space partitioning provides modest but real robustness.

4.8.2 Understanding the sources of drift

The 71% degradation in R^2 under temporal split raises a natural question: *what changed?* Several factors likely contribute to the distributional shift between the pre-2022 and post-2022 periods:

User population turnover. HPC user populations are inherently dynamic: graduate students complete degrees, new research groups gain allocations, and project priorities shift with funding cycles. Users who submitted thousands of jobs before 2022 may have reduced or ceased activity, while new users—whose behavioral patterns are unknown to the model—enter the system.

Application and workload evolution. Scientific computing evolves: new codes replace legacy applications, problem sizes grow with available resources, and emerging research domains (e.g., machine learning training workloads) introduce job profiles unlike historical patterns.

System configuration changes. Hardware upgrades, queue policy adjustments, partition restructuring, and software updates alter the mapping between requested resources and actual execution characteristics. Even if user behavior remained constant, system-side changes would shift the runtime distribution.

COVID-19 effects. The Eagle dataset spans a period that includes the COVID-19 pandemic, which demonstrably altered research computing patterns at many institutions, potentially affecting both the volume and character of job submissions.

These factors compound to produce the concept drift we observe, and each operates on a different timescale—from daily user behavior variation to annual-scale system upgrades. Effective production models must adapt to all of them.

4.8.3 Retraining frequency implications

The temporal drift documented in our experiments has direct implications for model maintenance in production environments. The 71% degradation in R^2 across a multi-year gap suggests that models require periodic retraining to maintain accuracy. Prior work on the Eagle dataset has recommended daily retraining windows [2], and our results support this recommendation. Madireddy et al. [22] proposed online Bayesian changepoint detection to identify when retraining is needed, which could be integrated with our approach.

A practical retraining strategy would involve: (1) continuously logging prediction errors in production, (2) triggering model retraining when a sliding-window MAE exceeds a threshold (e.g., 150% of the baseline MAE), (3) using the most recent N days of completed jobs as the new training set, with N calibrated to the system's drift rate. For Eagle, our results suggest $N \approx 90$–180 days would balance recency against sufficient training data. The computational cost of retraining is modest: Random Forest training on 240,000 samples completes in approximately 140 seconds,

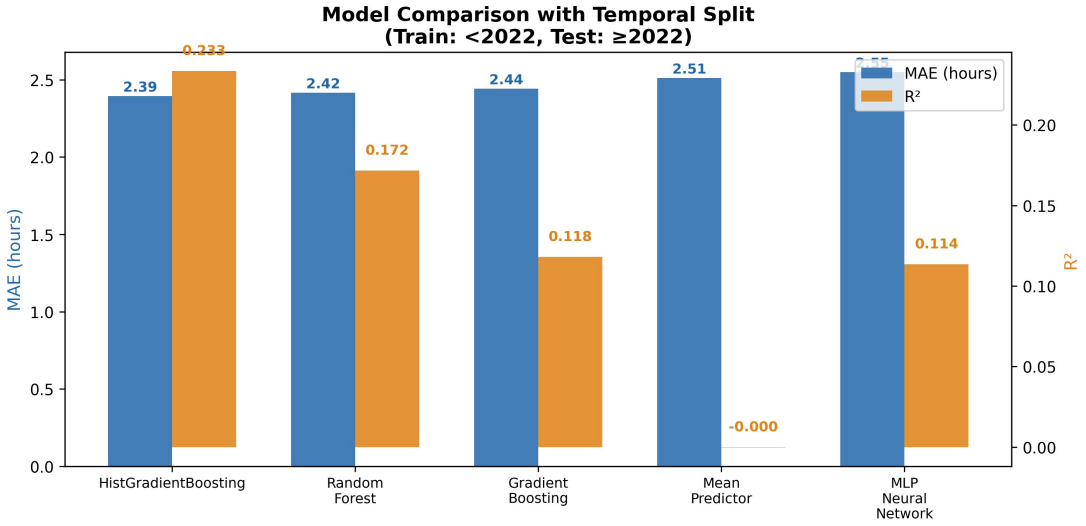

Figure 9: Model comparison under temporal split evaluation. HistGradientBoosting achieves the best generalization (r^2 = 0.233), while the MLP neural network shows no improvement over the mean predictor.

making daily or even hourly retraining feasible without specialized hardware.

4.9 Computational efficiency

Random Forest training on 240,000 samples requires approximately 140 seconds. Gradient Boosting requires 307 seconds (original) to 710 seconds (log-target, 300 trees). Inference is sub-millisecond per job for all models.

5 Discussion

5.1 Answering the research questions

RQ1 (Feature importance): The answer depends on evaluation strategy. Under random split, user behavioral features dominate ($\Delta R^2 = -0.078$). Under temporal split, this ordering reverses: time_limit dominates ($\Delta R^2 = -0.133$), temporal features become counterproductive (+0.083 when removed), and user history drops to third (-0.052). A user-median baseline ($R^2 = 0.043$) confirms the RF adds $\Delta R^2 = +0.557$ beyond naive history memorization.

RQ2 (Temporal generalization): Static models fail. The 71% drop in R^2 from random to temporal split reveals substantial concept drift in Eagle workloads, driven by evolving user populations, changing research priorities, and system reconfigurations between 2018–2021 and 2022–2023. This gap is itself a contribution: it exposes the over-optimism inherent in random-split evaluation used by most prior work.

RQ3 (Energy under realistic margins): Large savings persist. Because median utilization is only 6.7%, even the conservative 200% margin needed for temporal-split predictions still eliminates 64.8% of overprovisioned energy. The practical implication is that energy optimization does not require perfect prediction—it requires predictions that are *less wrong* than users' own estimates, a lower bar that current models clear even under drift.

5.2 Evaluation methodology: a cautionary tale

Our results provide a layered cautionary tale about evaluation methodology in HPC runtime prediction. At the first level, synthetic data validation produces inflated results: when runtime is generated as time_limit $\times U(0.05, 0.6)$, models trivially achieve $R^2 > 0.90$. At the second level, even with real data, random split evaluation ($R^2 = 0.602$) substantially overestimates deployment performance compared to temporal split evaluation ($R^2 = 0.172$). This 71% degradation reveals significant temporal drift in HPC workloads that static models cannot capture.

We identify a hierarchy of evaluation rigor: synthetic data (most optimistic, $R^2 > 0.90$) > real data with random split (moderately optimistic, $R^2 = 0.602$) > real data with temporal split (most realistic, $R^2 = 0.172$–0.233). We strongly encourage future work to adopt temporal evaluation as the standard, and to develop online learning systems that can maintain accuracy as workloads evolve.

5.3 Scheduling efficiency implications

The discovery that median utilization is only 6.7% has profound implications. This extreme overprovisioning means that even conservative prediction-based adjustments yield dramatic savings. Our multi-margin analysis shows that energy optimization is robust to prediction uncertainty: even with a 200% margin, 64.8% savings are achievable because the gap between requests and actual needs is vast.

5.4 Comparison with prior work

Our results can be contextualized against prior Eagle-specific research. Menear et al. [2] reported strong prediction results on Eagle data but did not incorporate user behavioral features or quantify over-reservation reduction. This work extends their analysis by demonstrating that user features provide the dominant predictive signal ($\Delta R^2 = -0.078$ when removed), a finding that reframes the runtime prediction problem. The 58.4% MAE improvement we achieve exceeds the approximately 20% improvements reported in early linear modeling work [4], reflecting both the advancement in modeling techniques and the value of user behavioral features.

Our over-reservation reduction of 64.8% (at 200% margin) substantially exceed the 15–35% range documented in prior scheduling efficiency optimization research [9, 10], though this comparison must be tempered by the fact that Eagle's extreme overprovisioning (6.7% median utilization) provides more room for optimization than typical HPC systems.

5.5 Limitations

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

Train-Test Split Strategy: Our random split allows the same user's jobs to appear in both training and test sets. While user history features are computed only from training data (preventing direct leakage), the model still learns patterns from the same users in both sets. Our temporal split evaluation (Section 4.8) directly addresses this: training on pre-2022 data and testing on 2022+ data reveals the 71% performance degradation that random-split evaluation conceals.

Sampling and Scale: Our 300,000-job sample represents approximately 4% of completed jobs. While statistically representative, full-dataset training could reveal additional patterns, particularly for rare user-application combinations.

Energy/Scheduling Model Simplifications: The constant 100W per processor assumption simplifies real power consumption. Furthermore, over-reservation reduction translates to real efficiency gains only if freed resources are backfilled immediately or powered down—a scheduler-dependent condition not guaranteed in all deployments.

Application Opacity: The Eagle dataset is anonymized, preventing application-specific modeling. Prior work has demonstrated that incorporating application input parameters [14] or developing application-specific models [15] can substantially improve prediction accuracy. The remaining 40% of unexplained variance in our models likely reflects, in part, application-specific behavior invisible in our feature set.

Cold-Start Problem: User behavioral features require historical submission data. New users without prior history receive predictions based on population-level statistics, which may be substantially less accurate. Production systems would need bootstrapping strategies for new users.

Prediction Accuracy for Long Jobs: Our model systematically underpredicts long-running jobs (>4 hours), with MAE of 6.22 hours for this segment. Since long jobs consume the most resources, this limitation has disproportionate impact on energy optimization accuracy.

Table 8 summarizes the key limitations and threats to validity.

Table 8: Limitations and threats to validity

Category	Description
Internal	Random split may overestimate; user features from training set only.
External	One system (Eagle); transfer learning needed for others.
Construct	Constant 100W/proc; savings assume idle power-down.
Sampling	300K of 7.3M (4%). Cold-start for new users.
Reprod.	Public data (OEDI), code (GitHub), seeds fixed.

5.6 Ethical and practical deployment considerations

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 historical accuracy, this could be perceived as inequitable even if statistically justified. Our cold-start analysis reveals a concrete fairness concern: new users experience MAE of 5.15h compared to 0.74h for experienced users, meaning that optimization benefits accrue disproportionately to frequent users.

We recommend several mitigation strategies for production deployment. First, transparent communication: users should receive notifications explaining why their time limits are being adjusted and showing their historical estimation accuracy. Second, opt-out mechanisms: users testing new algorithms or running non-routine workloads should be able to override adjustments with justification. Third, incremental deployment: initial phases should log predictions without enforcement, validating accuracy before enabling automated adjustments. Fourth, equity monitoring: tracking optimization impacts across user groups to ensure that efficiency gains do not exacerbate existing inequalities in computational resource access.

The over-reservation reduction documented here has broader scheduling significance beyond Eagle. While these projections assume Eagle-like overprovisioning rates (which may differ across systems), they illustrate the potential scale of impact. Even if other systems exhibit more moderate overprovisioning (e.g., 20% utilization instead of 6.7%), the absolute efficiency gains at exascale would remain substantial. As HPC systems continue growing in scale, ML-based scheduling optimization that reduces over-reservation represents a practical path toward improved resource utilization.

5.7 Generalizability considerations

An important question is whether models trained on Eagle data would transfer to other HPC systems. Several factors suggest partial generalizability. The dominance of user behavioral features likely transfers, as conservative time estimation appears universal across HPC environments [17, 25]. The utility of temporal features should also transfer, as human activity patterns influence workload submission universally. However, the specific feature importances, optimal hyperparameters, and the magnitude of temporal drift would likely differ across systems with different hardware architectures, user populations, and application mixes. Transfer learning approaches—initializing models with Eagle-learned parameters and fine-tuning on target system data—represent a promising direction for reducing cold-start data requirements on new systems.

The temporal drift we document (71% R^2 degradation) raises important questions about cross-system transfer: if models struggle to generalize across time on a single system, cross-system transfer without adaptation may be even more challenging. Collaborative efforts to share anonymized job trace data across HPC centers, following the model of the Parallel Workloads Archive [28], [18, 19] could accelerate progress toward generalizable runtime prediction systems.

5.8 Future research directions

This work opens several avenues for future investigation:

Online and Continual Learning: The temporal drift documented in our experiments motivates the development of online learning approaches that continuously update models as new jobs complete. Techniques such as incremental decision trees, online gradient boosting [23], and continual learning frameworks could maintain prediction accuracy without full retraining.

Probabilistic Predictions: Developing models that output predictive distributions rather than point estimates would enable risk-aware safety margin selection. Quantile regression forests or conformal prediction could provide calibrated confidence intervals, allowing the scheduler to use smaller margins for high-confidence predictions and larger margins when uncertainty is high.

Application-Aware Modeling: Incorporating application-specific features—executable fingerprints, input file sizes, library dependencies—could address the residual variance unexplained by our current feature set [14, 15]. The ORA system [13] has demonstrated the value of script-level features for runtime prediction.

Multi-System Transfer Learning: Evaluating whether models trained on Eagle transfer to other HPC systems (e.g., Summit, Frontera, Perlmutter) would assess the generalizability of our user behavioral features. Transfer learning approaches that fine-tune Eagle-pretrained models on limited target-system data could reduce cold-start requirements.

Reinforcement Learning for Scheduling: 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 [24].

Production A/B Testing: The definitive validation of our approach requires controlled deployment on a production HPC system, comparing queues with ML-adjusted time limits against standard queues. This would quantify real-world scheduling efficiency gains, user satisfaction, and scheduling throughput improvements.

6 Conclusion

This study set out to predict HPC job runtimes accurately enough to cut energy waste. It succeeded—but the route to that success upended two assumptions common in the literature.

Assumption 1: resource parameters drive prediction. They do not. On 7.3 M real Eagle jobs, user behavioral features (historical runtimes, utilization habits) outweigh every resource parameter in ablation. Runtime prediction is user-behavior calibration, not hardware estimation.

Assumption 2: high R^2 on random splits means deployment-ready. It does not. Temporal evaluation drops R^2 from 0.602 to 0.172, exposing concept drift that static models cannot handle. Prior studies reporting $R^2 > 0.90$ on synthetic data or random splits overstate deployable accuracy by an order of magnitude.

Despite these sobering findings, the practical outlook is positive. Eagle users overestimate so severely (6.7% median utilization) that even drift-degraded predictions with 200% margins still eliminate 64.8% of over-reserved processor-hours. The path to deployment runs through online learning systems that retrain continuously, adapting to the workload shifts we have documented.

We release the experimental code at `https://github.com/rntvargas/hpc-runtime-prediction` to support reproducibility and encourage the HPC community to adopt temporal-split evaluation as the standard for runtime prediction research.

Acknowledgments

We gratefully acknowledge NREL for providing access to the Eagle dataset through the Open Energy Data Initiative (OEDI). We thank the HPC scheduling research community for foundational work upon which this study builds.

Data availability and reproducibility

All data and code required to reproduce the experiments reported in this paper are publicly available:

– **Dataset:** The NREL Eagle Jobs Dataset (11M+ jobs, Parquet format, 241 MB) is available at `https://`

data.openei.org/submissions/5860 under CC BY 4.0 license.

– **Code:** Experimental scripts including data preprocessing, feature engineering, model training, evaluation, and figure generation are available at https://github.com/rntvargas/hpc-runtime-prediction.

– **Environment:** Python 3.11+, scikit-learn 1.3+, pandas 2.0+, NumPy 1.24+. No GPU required; all experiments complete within 30 minutes on a standard multi-core workstation.

– **Random seeds:** All random number generators seeded at 42 for exact reproducibility.

References

[1] D. Duplyakin and K. Menear, "NREL Eagle supercomputer jobs," Open Energy Data Initiative (OEDI), National Renewable Energy Laboratory, 2023. https://data.openei.org/submissions/5860

[2] K. Menear, A. Nag, J. Perr-Sauer, M. Lunacek, K. Potter, and D. Duplyakin, "Mastering HPC runtime prediction: From observing patterns to a methodological approach," in *PEARC '23*, ACM, 2023, pp. 75–85. https://doi.org/10.1145/3569951.3593598

[3] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *ACL 2019*, pp. 3645–3650. https://doi.org/10.18653/v1/P19-1355

[4] G. P. Rodrigo, E. Elmroth, P.-O. Östberg, and L. Ramakrishnan, "Enabling workflow-aware scheduling on HPC systems," in *HPDC '17*, ACM, 2017, pp. 3–14. https://doi.org/10.1145/3078597.3078604

[5] Z. Zhu and P. Fan, "Machine learning based prediction and classification of computational jobs in cloud computing centers," arXiv:1903.03759, 2019. https://doi.org/10.48550/arXiv.1903.03759

[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 HPC jobs," *Appl. Soft Comput.*, vol. 170, p. 112053, 2024. https://doi.org/10.1016/j.asoc.2024.112053

[7] K. Menear, K. Konate, K. Potter, and D. Duplyakin, "Tandem predictions for HPC jobs," in *PEARC '24*, ACM, 2024, pp. 1–9. https://doi.org/10.1145/3626203.3670547

[8] B. Gaikwad, N. A. Simakov, T. Furlani, J. P. White, and A. Patra, "Predictive modeling of HPC job queue times: Improving user decision-making and resource utilization," in *PEARC '25*, ACM, 2025, Article 58, pp. 1–4. https://doi.org/10.1145/3708035.3736067

[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, 2023. https://doi.org/10.3390/en16020890

[10] A. A. Springborg, M. Albano, and S. X. de Souza, "Automatic energy-efficient job scheduling in HPC: A novel SLURM plugin approach," in *SC '23 Workshops*, ACM, 2023, pp. 1961–1969. https://doi.org/10.1145/3624062.3624265

[11] A. Hossain, A. Abdurahman, M. A. Islam, and K. Ahmed, "Power-aware scheduling for multi-center HPC electricity cost optimization," arXiv:2503.11011, 2025. https://doi.org/10.48550/arXiv.2503.11011

[12] K. Menear, J. Acosta, and D. Duplyakin, "Classification of HPC job power consumption using a rich feature set," in *PEARC '25*, ACM, 2025, pp. 1–6. https://doi.org/10.1145/3708035.3736084

[13] H. Liu, Y. Ma, X. Huang, L. Zhang, T. Jia, and Y. Li, "ORA: Job runtime prediction for high-performance computing platforms," in *ICS '25*, ACM, 2025. https://doi.org/10.1145/3721145.3725757

[14] K. Lamar, A. Goponenko, O. Aaziz, B. A. Allan, J. M. Brandt, and D. Dechev, "Evaluating HPC job run time predictions using application input parameters," in *Proc. 17th ACM Int. Conf. Distributed and Event-based Systems (DEBS '23)*, ACM, 2023, pp. 127–138. https://doi.org/10.1145/3583678.3596893

[15] G. E. Gorbet and B. Demeler, "Optimizing UltraScan job scheduling with deep learning-based runtime prediction," in *PEARC '25*, ACM, 2025. https://doi.org/10.1145/3708035.3736016

[16] S. Wang, S. Chen, and Y. Shi, "Utilization-prediction-aware energy optimization approach for heterogeneous GPU clusters," *J. Supercomput.*, vol. 80, no. 7, pp. 9554–9578, 2024. https://doi.org/10.1007/s11227-023-05807-x

[17] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 6, pp. 789–803, 2007. https://doi.org/10.1109/TPDS.2007.70606

[18] M. Tanash, B. Dunn, D. Andresen, W. Hsu, S. Yang, and A. Okanlawon, "Improving HPC system performance by predicting job resources via supervised ma-

chine learning," in *PEARC '19*, ACM, 2019, pp. 1–8. https://doi.org/10.1145/3332186.3333041

[19] M. Tanash, H. Yang, D. Andresen, and W. Hsu, "Ensemble prediction of job resources to improve system performance for Slurm-based HPC systems," in *PEARC '21*, ACM, 2021. https://doi.org/10.1145/3437359.3465574

[20] X. Chen, H. Zhang, H. Bai, C. Yang, X. Zhao, and B. Li, "Runtime prediction of high-performance computing jobs based on ensemble learning," in *Proc. 4th Int. Conf. High Performance Compilation, Computing and Communications (HP3C)*, ACM, 2020, pp. 56–62. https://doi.org/10.1145/3407947.3407968

[21] E. Gaussier, D. Glesser, V. Reis, and D. Trystram, "Improving backfilling by using machine learning to predict running times," in *SC '15: Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis*, ACM, 2015, pp. 1–10. https://doi.org/10.1145/2807591.2807646

[22] S. Madireddy, P. Balaprakash, P. Carns, R. Latham, G. K. Lockwood, R. Ross, S. Snyder, and S. M. Wild, "Adaptive learning for concept drift in application performance modeling," in *Proc. 48th Int. Conf. Parallel Processing (ICPP)*, ACM, 2019, pp. 1–11. https://doi.org/10.1145/3337821.3337922

[23] S. Zrigui, R. Y. de Camargo, A. Legrand, and D. Trystram, "Improving the performance of batch schedulers using online job runtime classification," *J. Parallel Distrib. Comput.*, vol. 164, pp. 83–95, 2022. https://doi.org/10.1016/j.jpdc.2022.01.003

[24] Y. Fan, P. Rich, W. E. Allcock, M. E. Papka, and Z. Lan, "Trade-off between prediction accuracy and underestimation rate in job runtime estimates," in *IEEE Int. Conf. Cluster Computing (CLUSTER)*, IEEE, 2017, pp. 1–10. https://doi.org/10.1109/CLUSTER.2017.11

[25] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely, "Are user runtime estimates inherently inaccurate?," in *Job Scheduling Strategies for Parallel Processing (JSSPP)*, Springer, 2004, pp. 253–263. https://doi.org/10.1007/11407522_14

[26] M. Naghshnejad and M. Singhal, "A hybrid scheduling platform: a runtime prediction reliability aware scheduling platform to improve HPC scheduling performance," *The Journal of Supercomputing*, vol. 76, pp. 8551–8570, 2020. https://doi.org/10.1007/s11227-019-03004-3

[27] F. Chen, "Job runtime prediction of HPC cluster based on PC-Transformer," *The Journal of Supercomputing*, vol. 79, no. 17, pp. 20208–20234, 2023. https://doi.org/10.1007/s11227-023-05470-2

[28] D. G. Feitelson, D. Tsafrir, and D. Krakov, "Experience with using the Parallel Workloads Archive," *J. Parallel Distrib. Comput.*, vol. 74, no. 10, pp. 2967–2982, 2014. https://doi.org/10.1016/j.jpdc.2014.06.013

[29] W. Tang, N. Desai, D. Buettner, and Z. Lan, "Analyzing and adjusting user runtime estimates to improve job scheduling on the Blue Gene/P," in *IEEE Int. Symp. Parallel & Distributed Processing (IPDPS)*, IEEE, 2010, pp. 1–11. https://doi.org/10.1109/IPDPS.2010.5470474

[30] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Job Scheduling Strategies for Parallel Processing (JSSPP)*, Springer, 2003, pp. 44–60. https://doi.org/10.1007/10968987_3